Sensornettverk

Batterisparende aktivering av sensorer

Masteroppgave

ved

Masterutdanning i

informasjons- og kommunikasjonsteknologi

av
Roberth Vestbø

Grimstad, mai 2007

Sammendrag

Batteridrevne trådløse sensornettverk kan dra nytte av intelligente metoder for å spare strøm på batteriene. Denne oppgaven ser på tre forskjellige maskinlæringsalgoritmer for kontroll av strømforbruket, Hensikten med den intelligente kontrollen er å oppnå ønsket tjenestekvalitet, samtidig som sensorenes forbruk av batteri minimaliseres.

I utfordrende miljøer der sensorer kan bli ødelagt eller der radioforbindelsen kan ha vanskelige sendingsforhold, kan man benytte seg av flere sensorer enn man egentlig har behov for. Det er da viktig at de sensorene som er overflødig ikke er aktive, men skrudd av. På den måten kan sensorene spare på batteriet. Det er nettopp den styringen av hvem som skal være aktive og hvem som skal være avslått som er vurdert i denne masteroppgaven.

De tre metodene jeg har vurdert er alle maskinlæringsalgoritmer. Den første metoden er basert på Goore-spill paradigmet og viser seg å være en treg metode. Goore-spill metoden bruker over 5000 iterasjoner for å slå på riktig antall sensorer. Iterasjoner regnes å være like for alle metodene og varer ett sekund. Den andre metoden jeg har jobbet med kalles Stochastic Weak Estimators. Denne metoden viser seg å være en rask og stabil metode, som ikke trenger mer enn 100 iterasjoner for å slå på riktig antall sensorer. Den tredje og siste metoden baserer seg på Bayes teorem, og viser seg å være den raskeste metoden. Den trenger bare 2-3 iterasjoner for å slå på det riktig antallet sensorer.

Metodene er også testet med støy. Støy betyr tap av meldinger fra sensor til basestasjon, og grunnen er som regel vanskelige sendingsforhold. Det viser seg at Stochastic Weak Estimators metoden klarer seg med 0-30 flere iterasjoner selv om man har støy helt opptil 50%. Den Bayesianske metoden trenger ingen flere iterasjoner. Goore-spill metoden ser ikke ut til å bli særlig påvirket av støy på opptil 20 %, men metoden viser kun svake tegn til læring med støy på 50 %. Ellers ser man at støy gir et visst utslag for alle metodene ved at variasjonen øker (selv om middelverdien forblir den samme).

Når det gjelder strømforbruket, så vil metoder som bruker få iterasjoner slik som Stochastic Weak estimators og den Bayesianske metoden, bruke mindre strøm enn metoder som trenger lenger tid for å nå riktig antall sensorer. Det kan også være andre faktorer som vil virke inn på strømforbruket, som for eksempel prosessering i sensorer, sensorer som er mer komplekse enn andre kan for eksempel bruke mer strøm enn de mer enkle utgavene. I denne oppgaven er Goore-spill metoden den metoden som bruker de mest komplekse sensorene.

Innholdsfortegnelse

Figuroversikt

Forord

Denne hovedoppgaven er gjennomført i forbindelse med masterstudiet, retning IKT, ved Høgskolen i Agder våren 2007.

Tittelen på oppgaven er ”Sensornettverk – Batterisparende aktivering av sensorer”. Faglig ansvarlig og veileder har vært Ole-Christoffer Granmo ved Høgskolen i Agder. I tillegg til konstruktiv og god faglig veiledning, har han også ved sin interesse for oppgaven gitt meg den nødvendige inspirasjon i arbeidets forskjellige faser.

Grimstad, 29. mai 2007
Roberth Vestbø

1 Innledning

I dette innledende kapittelet presenteres bakgrunnen for oppgaven og mål for oppgaven beskrevet av hypoteser. Deretter ser vi på begrensninger og forutsetninger for oppgaven, oppgavedefinisjonen og til sist en videre oversikt over de påfølgende kapiteler.

1.1 Bakgrunn

I denne masteroppgaven vil jeg presentere metoder for batterisparende aktivering av sensorer i trådløse sensornettverk. I hovedsak betyr det at man har kontroll over sensorene. Det man ønsker er at den tjenesten som det trådløse sensornettverket tilbyr skal holde et ønsket kvalitetsnivå til enhver tid, med minst mulig forbruk av batteri. Slike tjenester kan være overvåkning i forskjellige forbindelser, som trafikk, vær og temperatur.

Et eksempel kan være at man ønsker å samle inn data fra månen. Anta at man har 1000 sensorer spredt utover et landskap. Man ønsker å vite temperaturen til enhver tid for å se hvordan den varierer i det gitte området. Disse sensorene er utstyrt med batteri. Alle disse sensorene trenger ikke å være aktive til enhver tid for at man skal få målt temperaturen riktig. Det kan tenkes at mange sensorer ligger nær hverandre, og at noen kan være av mens andre er aktive. I dette tilfellet kan vi si det holder at 200 sensorer er aktive samtidig. På den måten kan vi spare strøm på sensorene, ved at 200 er på og 800 er av. De 1000 sensorene bytter da på å være aktive og passive. Samtidig kan man tilføre nye sensorer hvis man ønsker det.

Man kan tilføre nye sensorer ved behov. Det kan være at man mister sensorer på noen områder man prøver å overvåke. Da kan man for eksempel fly over områdene og slippe ut mange nye sensorer spredt over de områdene man ikke lenger kunne overvåke. På den måten kan sensornettverket opprettholdes selv om sensorer går tom for batteri eller blir defekte av andre grunner.

Et problem som melder seg er støy. Signalene kan bli påvirket av støy, noe som kan medføre tap av meldinger. Det kan bety at levetiden for sensorene kan bli kortet ned, for hvis man mister meldinger, så må flere sensorer være aktive for å sende. Dette for å demme opp for de meldingene man mister. Hvis man i dette tilfellet har 10 % meldingstap og ønsker 200 svar fra sensorene, så må 223 sensorer sende svar, derav 22,3 vil falle bort i støy.

Enda et problem kan være tap av sensorer. I et utfordrende miljø kan for eksempel sensorer bli ødelagt. Når noen sensorer faller bort kreves nye tilpasninger av kontrollstrategi, slik at sensorer som tidligere var slått av, nå må slås på.

Som antydet over, kan man redusere den totale energibruken i et sensornettverk ved å kontrollere aktivitetsnivået i nettverket. Dermed kan levetiden til sensornettverket økes. Hvor mye denne effekten kan dras nytte av bestemmes åpenbart av hvilken strategi man benytter for å kontrollere sensornettverket.

I denne masteroppgaven evaluerer jeg en ny og lovende metode for kontroll av energibruken i sensornettverk baserte på såkalte desentralisert læring ved hjelp av Goore-spill. Jeg ser både på hvor raskt metoden finner ønsket aktivitetsnivå og i hvilken grad metoden håndterer støy. Jeg gjør dermed en grundigere evaluering enn hva som er gjort i tidligere arbeider. Samtidig foreslår jeg to helt nye metoder for kontroll av energibruken i et sensornettverk. Jeg baserer meg på to andre teoretiske fundament, nemlig *Stochastic Weak Estimators* og *Bayesiansk estimering*. Anvendelser av de to sistnevnte teoretiske fundamentene har vist seg å være svært vellykket i andre sammenhenger, og det er derfor av interesse å finne frem til nye metoder som gjør bruk av disse fundamentene i sensornettverksammenheng.

Det er spesielt viktig å sammenlikne disse tre metodene mot hverandre. Dette for å se om de har forskjellige kvalitative egenskaper, og om noen skiller seg positivt ut med tanke på å redusere sensorenes batteribruk. Det vil også være interessant å se i hvilken grad de ulike metodene håndterer støy og kommunikasjonssvikt.

1.2 Mål for oppgaven

Jeg har organisert forskningen min ved hjelp av åtte hypoteser. Målet har vært å styrke eller svekke hypotesene ved hjelp av en rekke eksperimenter. Eksperimentene er utført ved å implementere de tre metodene og knytte dem opp mot egnede simuleringer. Hypotesene som jeg skal vurdere er listet under:

1.) *Metoden basert på Goore-spill kan brukes til å minimalisere batteriforbruk ved en gitt ønsket oppløsning*. Denne metoden ble foreslått og evaluert i artikkel [1], men eksperimentene rapportert i artikkelen er noe begrensede da de for eksempel ikke dekker kommunikasjonsstøy. Det er derfor ønskelig å gjøre en grundigere evaluering for å tydeligere avdekke teknikkens styrker og svakheter.

2.) *Metoden Stochastic Weak Estimators kan brukes til å minimalisere batteriforbruk ved en gitt ønsket oppløsning.* Goore-spill er basert på desentralisert læring. Desentralisert metoder lærer gjerne saktere enn sentraliserte metoder. Målet med å utvikle en metode basert på Stochastic Weak Estimators er å kunne estimere antall aktive sensorer til enhver tid ved hjelp av sentralisert læring, for dermed å oppnå hurtigere læring enn hva man oppnår ved Goore-spill.

3.) *Metoden Bayesiansk estimering kan brukes til å minimalisere betteriforbruk ved en gitt ønsket oppløsning*. Her gjelder det samme prinsippet som påstand 2.

4.) *Metoden Goore-spill kan håndtere støy og kommunikasjonsvikt/ feil.* Et kjennetegn ved sensornettverk er tap av meldinger på grunn av støy (f.eks. atmosfæriske forhold). En metode som skal brukes til å kontrollere et sensornettverk må derfor håndtere denne typen støy. Goore-spill teknikken er ikke tidligere evaluert for situasjoner hvor meldinger kan gå tapt. Om Goore-spill teknikken ikke fungerer under slike forhold vil den være av begrenset nytte i praksis.

5.) *Stochastic Weak Estimators kan håndtere støy og kommunikasjonsvikt/ feil*. Her gjelder det samme prinsippet som i påstand 4.

6.) *Bayesiansk estimering kan håndtere støy og kommunikasjonsvikt/ feil*. Her gjelder det samme prinsippet som i påstand 4.

7.) *Stochastic Weak Estimators er bedre metoder enn Goore-spill baserte metoder*. Ut fra generelle egenskaper ved desentralisert og sentralisert læring, er det grunn til å tro at en løsning basert på sentralisert læring (Stochastic Weak Estimators) vil skalere bedre enn en løsning basert på desentralisert læring (Goore-spill). Jeg ønsker å undersøke om dette også gjelder for sensornettverk ved å utvikle metoder basert på Stochastic Weak Estimators for denne problemstillingen.

8.) *Bayesiansk estimeringer er bedre metoder enn Goore- spill baserte metoder.* Ut fra de samme generelle egenskaper som påstand 7. Jeg ønsker å undersøke om dette også gjelder for sensornettverk ved å utvikle metoder basert på Bayesiansk estimeringer for denne problemstillingen.

1.3 Begrensninger og forutsetninger

Batterienes og sensorenes teknologi generelt har ingen betydning, men batteriets levetid og sensorenes meldinger er viktig. Det skal også være mulig å skru sensorene av og på.

Nettverkets trådløse teknologi er heller ikke viktig, men det forutsettes at det virker, bortsett fra at støy og kommunikasjonsvikt kan inntreffe.

Det vil ikke bli utviklet en fabrikk ferdig løsning i denne oppgaven, men prototyper som vil beskrive virkeligheten.

Det blir heller ikke gjort noen tester i ekte sensor nettverk, men simulert ved hjelp av fabrikkerte data i prototyper utviklet i Java.

Når det gjelder utviklingsverktøy og programmeringsspråk er det ikke satt noen krav fra oppdragsgiver. Det vil bli brukt Java til å programmere prototypene, og Microsoft Excel til å tegne grafer som beskriver resultatene.

Løsningen blir kun basert på problemstillingen, og vil ikke løse flere problemer i samme problemområde.

1.4 Oppgavedefinisjonen

"Målet med denne oppgaven er å finne ut hvordan et distribuert intelligent system kan brukes til å kontrollere aktiveringen av sensorer i et batteridrevet sensornettverk. Hensikten med den intelligente kontrollen er å oppnå ønsket tjenestekvalitet, samtidig som sensorenes forbruk av batteri minimaliseres. En viktig del av oppgaven vil være å lage en prototype, og ved hjelp av simulering, sammenligne ytelsen til prototypen med ulike state-of-the-art teknikker."[4].

1.5 Oppgavens oppbygning

I tillegg til innledningen er oppgaven inndelt i 5 hovedkapitler:

Kapittel 2: Gir en innføring i begrepet sensornettverk og dets komponenter som basestasjon og sensorer

Kapittel 3: Beskriver grundig de tre metodene som skal testes mot hverandre.

Kapittel 4: I dette kapittelet beskrives eksperimentoppsettet, og resultatene blir illustrert ved hjelp av grafer.

Kapittel 5: I dette kapittelet drøftes hypotesene i lys av eksperimentresultatene.

Kapittel 6: Her konkluderes oppgaven.

2 Hva er et trådløst sensornettverk?

I dette kapittelet vil jeg gi en grunnleggende innføring i trådløse sensornettverk, fordi det er sentralt i denne oppgaven. Jeg gir en kort oversikt og en kort beskrivelse av de mest sentrale enhetene i sensornettverk. Det presenteres mer teori spesifisert mot hver enkelt metode som skal testen i denne masteroppgaven i kapittel 3 som omhandler metodene.

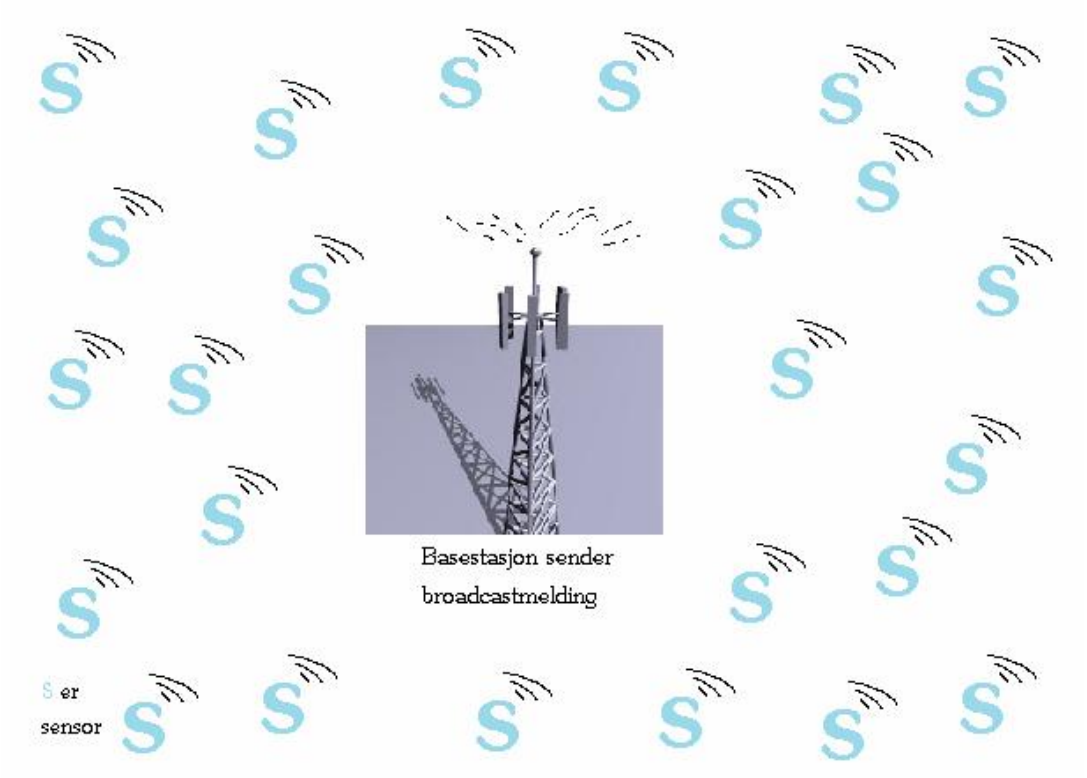
2.1 Oversikt

Et trådløst sensornettverk består av distribuerte enheter som er utstyrt med sensorer for overvåkning av egenskaper ved fysiske eller virtuelle miljø. Dette kan være egenskaper som temperatur, lyd, vibrasjon, trykk, bevegelse eller forurensing. Målingen av egenskaper utføres gjerne på flere forskjellige steder, for eksempel spredt over et geografisk område.

Utviklingen av trådløse sensornettverk var i utgangspunktet et militært foretagende. Et viktig mål var å kunne overvåke kamparenaer på en mer fleksibel og sikker måte. I dag brukes slike trådløse sensornettverk også til mange sivile formål, som for eksempel trafikkontroll.

Sensorene er som regel utstyrt med en enhet for trådløs kommunikasjon, en mikrokontroller, og en energikilde som ofte er et batteri. En slik sensor kan variere i størrelse og pris, tatt i betraktning behovet som skal dekkes.

I vårt scenario mottar sensorene ***broadcast*** meldinger fra en basestasjon (en melding som kringkastes til alle sensorer). Videre sender sensorene svar til basestasjonen med målingene sensorene har utført. Merk at basestasjonen ikke kan sende meldinger til hver enkelt sensor individuelt, fordi basen vet ikke sensorenes identitet og heller ikke antallet på hvor mange sensorer som er tilgjengelig.



Figur 1: Illustrerer trådløst sensornettverk. S står for sensor.

Legg også merke til at ressurser som batteri, minne, CPU hastighet og båndbredde gjerne utgjør en kritisk begrensing, ved bruk av trådløse sensornettverk. Som jeg utdyper i følgende seksjoner danner denne begrensningen utgangspunktet for dette masteroppgavearbeidet.

2.1.1 Basestasjonen

I min tilnærming er en basestasjon en enhet som mottar og analyserer data sendt fra sensorene i sensornettverket. Basestasjonen kan også sende *broadcast*-meldinger til sensorene som gir dem instrukser, blant annet for å instruere dem.

2.1.2 Sensorene

Med sensorer mener jeg en enhet som kan sende og motta meldinger i trådløse nettverk, og som samler inn data som den sender til en basestasjon. Sensorene kan skrus av og på, og er i dette tilfellet batteridrevet. Bortsett fra disse typiske egenskapene vil sensorer kunne variere mye. Noen sensorer vil for eksempel kunne være utstyrt med minne (noe man behøver i en av metodene som skal testes).

2.1.3 Samspillet mellom enhetene

Basestasjonen vil være med å påvirke sensorer til å ta riktige valg, i henhold til hva man ønsker. Sensorene jobber med å samle inn data for den tjenesten som utføres og sender disse data til basestasjonen. Ikke alle sensorene gjør dette, noen har fått beskjed om at de skal være avslått. Denne påvirkningen som basestasjonen har, kan varierer for hver enkelt metode som er brukt.

Her kan man skille mellom sentralisert kontroll og desentralisert kontroll. Hvis basestasjonen tar seg av avgjørelsene og er den intelligente enheten i sensornettverket, kan man si at man har sentralisert kontroll. Da vil sensorene bare lystre basens instrukser, og ha liten innvirkning på resultatet selv. Derimot hvis sensorene tar avgjørelser og er intelligente, har basen mindre innflytelse, og man kan si at man har desentralisert kontroll. For å summere kan vi man si at påvirkningen av basen er større i en sentralisert metode enn en desentralisert metode.
Vi vil se eksempler på begge scenarioene når metodene beskrives i kapittel 3.

Kommunikasjonen mellom sensorene og basen er trådløs (se figur 1), og man må derfor beregne en høyere forekomst av støy, enn om det var ledninger mellom dem. Støy kan påvirke samspillet mellom sensorene og basen, fordi enkelte meldinger går tapt. Det vil i praksis bety at hvis basen ikke mottar alle meldingene som den ønsket, vil den sende instrukser som krever flere svar neste runde.

3 Metodene

Jeg skal teste tre forskjellige intelligente metoder som skal sørge for at sensorer i sensornettverk bevarer meste mulig av sin batterikapasitet. Samtidig skal sensorene sammen klare å sende mange nok meldinger til basen for å sikre at den kan opprettholde kvaliteten på tjenesten den skal utføre (QoS – Quality of Service), i dette tilfellet romlig oppløsning.

Med dette eksempelet vil jeg forklare hva som menes med oppløsning i denne sammenhengen. Noen er kanskje vant til at oppløsning referer til hvor mange piksler et skjermbilde inneholder. Men i dette tilfellet ser man på oppløsning som et areal, der man må ha nok punkter som er aktive for å opprettholde den tjenesten som blir utført. Det kan sammenliknes ved å se film på en TV, der man bare har halvparten av pikslene aktive, og resten passive. Man vil da fortsatt kunne danne et bilde, og hvis bildet er tilfredsstillende kan man altså på denne måten spare strøm. Videre ser man at det ikke er hensiktsmessig at to aktive punkter ligger veldig nær hverandre. For eksempel hadde skjermbilde lignet en klump i det ene hjørnet, ville det ikke vært brukbart.

I korte trekk er målet vårt å kunne skru sensorene av og på slik at man sparer strøm, men likevel opprettholder en spesifisert oppløsning.

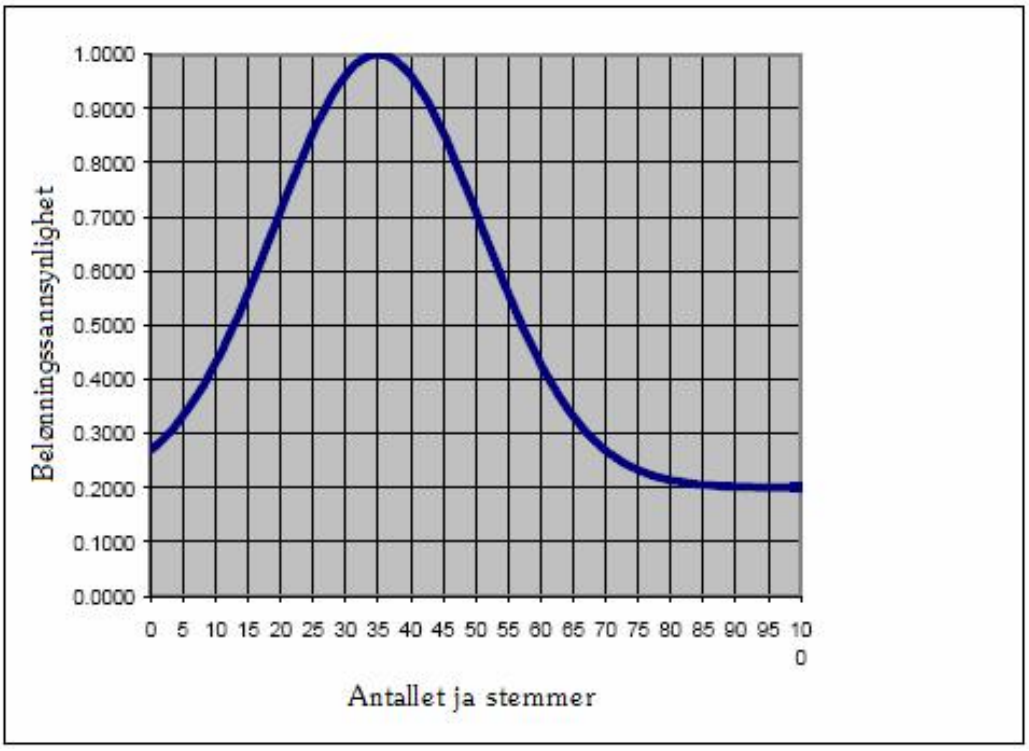
3.1 Goore-spill paradigmet

Den første metoden jeg undersøker er basert på et såkalt Goore-spill i artikkelen [1]. I denne metoden vil hver sensor inneholde en såkalt Tsetlinautomat med *N* tilstander. Jeg gir derfor først en kort innføring i Goore-spill paradigmet og Tsetlinautomater.

3.1.1 Goore-spill Paradigmet

For å forklare paradigmet bruker jeg det samme eksempelet som man finner i [1].

”Se for deg at man har mange spillere og ingen av dem er bevisst hverandre. I tillegg har man en dommer. Dommeren spør hver spiller hvert sekund om å stemme *ja* eller *nei,* deretter teller han opp *ja* og *nei* svarene. En belønningssannsynlighet r = r (k) blir generert som en funksjon av tallet k av spillere som svarte *ja*. Det er vanlig å forutsette at 0<=r (k)<=1 og en typisk



Figur 2 hentet fra [1] og gir en beskrivelse av en typisk belønningsfunksjon.

Hver spiller, uansett hva de stemte, får hver for seg enten en belønning (med sannsynlighet *r*) eller en straff (med sannsynlighet *1-r*). La oss for eksempel anta at ved en runde *i* spillet stemte k_i spillere *ja*. Belønningssannsynligheten ville da vært $r(k_i)$. Dermed blir hver spiller belønnet med sannsynligheten $r(k_i)$. Merk at man i dette eksempelet oppnår den høyeste sannsynligheten for belønning når 35 spillere stemmer ja. Vi lar *k** angi det optimale antallet *ja*-stemmer. Man kan da vise følgende: Uansett hvor mange spillere det er, kan man lage de på en måte slik at ca *k** av spillerne (i dette tilfellet 35) stemmer *ja* etter mange nok runder. Denne Goore-spill egenskapen gjelder for nesten alle funksjoner uansett om den er, eller ikke er sammenhengende eller har flere topper. Merk videre at den individuelle automaten ikke vet om nummeret *k* eller belønningsfunksjonen *r(k).* Dessuten opererer hver spiller alene på en grådig måte ved at den forsøker å velge handling som gir den best utbytte. Grådighet har en tendens til å påvirke resultater på en uforutsigbar måte. I dette tilfellet derimot, vil man ikke få denne effekten fordi spillerne ikke har noen mulighet til å forutse de andre spillernes atferd. Spillerne må derfor prøve å feile og bare gjenta de handlingene som hver enkelt spiller anser som mest lønnsom for dem."

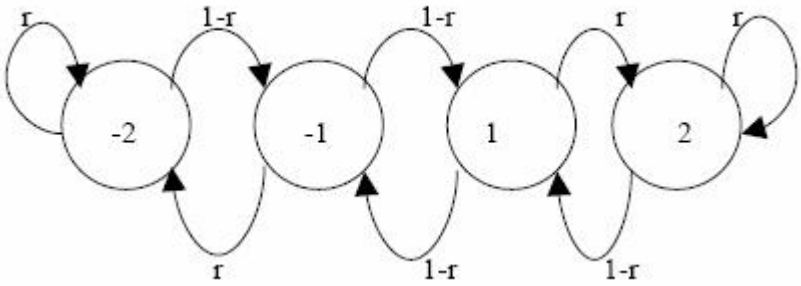
Jeg vil bruke dette eksempelet senere i rapporten, men aller først ser jeg nærmere på Tsetlinautomater, som danner grunnlaget for å maksimalisere *r(k)*. Minner om at ved å maksimalisere *r(k)* som er belønningsfunksjonen, så oppnår man forhåpentligvis etter noen runder ønsket antall svar.

3.1.2 Hva er Tsetlinautomater?

I sin enkleste form skal en Tsetlinautomat velge mellom to handlinger. Når en handling er valgt, forventer automaten å få en straff eller belønning, avhengig av om valget var fornuftig eller ei. Målet er å finne handlingen som gir størst sannsynlighet for å motta belønning ved å lære fra responsene som gis på handlingene. På den måten kan automaten lære av sine tidligere avgjørelser.

En Tsetlinautomat har et gitt antall tilstander *N*, som er med å påvirke hvor raskt automatene lærer og hvor sikre de blir på avgjørelsene sine. Hvis en automat blir veldig sikker på sine avgjørelser kan det ta lang tid før den ombestemmer seg. Derfor må man finne *N* slik at systemet kan reagere raskt, men samtidig være stabilt.

Når jeg snakker om responsen som Tsetlinautomatene får, så kan det være enten belønning eller straff. Feil avgjørelse gir straff, og riktig avgjørelse gir belønning. I vårt tilfelle gis responsen av basestasjonen, som sender en belønningsprosent til alle sensorene. Sensorene regner selv ut om de fikk belønning eller straff. Sensorene trekker altså et tilfeldig tall mellom 0 og 1, og sammenlikner det med belønningsprosenten. Trekker de et høyere tall enn belønningsprosenten får de straff, derimot trekker de et mindre tall får de belønning. Dette gir mening fordi når metoden nærmer seg ønsket antall svar, så bli belønningsprosenten høyere og gir mange belønning slik at man stabiliserer seg rundt riktig antall svar fra sensorene.



Figur 3: hentet fra [1] og beskriver tilstander i Tsetlinautomater. Størrelse N=2.

Tsetlinautomater støtter læring fordi hver enkelt sensor gis et eget minne, slik at de kan huske resultatet av sine tidligere avgjørelser.

Nå vil jeg fortsette på eksempelet fra forrige kapittel 3.1.1 Goore-spill hentet fra [1], for å gi en bedre forståelse av hva en Tsetlinautomat er.

”Spesifikt assosieres hver spiller *j* med en Tsetlinautomat M_i. Den endelige tilstandsautomaten representerer spillerens minne. Minnet er en rekke av tilstander hvor den totale størrelsen på minnet i dette eksempelet er 2N, som illustrert i Figur 3 over. Hvis man starter med tilstanden lengst til venstre, nummererer vi tilstandene fra *–N* til *-1*, så fra *1* til *N*. For Tsetlin-automaten fra figuren er *N* lik *2*.

Merk at rekken skilles i en venstre halvdel (med negative tilstandsnummer) og en høyre halvdel (med positive tilstandsnummer). Spilleren er kun tillatt å være i en tilstand om gangen. Spilleren kan kun flytte seg til en tilstand som er tilstøtende (til høyre eller til venstre i rekken). Hvis tilstanden er *N* (Altså tilstanden helt ytterst til høyre), kan overgangen kun være *N-1* og *N* (en løkke tilbake til seg selv). Et liknende tilfelle eksisterer når tilstanden viser seg å være *–N* (altså tilstanden helt ytterst til venstre).

En spiller stemmer *ja* når den er i en positiv nummerert tilstand, og *nei* når den er i en negativ nummerert tilstand. Når spilleren er i en negativ nummerert tilstand beveger den seg mot venstre hvis den blir belønnet av dommeren og mot høyre når den blir straffet. Det tilsvarende er tilfellet når spilleren befinner seg i en positiv nummerert tilstand. Spilleren beveger seg da mot høyre når den blir belønnet av dommeren og mot venstre når den blir straffet. Med andre ord ”mot midten søkende” oppførsel ved straff og ”mot kanten søkende” oppførsel ved belønning. Med dette oppsettet har det blitt bevist at Goore-spill egenskapen er gyldig.”

Kort forklart er det slik at hvis en sensor blir belønnet ofte, blir den sikrere i valget sitt, og det tar lenger tid for at den skal bytte mening. Derimot de sensorene som ligger i en tilstand nær midten bytter ofte mening.

3.1.3 Goore-spill Paradigmet Metoden

Dette er en metode som kan klassifiseres som desentralisert, ettersom det er sensorene som lærer og tar valg ved å lære av tilbakemeldinger fra basen. Sensorene baserer seg på valg de har gjort tidligere og er utstyrt med minne.

Metoden er bygget opp slik at først opprettes et gitt antall automater, med et gitt antall tilstander. Dette er de sensorene som er med i simuleringen. Deretter velger sensorene om de vil være aktive eller passive. Det betyr at de velger seg en helt tilfeldig tilstand for automaten sin, og det er enten 1 eller -1 (se figur 3).

Basen teller deretter opp hvor mange meldinger som den har fått av sensorene. Deretter beregner basen sannsynligheten for belønning ved bruk av følgende uttrykk hentet fra [1] der belønningssannsynligheten er *p*:

$$p = 0.2 + 0.8e^{-0.002*(antallSvar - ønsketAntallSvar)^2}$$

Deretter deles belønningen og straffen ut til sensorene med sannsynligheten *p*. Sensorene oppdaterer sine tilstander, og stegene gjentas flere ganger.

Pseudokode:
Opprett et antall automater med ønsket antall tilstander hver
Utfør et gitt antall runder
    Registrer svar fra automatene
    Tell antall svar
    Regn ut sannsynlighet p for belønning
    For hver automat:
        Gi belønning med sannsynlighet p

Vi ser i seksjonene over at antall svar brukes for å beregne *p*. Det er da viktig å merke seg at evt. støy vil virke inn på antall svar å dermed påvirke *p*. Jeg undersøker om Goore-spill klarer å håndtere støy i kapittel 4 – resultater. Det er også som en del av å vurdere påstand 4 i hypoteseoversikten i kapittel 1.2.

3.2 Stochastic Weak Estimators

3.2.1 Overordnet beskrivelse

Denne metoden er konseptuelt sett mindre komplisert enn Goore-spill paradigmet. Den baserer seg på en sentralisert styring der det er basen som er den lærende enheten i stedet for sensorene.

Sensorene i denne metoden er svært enkle. Basen broadcaster en verdi *p.* Sensorene slår seg deretter på med sannsynlighet *p* ved hjelp av en generator for tilfeldige tall. Merk at en slik enkel sensoralgoritme sannsynligvis vil virke positivt inn på strømforbruket til sensorene, men dette blir ikke testet i denne masteroppgaven. Vi fokuserer først og fremst på strømforbruk knyttet til sending av meldinger fra sensor til base.

Målet for basen er dermed å finne en *p*-verdi som gir ønsket antall svar fra sensorene i snitt over tid. Jeg ønsker med andre ord at læringen kun skal skje i basen. Det foregår ved at forholdet mellom det mottatte resultatet og det ønskede resultatet for hver runde brukes til å justere *p*. På denne måten blir påvirkningen av sensorene runde for runde justert i forhold til resultatene som mottas.

3.2.1 Algoritmen

Algoritmen for sensorene er svært enkel. En runde *t* begynner med at sensorene velger om de skal være aktive eller passive. Dette skjer ved at de velger en tilfeldig verdi mellom *0* og *1* og sammenligner den med tallet *p*, som er distribuert av basen. Hvis den tilfeldige verdien er mindre enn *p* vil sensoren være aktiv (slå seg på), hvis den er større vil den være passiv (slå seg av).

Basen teller opp antallet svar som mottas fra aktive sensorer. Basert på resultatet forsøker den så å finne et sikrere estimat *n(t)* på det totale antallet sensorer enn det den hadde i forrige runde, dvs. *n(t-1)*. Det nye estimatet *n(t)* finner den ved hjelp av følgende uttrykk:

$$n = (\alpha * AntattAntallSensorer) + ((\frac{AntallSvar}{p}) * (1 - \alpha))$$

α er en vekt som bestemmer hvor mye *n(t-1)* fra forrige runde skal påvirke nytt estimat *n(t)*. Det blir altså beregnet en ny *n(t)* ut ifra forholdet mellom antall svar og prosenten *p* som ble distribuert. Hvor fort tilnærmingen skjer, bestemmes av α. Så det er α som kontrollerer læringshastigheten kontra læringsnøyaktigheten. α er satt til 0,95 i denne oppgaven.

p regnes deretter ut ved hjelp av uttrykket:

$$p = \frac{ønsketAntallSvar}{n}$$

p distribueres så til sensorene. Sensorenes valg mellom å svare eller ikke å svare påvirkes av sannsynligheten *p* for å svare og sannsynligheten for at de påvirkes av støy. Deretter fortsetter metoden fra begynnelsen igjen.

Pseudo-kode for metoden:
Opprett et gitt antall sensorer
Utfør dette et gitt antall runder:
    Tell antall svar fra sensorene
    Regn ut antatt antall sensorer
    Regn ut ny sannsynlighet p for at sensorene skal svare
    Distribuer p

Resultatene ser vi i kapittel 4.

3.3 Bayesiansk estimering

For å forstå denne metoden må man vite litt om Bayes regel, Bernoulli-forsøk, Binomisk fordeling og normalisering. Jeg gir derfor her først en oversikt over disse konseptene før jeg presenterer selve metoden.

3.3.1 Bayes regel

Denne definisjonen er hentet ifra [3].

"Bayes' teorem (også kjent som Bayes' regel eller Bayes' lov) er et resultat innenfor læren om sannsynlighetsteori, som relaterer betinget og uordnet sannsynlighetsfordelinger av tilfeldige variabler. I noen tolkninger av sannsynlighet bestemmer Bayes' teorem hvordan man oppdaterer eller revurderer påstand i lyset av nye bevis.

Sannsynligheten for en hendelse A betinget en annen hendelse B er generelt sett forskjellig fra hendelsen B betinget hendelsen A. Men det er et annet bestemt forhold mellom de to, og Bayes' teorem gir en forklaring av det forholdet.

For å utlede teoremet begynner jeg med definisjonen av betinget sannsynlighet. Sannsynligheten for hendelsen A gitt hendelsen B er:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

Og omvendt, sannsynligheten for hendelsen B gitt hendelsen A er:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}.$$

Om man skriver om og kombinerer disse likningene får man:

$$P(A|B)\,P(B) = P(A \cap B) = P(B|A)\,P(A).$$

Denne hjelpesetningen er noen ganger kalt produktregelen for sannsynligheter. Ved å dele begge sider med Pr(B), forutsetter at det ikke er null, får vi Bayes' teorem:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}."$$

3.3.2 Bernoulli-forsøk

Definisjonen og eksemplene er hentet fra [2] side 86.

Man har n Bernoulli-forsøk dersom:

”1) Hvert forsøk har bare 2 utfall:

J eller *N* (*J* for *Ja* og *N* for *Nei*).

2) Sannsynligheten for positivt utfall (*J*) er lik i hvert eksperiment:

$$P(J) = p.$$

3) Utfallene av de enkelte forsøkene er uavhengig av hverandre.

Jeg innser at *n* Bernoulli-forsøk tilsvarer en urnemodell der Jeg har trekning med tilbakelegging blant *n* lapper der *np* av dem er merket *J* og $n(1-p)$ av lappene er merket *N*.
Noen eksempler på konkrete situasjoner der Bernoulli-forsøk kan være en rimelig modell, er følgende:

• *n* myntkast.

• Teste en medisin på *n* «tilfeldige» forsøksdyr og måle hvorvidt reaksjonen er positiv (*J*) eller negativ (*N*).

• *n* lodd i pengelotteriet.

• *n* tilfeldig utfylte Lotto-kuponger.”

I vårt tilfelle dreier det seg selvsagt om *n* sensorer som enten sender en melding til basestasjonen (sannsynlighet *p*) eller lar vær å sende (sannsynlighet *1-p*).

3.3.3 Binomisk fordeling

Denne definisjonen er hentet fra [2] s 86-87.

"Den binomiske fordeling (også kalt binomialfordelingen) kommer til anvendelse når det er rimelig å si at en statistisk undersøkelse består av n Bernoulli-forsøk (binomisk forsøksrekke):

Vi definerer nå den stokastiske variabelen, *X*, som følger:

X = antall positive utfall (*J*) av *n* Bernoulli-forsøk

Fordelingen, *f*(*x*), til *X*, vil da være det jeg kaller en binomisk fordeling med parametere *n* og *p*:

Binomisk fordeling Bino(*n*,*p*)

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x = 0, 1, 2, \ldots, n$$

der *x* = antall positive utfall (*J*) av *n* Bernoulli-forsøk og *p* = *P*(*J*) i hvert forsøk."

3.3.4 Normalisering

Normalisering i denne metoden betyr at alle elementer i et array summeres og deles på summen, slik at den nye totale summen av alle elementene i arrayet er 1.

3.3.5 Metode basert på Bayesiansk estimering

I denne metoden vet man ikke hvor mange sensorer som er tilstede, men man vet hva som er det maksimale antallet sensorer. Det vil si at man for eksempel kan ha en maks ramme på 1000 sensorer, mens det i virkeligheten kun er 100 sensorer tilgjengelig. Det er hvor mange sensorer som faktisk er tilgjengelige denne metoden vil prøve å beregne. Når det faktiske antallet tilgjengelige sensorer er kjent, kan man finne og sende ut en sannsynlighet *p* til sensorene på samme måte som i metode Stochastic Weak Estimators.

I utgangspunktet kan vi se for oss at det er like sannsynlig å trekke ut en hvilket som helst *m*. I dette tilfellet så er *m* maks antall sensorer. Anta at man har en fordeling P_n med *m 1/m*-elementer. Slik at summen av P_n er *1*, slik vi så i kapittelet 3.3.4 om normalisering. Om vi for eksempel har en *m = 10,* som betyr en maks ramme på 10 sensorer, vil P_n ha *10* elementer: [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

Som første ledd i en runde trekker man en tilfeldig *n* fra P_n. Denne *n*'en representerer en mulig situasjon og sannsynligheten for denne situasjonen er $P_n[n]$. Deretter finner man *p* ved formelen:

$$p = \frac{Q}{n}$$

,der Q er ønsket antall svar.

Videre sender man p ut til sensorene og teller antall svar. Med disse opplysningene kan man regne ut den binomiske fordelingen ved hjelp av formelen fra kapittel 3.3.3 Binomisk Fordeling:

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x = 0, 1, 2, \ldots, n$$

Denne Binomiske Fordelingen multipliseres med P_n fordelingen, og etterpå normaliseres P_n som da er klar for neste runde, der man begynner med å trekke en ny tilfeldig *n*, sender ut *p* osv.

Pseudo-kode for metoden:
Opprett P_n med *m* elementer
Utfør hver runde:
- Velg tilfeldig *n*
- Beregner *p* og sender *p* til sensorene $0<p<1$
- Registrer antall svar
- Regn ut den binomiske fordelingen
- Multipliser P_n med den binomiske fordelingen
- Normaliser den nye P_n

Støy har sin innvirkning på antall svar i denne metoden på samme måte som i metode Stochastic Weak Estimators. Hvis et tilfeldig tall er mindre enn sendt ut *p* og et tilfeldig tall er større enn støy, får jeg et svar. Dette kan også sies å være en sentralisert metode, og resultatene vises i kapittel 4.

4 Eksperimentoppsett og resultater

I denne delen forklarer jeg hvordan testingen av metodene ble gjennomført, og hvilke nøkkelverdier jeg brukte. Resultatene blir også grundig beskrevet og illustrert ved hjelp av grafer. Først presenteres eksperimentoppsettet.

4.1 Eksperimentoppsett

Programvare som ble brukt under testing:

-Microsoft Windows XP sp2 MediaCenter edition
-Eclipse SDK
-Microsoft Excel 2003

I denne oppgaven er metodene som er beskrevet i kapittel 2 skrevet i Java. jeg brukte Eclipse SDK som programmeringsverktøy, og til å kjøre kildekoden. Når man kjører metodene får man ut de tallene som MS Excel baserer grafene sine på. Det er denne fremgangsmåten som blir brukt i alle simuleringene som er gjort i denne oppgaven.

Når jeg skal teste de forskjellige metodene bruker jeg antall iterasjoner som tidsenhet. En runde i en metode vil være en iterasjon. Man forsøker å bestemme hvilken metode som trenger minst iterasjoner for og komme frem til ønsket antall sensorer. En av fordelene med å gjøre det slik er at maskinytelsen ikke har betydning for resultatene, og vi får et meget nøyaktig bilde av metodene.

I alle metodene måtte jeg sette noen parametere. Før forsøkene ble gjennomført ble alle metodene stilt inn med de samme parametrene for å kunne sammenlignes senere.

- Antall sensorer ble satt til 100.
- Ønsket antall svar ble satt til 35,
- Antall iterasjoner ble satt til 5000 på metoden Goore-spill Paradigmet
- Antall iterasjoner ble satt til 1000 på de to andre metodene, ettersom at det viste seg at de var mye raskere til å oppnå ønsket antall svar.
- Støy ble satt til 0 %, 5 %, 10 %, 20 % og 50 %.

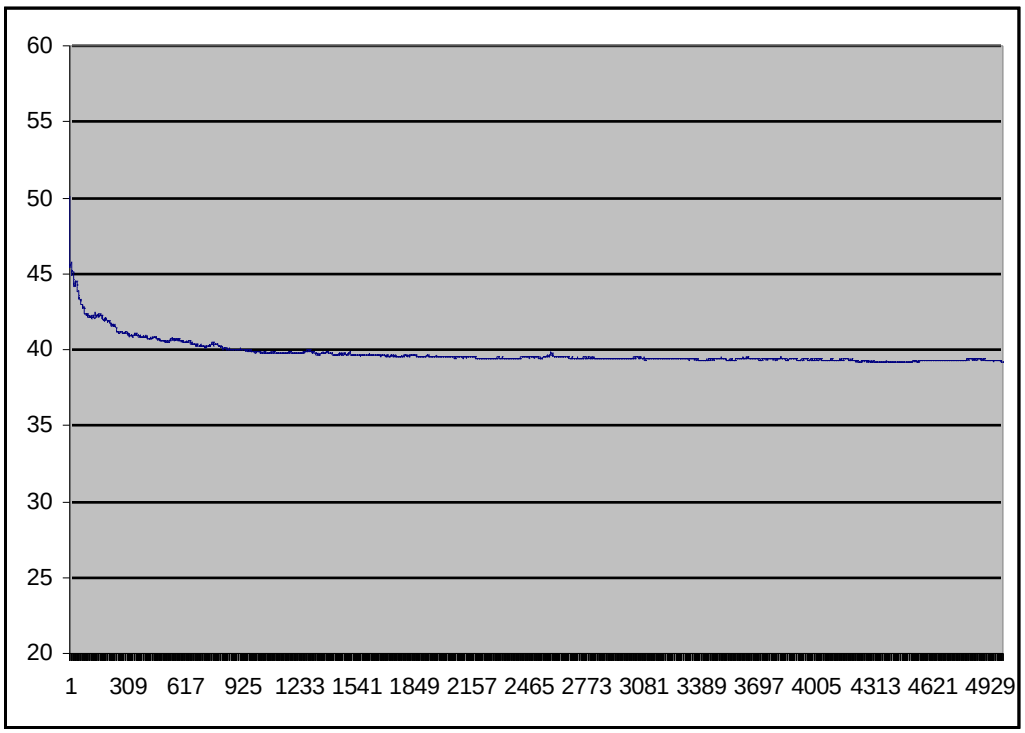
Støy betyr tap av meldinger fra sensor til basestasjon, grunnen er som regel vanskelige sendingsforhold. I praksis vil det bety at i en situasjon der man har 100 sensorer som svarer og 10 % meldingstap, vil man få 90 svar. Resten av svarene kommer ikke frem.

Hver metode blir simulert 100 ganger og deretter tar man gjennomsnittet av alle disse simuleringene for å få et mer nøyaktig og stabilt bilde av metodene. Dette blir utført for hvert enkelt støynivå og danner grunnlaget for resultatene i kapittel 4.2

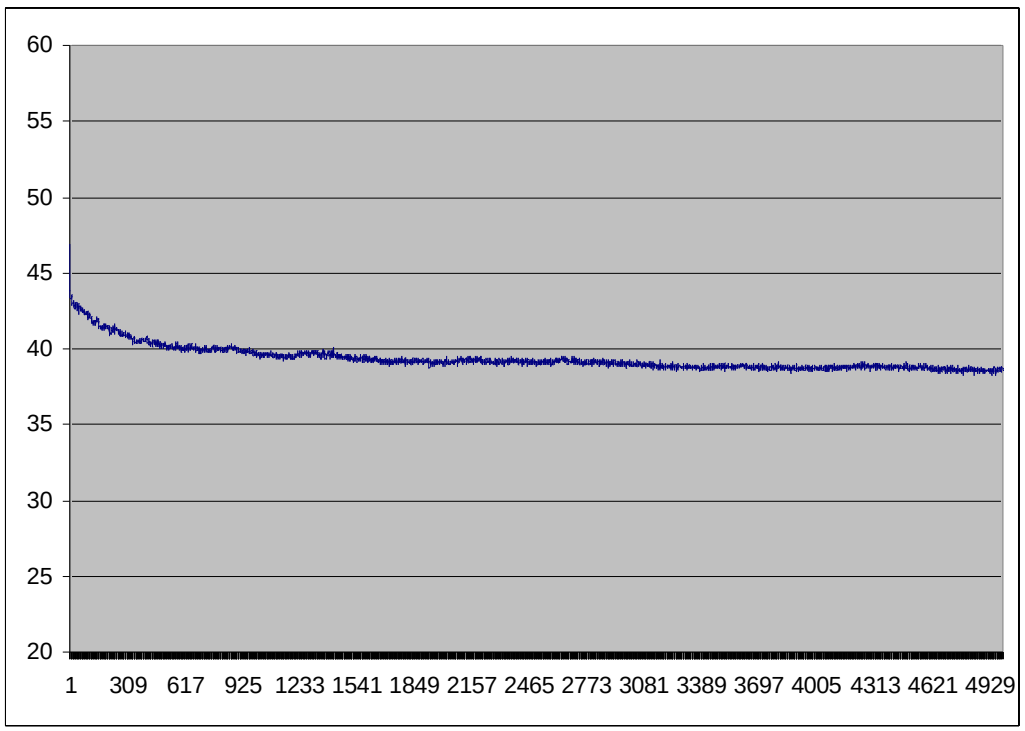
4.2 Resultater

I dette kapittelet presenterer jeg resultatene for alle metodene, med de forskjellige støy scenarioene. Husk at det er 35 på y-aksen som er ønsket antall svar. X-aksen beskriver antall iterasjoner.

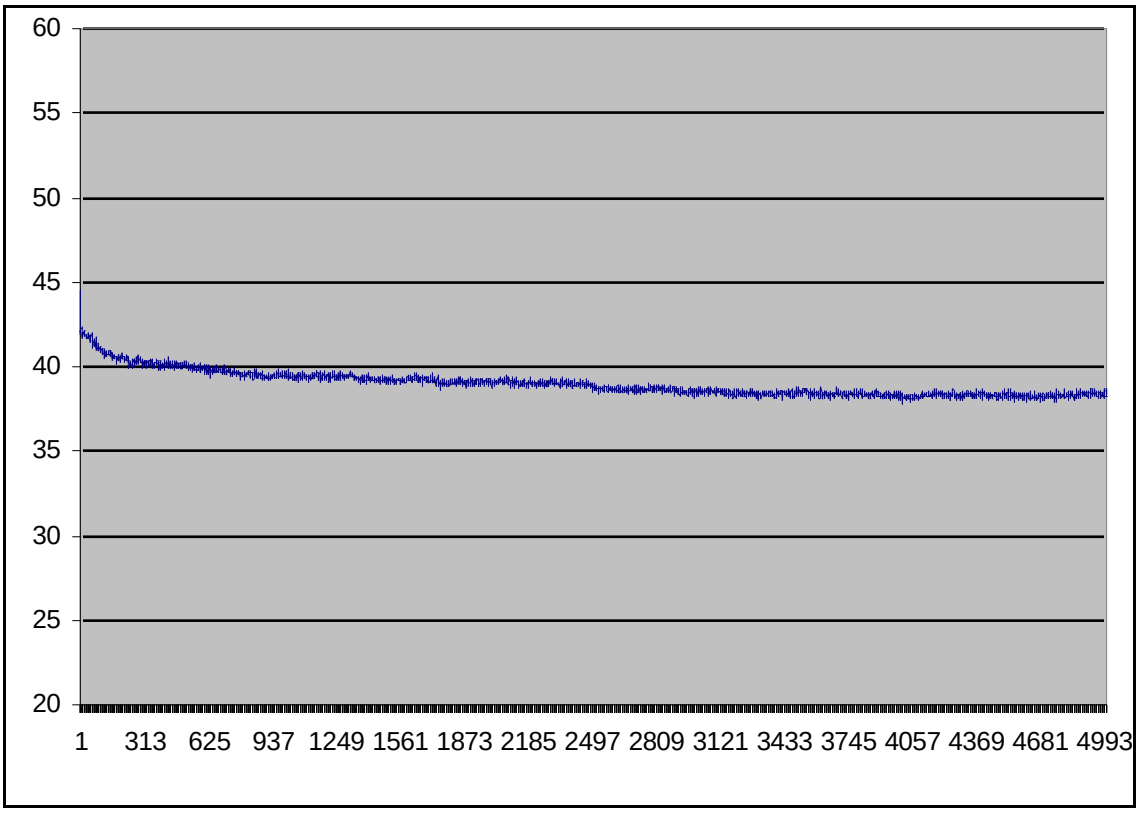
4.2.1 Goore-spill baserte metoden



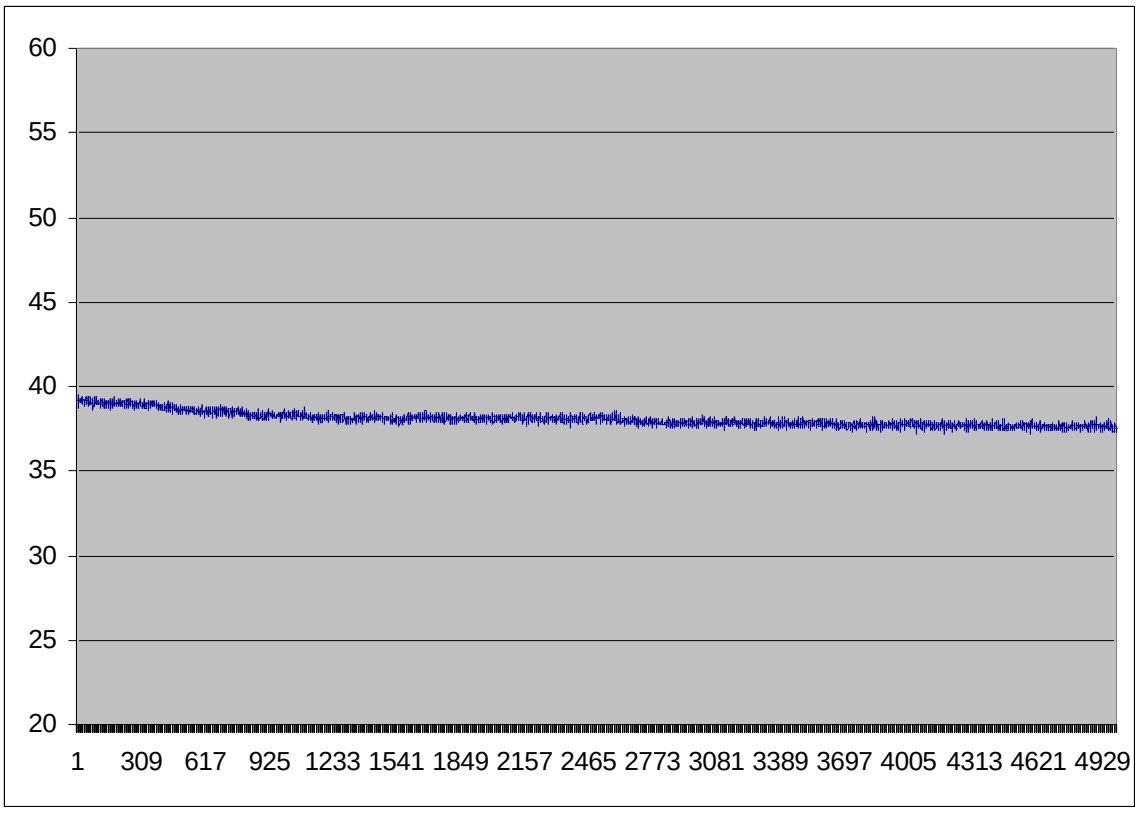
Figur 4: Goore-spill resultater uten støy



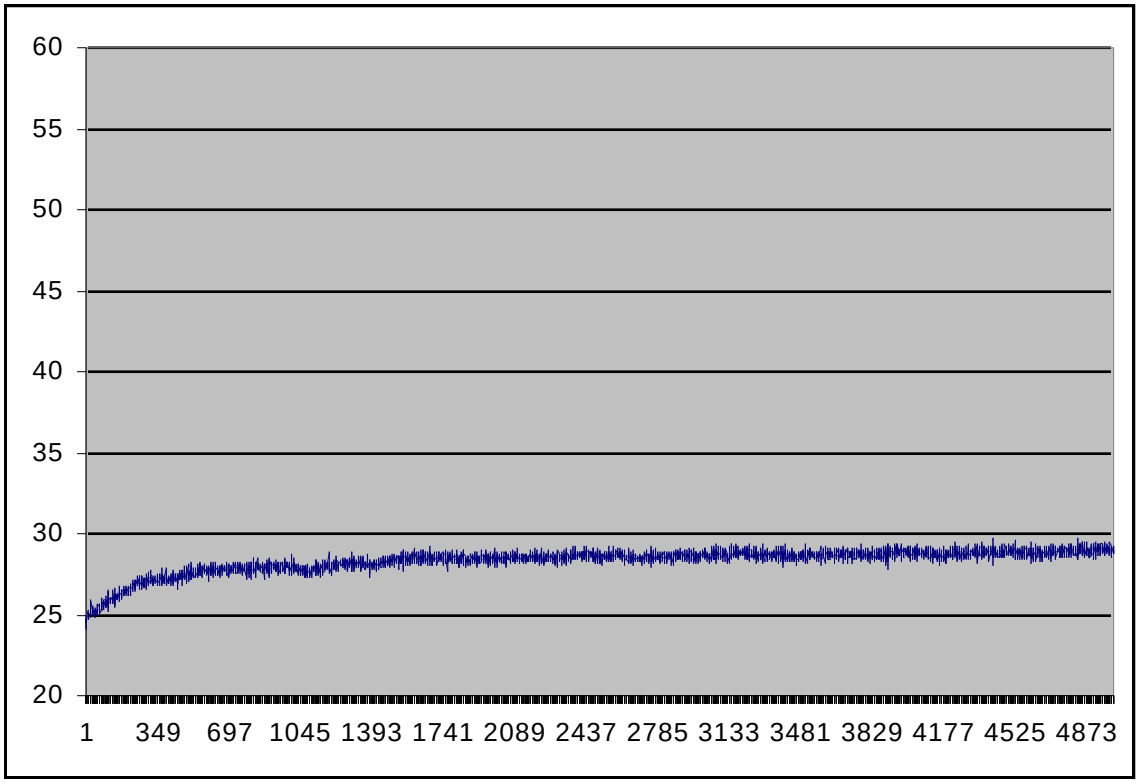
Figur 5: Goore-spill resultater med 5 % støy



Figur 6: Goore-spill resultater med 10 % støy



Figur 7: Goore-spill resultater med 20 % støy



Figur 8: Goore-spill resultater med 50 % støy

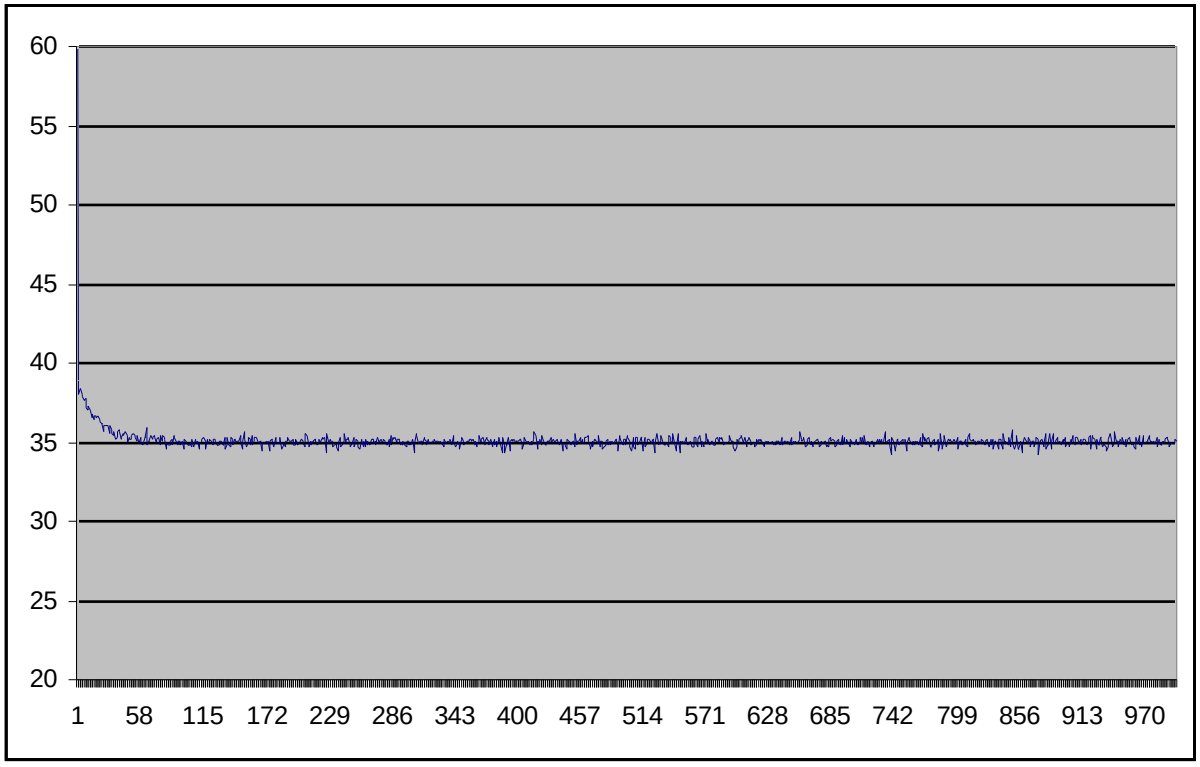
4.2.2 Kommentarer til Goore-spill baserte metoden

Goore-spill metoden er nå testet uten støy, med 5 %, 10 %, 20 %, 50 % støy.

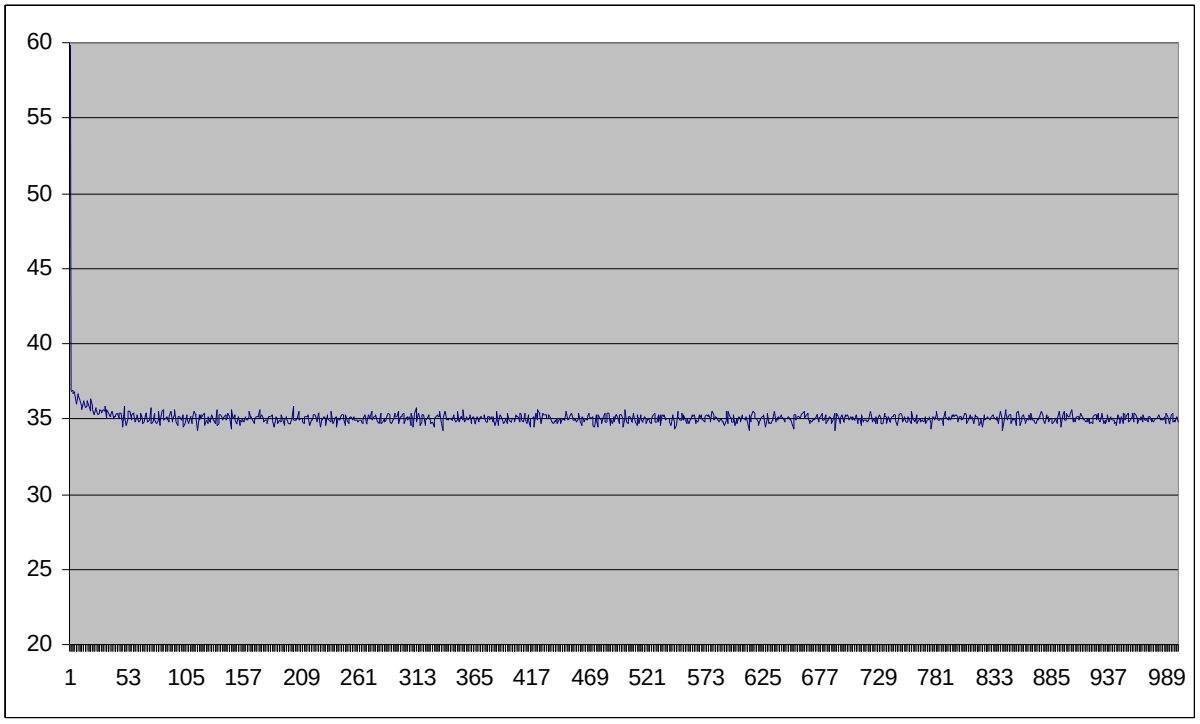
Først ser man at metoden ikke når målet som er 35 aktive sensorer, på 5000 iterasjoner. Men jeg ser tendensen til læring er der, og at metoden er på riktig vei mot målet. Innvirkningen av støy ser ut til å ha liten betydning når man ser på 5,10,20 % støy. Her ser det ut som at støy nærmest har en positiv effekt og driver grafen litt nærmere målet. Ser man derimot på 50 % støy, så ser man at metoden får store problemer med å lære, og vil kanskje aldri komme til målet.

Generelt på grafene så ser Goore-spill stabil ut, og man får ikke en veldig variabel linje.

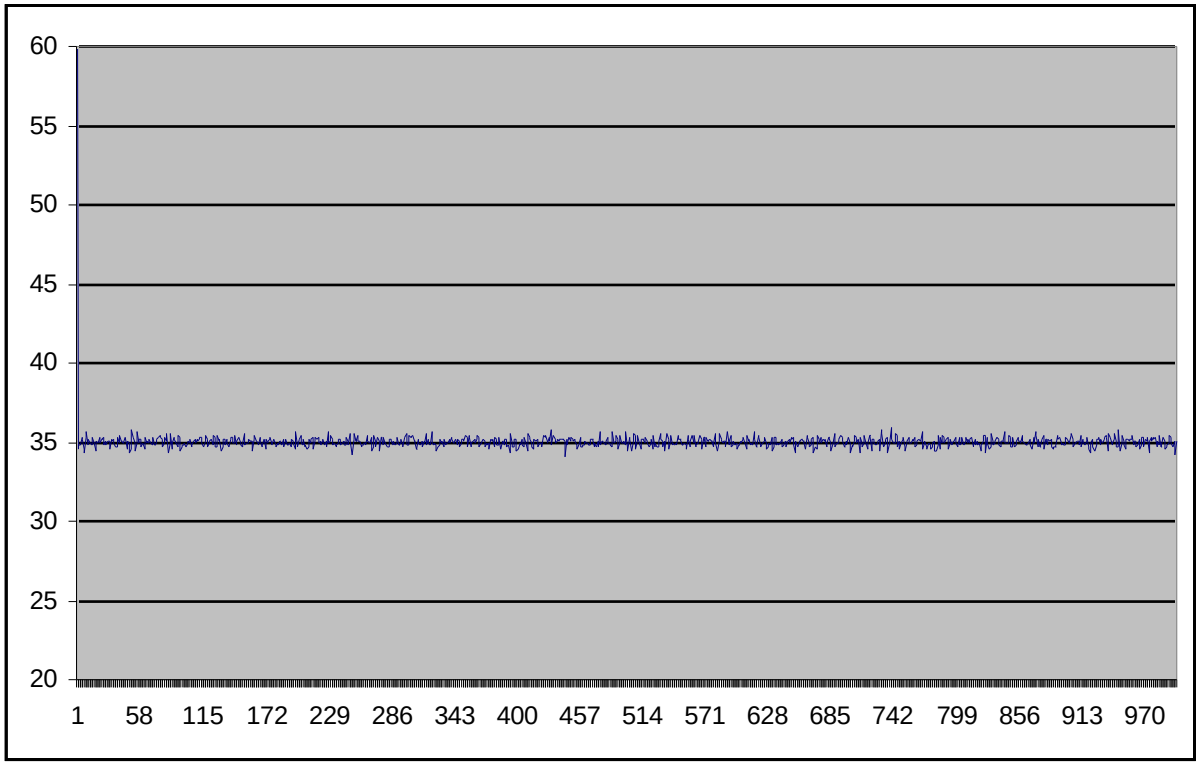
4.2.3 *Stochastic Weak Estimators* metoden



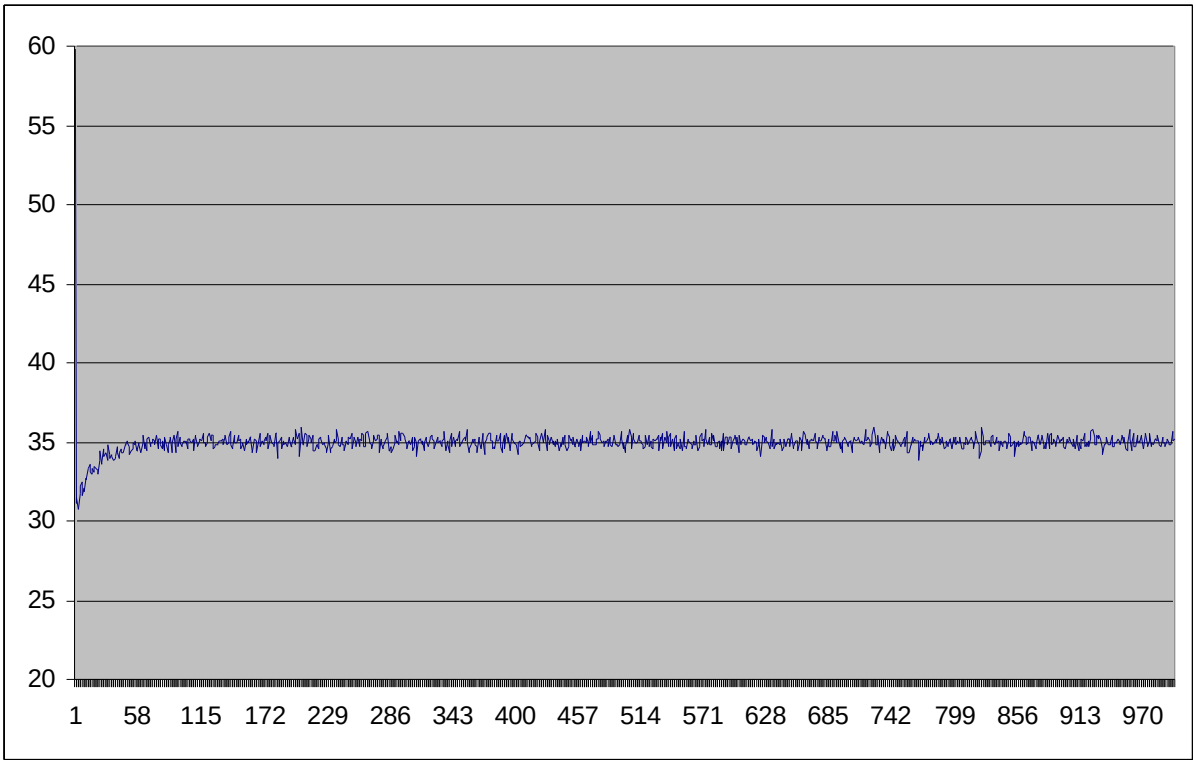
Figur 9: Stochastic Weak Estimators resultater uten støy



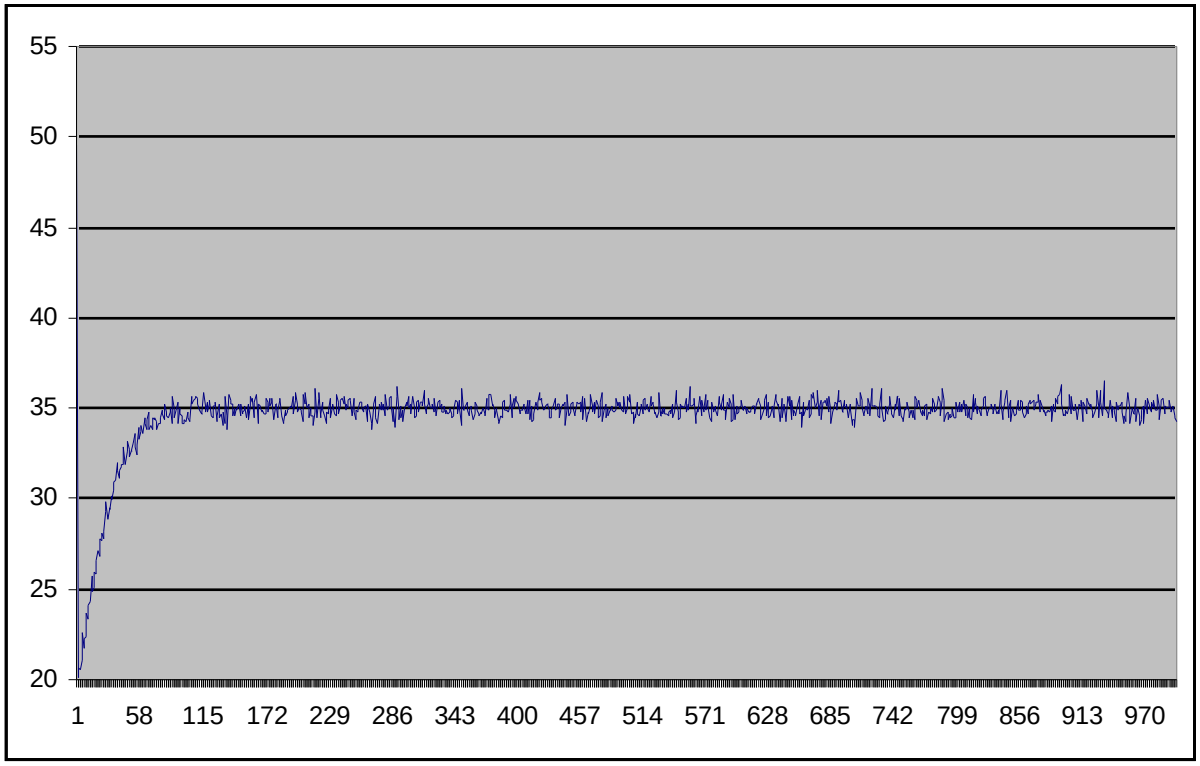
Figur 10: Stochastic Weak Estimators resultater med 5 % støy



Figur 11: Stochastic Weak Estimators resultater med 10 % støy



Figur 12: Stochastic Weak Estimators resultater med 20 % støy



Figur 13: Stochastic Weak Estimators resultater med 50 % støy

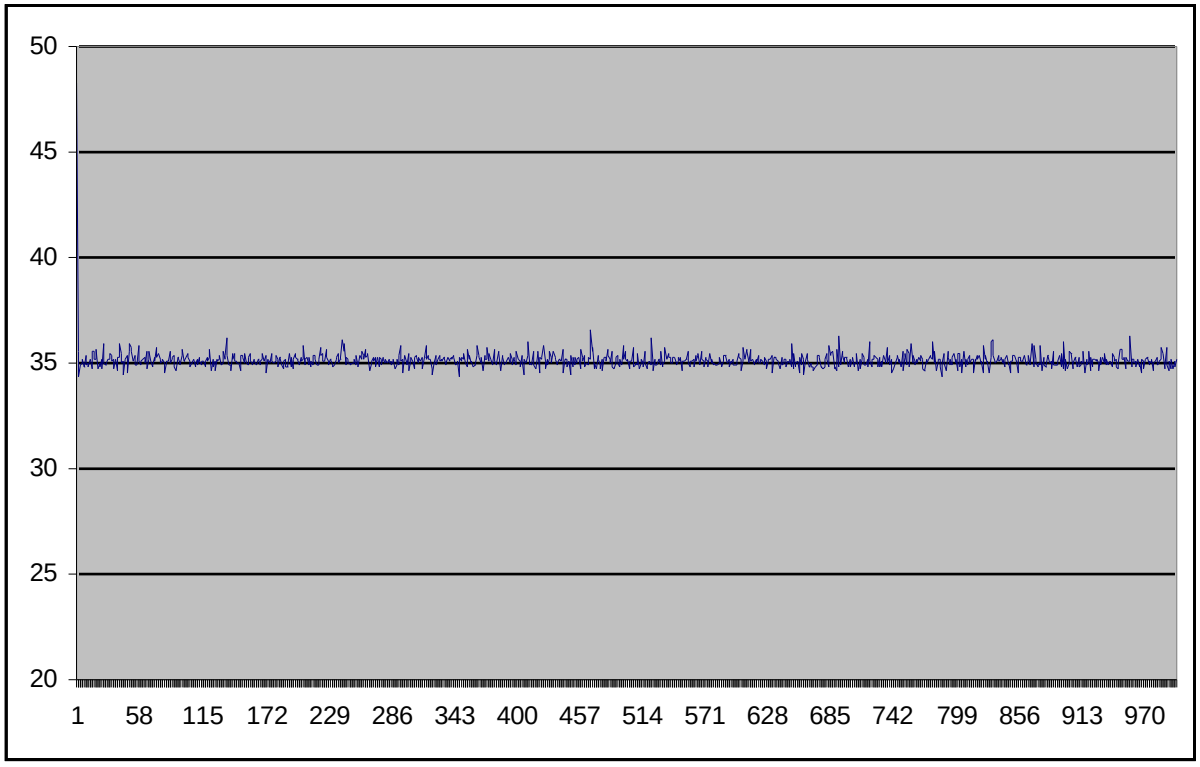
4.2.4 Kommentarer til Stochastic Weak Estimators

Stochastic Weak Estimators metoden er nå testet uten støy, med 5 %, 10 %, 20 %, 50 % støy.

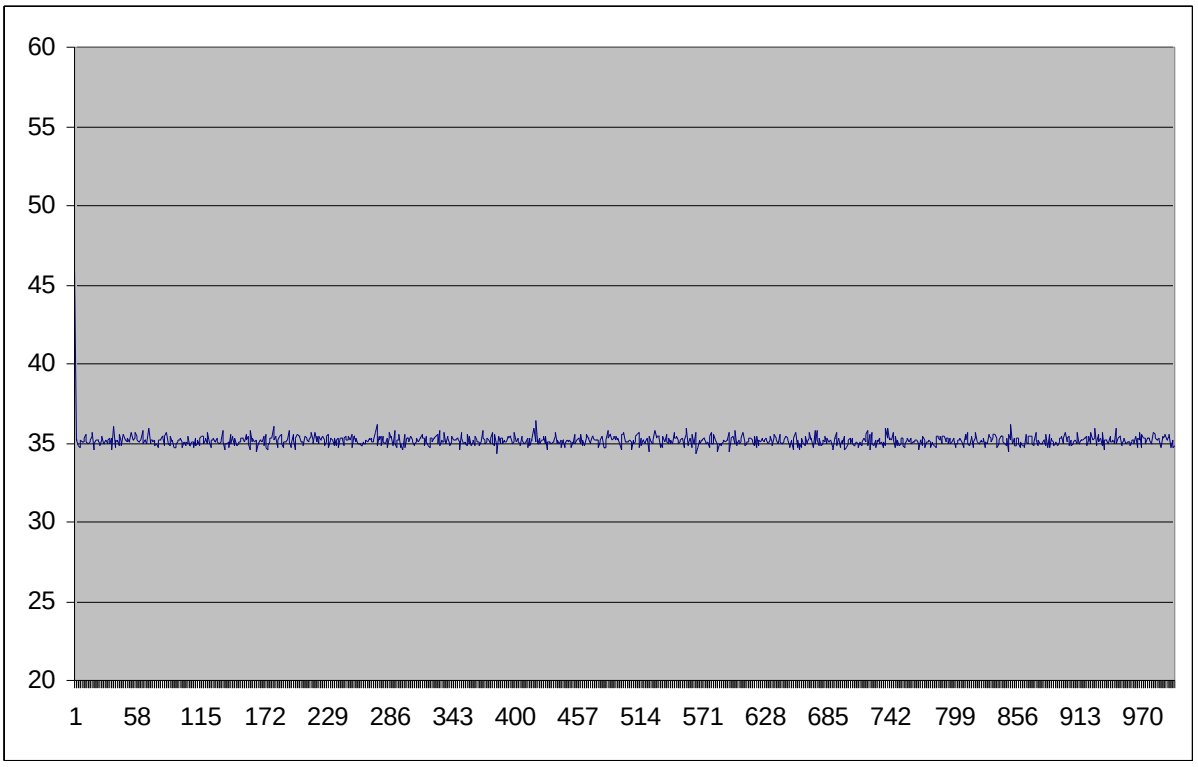
Først ser jeg at denne metoden raskt når målet på 35 aktive sensorer og den klarer altså dette på rundt 100 iterasjoner. Imidlertid ser man at grafen slår seg aldri hel til ro på 35 sensorer, men varierer noe rundt dette tallet. Men vi ser klart at resultatene klamrer seg rundt tallet 35 i grafene.

Støy har ikke mye innvirkning på hvor raskt metoden kommer frem til riktig antall svar. Men man ser at grafene varierer noe mer med mer støy. Metoden håndterer altså støy helt opp i 50% meget bra.

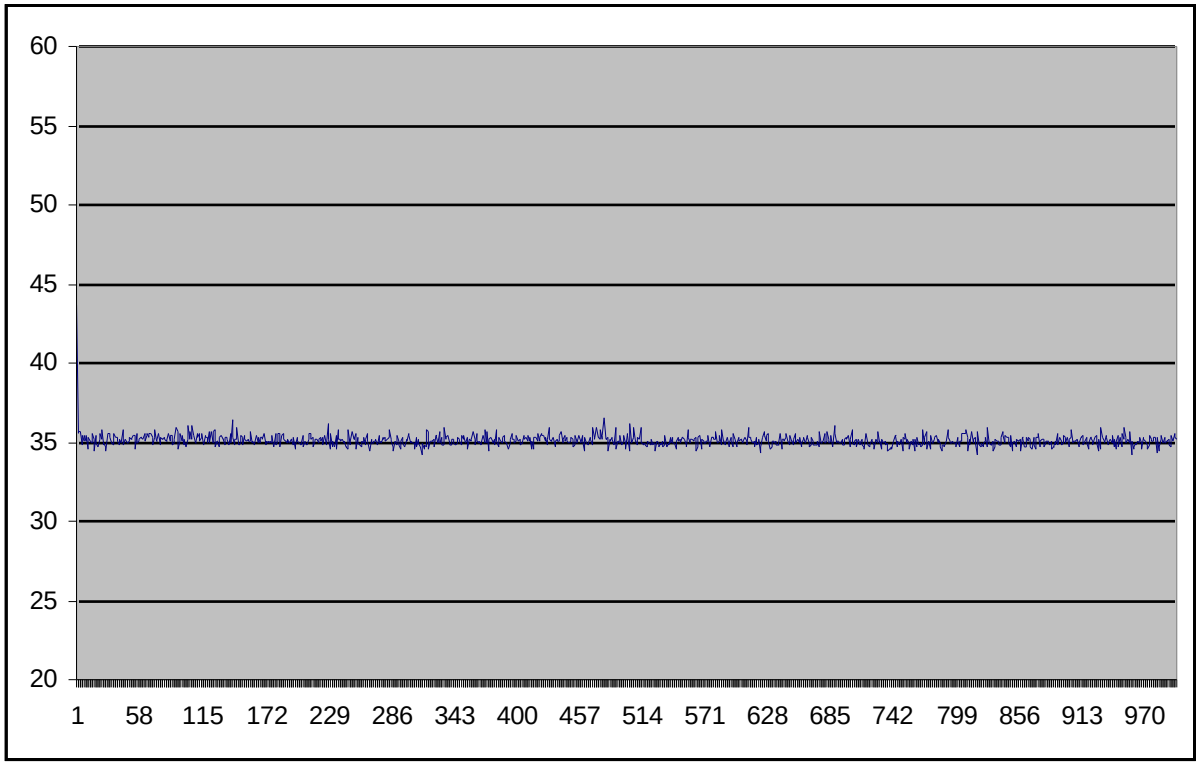
4.2.5 Bayesiansk estimering metoden



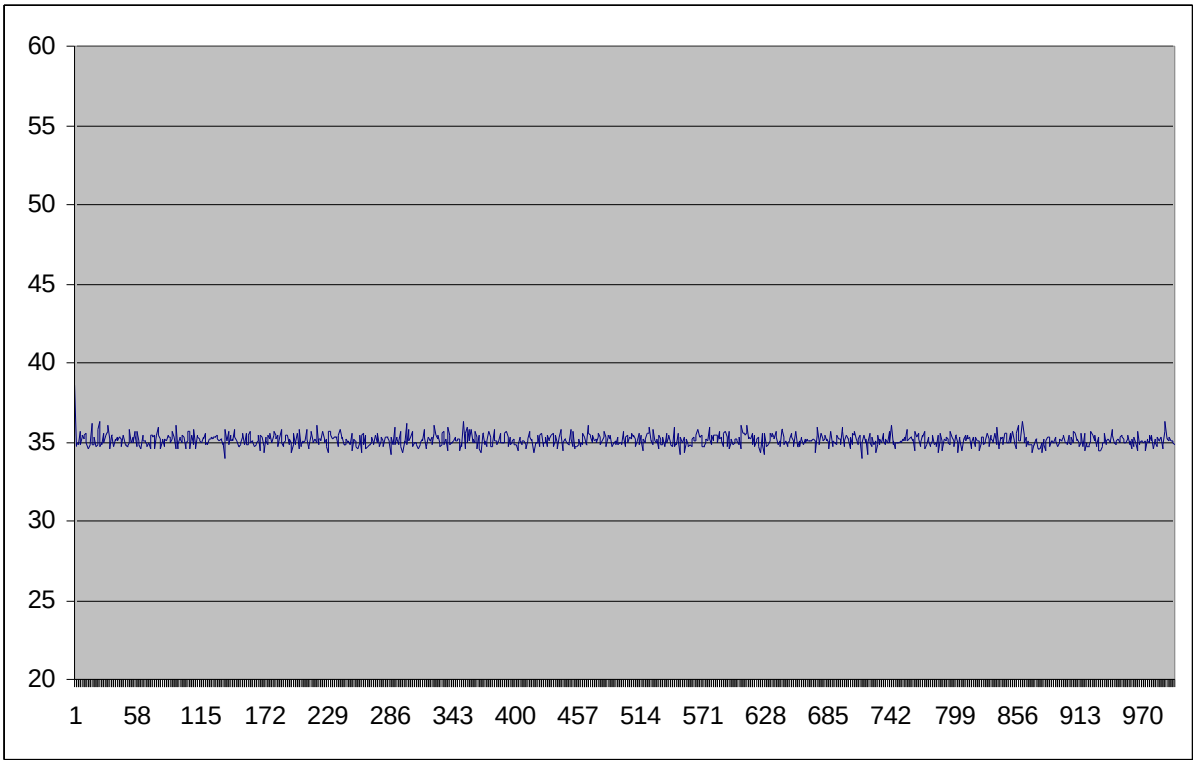
Figur 14: Bayesiansk estimering resultater uten støy



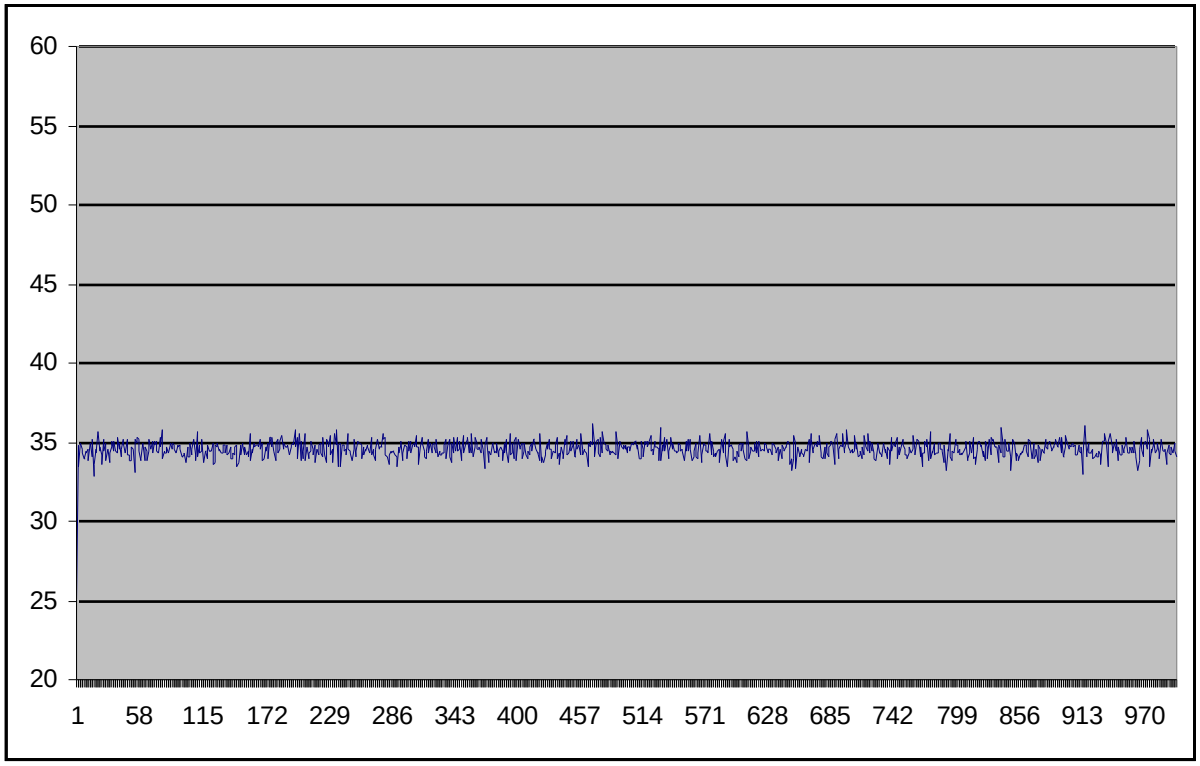
Figur 15: Bayesiansk estimering resultater med 5 % støy



Figur 16: Bayesiansk estimering resultater med 10 % støy



Figur 17: Bayesiansk estimering resultater med 20 % støy



Figur 18: Bayesiansk estimering resultater med 50 % støy

4.2.6 Kommentarer til *Bayesiansk estimering metoden*

Bayesiansk estimering metoden er nå testet uten støy, med 5 %, 10 %, 20 %, 50 % støy.

Først ser jeg at denne metoden er veldig rask til å stabilisere seg på riktig antall aktive sensorer. Den bruker rundt 2 iterasjoner, selv med støy. Men støyen har en innvirkning på hvor denne grafen stabiliserer seg. Ser man på støy ved 5,10,20 %, så ser man at grafen har en liten tendens til å ligge litt over 35. Mens ser man på 50 %, så har grafen litt tendens til å ligge litt under 35. Grafen ligger allikevel veldig nær 35. Grafen slår seg aldri hel til ro på 35 sensorer, men varierer noe rundt dette tallet. Men vi ser klart at resultatene holder seg rundt tallet 35 i grafene.

5 Diskusjon

I dette kapittelet blir masteroppgaven diskutert. Det innebærer en drøfting av trådløse sensornettverk, støy og strømforbruk. De forskjellige metodene som er testet blir vurdert hver for seg og siden sammenliknet med hverandre. Her kan man finne svar på hypotesene i kapittel 1.2.

5.1 Trådløse sensornettverk og støy

Jeg har i denne oppgaven basert meg på trådløse sensornettverk der man kan ha tilfeller av støy, for eksempel på grunn av dårlige sendeforhold. Støy i denne oppgaven betyr tap av meldinger fra sensor til basestasjon. Dette er lagt til metodene for at man skal ha en nær tilknytning til virkeligheten. Selv om man i denne oppgaven ikke har testet disse algoritmene i et ekte sensornettverk, skal resultatene uansett ta høyde for støy og kommunikasjonssvikt, som man meste sannsynlig vil bli utsatt for i et virkelig system. Hvor mye støy man skal beregne og bli utsatt for er ikke lett å si, men jeg har testet metodene i forskjellige scenarioer med 0,5,10,20 og 50 % støy. Merk at meldinger ifra basen til sensorer er i denne oppgaven ikke utsatt for støy, og man antar at sensorene alltid får broadcast meldingen fra basen.

5.2 Goore-spill metoden

Goore-spill metoden har vist seg å være den tregeste metoden av de 3 jeg har sett på. Det jeg mener med treg er at metoden bruker mange iterasjoner, i dette tilfellet over 5000, for å nå ønsket antall svar. Jeg vil kommentere minnet som i mine simuleringer er satt til N=3. Man har altså 3 positive tilstander og 3 negative, i motsetning til det eksempelet som jeg gikk igjennom tidligere i kapittel 3.1.2, der man hadde N=2. Dette betyr at sensorene kan bli enda sikrere i sitt valg og man får en mindre variabel graf, men som kanskje ikke beveger seg med tilfredsstillende få iterasjoner mot ønsket antall svar. Denne minnestørrelsen ble valgt på grunnlag av [1] der de har funnet at N=3 er den optimale størrelsen.

Denne metoden har mer komplekse sensorer enn de andre metodene som jeg har sett på, og det vil mest sannsylig resultere i at disse sensortypene bruker mer strøm. Dette er ikke noe jeg har testet men understreker at jeg antar at dette er tilfellet. Jeg vil derfor påpeke at dette ikke er en god metode for å spare strøm i sensornettverk, ettersom at den bruker mange flere iterasjoner på å nå ønsket antall sensorer, og er i seg selv mer krevende. Det må derimot også tas i betraktning at dette er den metoden som gir minst variasjon, og at dette kanskje på lang sikt kan resultere i at man får en gevinst i form av batterisparing. Faren er at det skal gå så lang tid før man når ønsket antall svar, at batteriene allerede er utladet.

Jeg har en kommentar til min javakode for denne metoden. I eksempelet i kapittel 3.1.2 vises det til at tilstandene er -1 og 1 i startfasen av metoden. I min kode er det altså -1 som er første negative tilstand og 0 er første positive. Derfor har jeg tilstandene negative [-3,-2,-1] og positive [0,1,2]

5.3 Stochastic Weak Estimators

Stochastic Weak Estimators metoden viser seg å være rask til å stabilisere seg rundt ønsket oppløsning. Den bruker under 100 iterasjoner. Jeg vil kommentere faktoren α som er brukt i metoden, og satt til 0,95. Desto nærmere α faktoren er 1, desto senere skjer tilnærmingen. Men metoden er da til gjengjeld mer robust mot støy eller feil.

Sensorene er også veldig enkle og jeg antar at de ikke vil bruke mye strøm på prosessering, i forhold til Goore-spill metoden. Dette er ikke noe jeg har testet og er kanskje en god ide for videre forskning.

5.4 Bayesiansk estimering

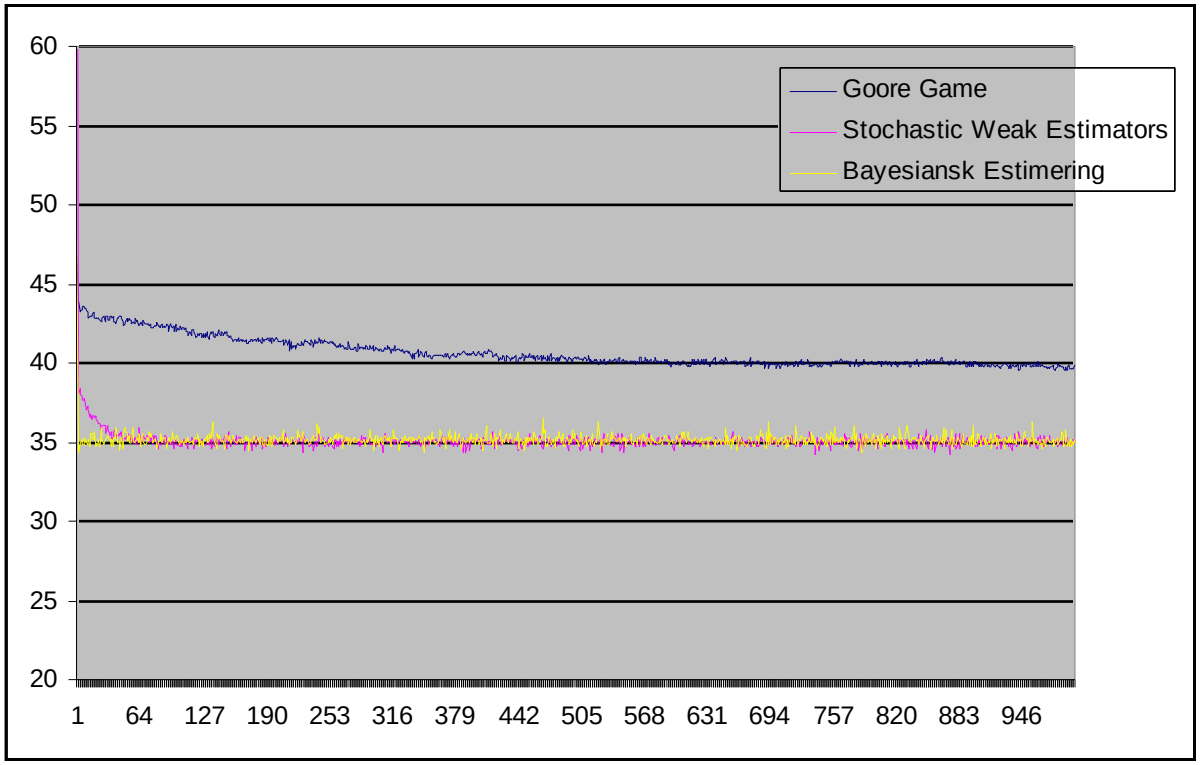
Metoden basert på Bayesiansk estimering trengte veldig få iterasjoner for å stabilisere seg rundt ønsket oppløsning. Den var den raskeste av de 3 metodene som jeg har sett på. Den har like enkle sensorer som Stochastic Weak Estimators.

Jeg vil nevne noen matematiske problemer jeg oppdaget underveis. Får man p som er større eller lik 1 settes p til 0,9999 fordi en p lik 1 betyr at man vet det riktige tallet sensorer og trenger ikke lete mer. P lik 1 vil resultere i at den binomiske fordelingen vil bli bare 0'er og en 1'er ved det riktige antallet sensorer. Men vi må huske at det kan alltid skje endringer slik at mulighetene må ikke låses slik. Det blir låst av den enkle grunn at man senere da bare vil gange med tallet 0, som gir 0. Derfor må vi legge til et svært lite tall 0,000001 til alle elementene i Pn når den normaliseres slik at man ikke risikerer og bare ha 0'er som vil gi 0 uansett hva den multipliseres med utenom den riktige n, som kanskje ikke er riktig lenger.

5.5 Sammenlikning av metodene

Jeg har testet 3 metoder i denne masteroppgave, og disse tre metodene har alle hatt forskjellig måte å prøve å løse samme problem. Som jeg har vist i resultatkapittelet så har metodene hatt forskjellige egenskaper. Noen bruker få iterasjoner for å nå ønsket oppløsning og noen er mer stabile men bruker mange iterasjoner for å nå ønsket oppløsning.

I figur 19 under ser vi en sammenlikning av metodene uten støy.

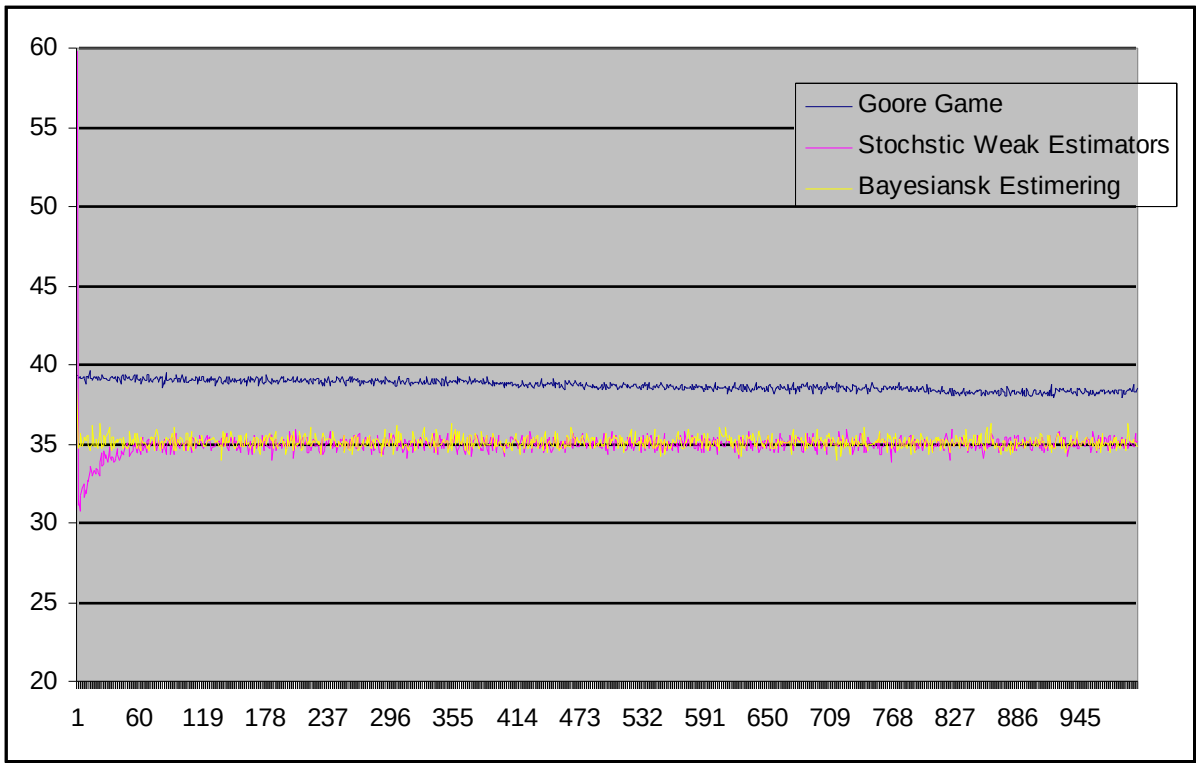


Figur 19 sammenlikning av metodene uten støy.

Denne figuren gir oss et godt bilde på forskjellene på disse metodene. Man ser Goore-spill som er treg til å komme til ønsket antall svar, men som er litt mindre hakkete enn de to andre metodene. Vi ser at Stochastic Weak estimators er raskere enn Goore-spill til å stabilisere seg rundt 35 svar, men tregere enn Bayesiansk estimering metoden som nesten begynner på riktig antall svar. Disse to estimeringsmetodene er nokså like stabile her uten støy, men Stochastic Weak Estimators metoden er nok litt mer stabil enn Bayesianske metoden.

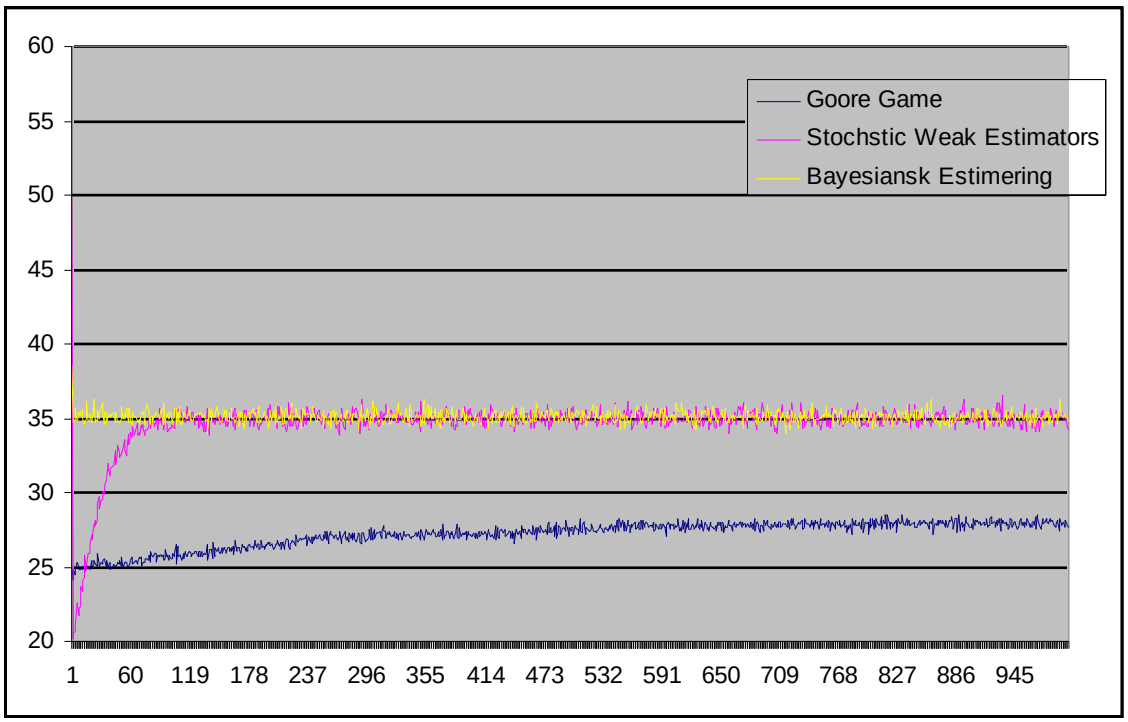
Samtidig er Stochastic Weak Estimators metoden ikke så stabil som Goore-spill metoden, og kan dermed ikke levere like stabile resultater. Det kan være systemer som stiller strenge krav til å få minimum antall svar til enhver tid, for å opprettholde tjenesten. Derimot et system der man kan forvente hyppig tap av sensorer og der det er nødvendig å utplassere grupper av nye sensorer med jevne mellomrom, vil det være nødvendig med rask tilpassing til endringene. Så vil det passe bedre med Stochastic Weak Estimators eller Bayesiansk estimering metodene. Det skal også her nevnes at jeg har ikke testet scenarioer der jeg utplasserer mange nye sensorer. Men dette er nevnt som mulig videre arbeid i kapittel 5.7.

I figur 20 under har vi sammenliknet metodene med 20 % støy.



Figur 20. Sammenlikning av metodene med 20 % støy

Når jeg nå sammenlikner metodene med 20 % støy, ser jeg fortsatt de samme tegnene som uten støy. Dette bekrefter at med 20 % støynivå klarer metodene seg meget tilfredsstillende. Generelt ser jeg at grafene varierer noe mer enn uten støy, men ligger rundt ønsket oppløsning. Unntaket er Goore-spill metoden som ikke ligger så nært ønsket oppløsning som man kanskje hadde håpet, men den ligger ikke nærmere uten støy, så derfor kan ikke støyen ta skylden for det.



Figur 21. Sammenlikning av metodene med 50 % støy

Sammenlikner jeg metodene med 50 % støy, ser jeg at grafene varierer mer, og Goore-spill metoden henger ikke med de andre to metodene. Goore-spill metoden ser ut til å bare vise svake tegn til læring og ser ikke ut til å være i stand til å nå ønsket oppløsning. Det begrunner jeg med at grafen ser nesten ut til å snu mot slutten og bevege seg bort ifra ønsket oppløsning. Stochastic Weak Estimators og Bayesiansk estimering metodene ser ut til å klare seg meget tilfredsstillende selv med 50 % støy.

5.6 Strømforbruk

Litt generelt om strømforbruket i trådløse sensornettverk. Strømforbruket i et sensornettverk blir påvirket av hvor mange sensorer som er aktive, i tillegg til andre problemer som hvor lenge sensorene er aktive, og hvor mye strøm sensorene bruker når de er aktive. Sensorene bruker batteri, og det er sensorenes kompleksitet og prosessorkraft som avgjør om hvor mye strøm hver enkelt sensor krever for å operere.

Minner om at den strømstyringen som er vurdert i denne oppgaven er kontrollen på hvor mange sensorer som er aktive og sender meldinger, og hvor mange som sover. Dette er den aller viktigste delen av strømsparingen.

Det kan tenkes at i eksempelet i innledningen, der jeg snakket om trådløse sensornettverk på månen for å måle temperatur, så kan sensorene gjerne lade opp batteriene sine ved hjelp av solcellepaneler eller lignende. Dette er ikke vurdert i denne oppgaven, men mulighetene er mange.

5.7 Videre arbeid

I mitt arbeid med denne masteroppgaven har det dukket opp temaer som kunne vært interessant å forske videre på. Det er for eksempel ikke testet hvordan metodene reagerer ved at man tilfører en ny stor mengde sensorer til scenarioet. Det er allikevel nærliggende å tro at metodene vil oppføre seg slik som i startfasen, siden metodens startparameterer er satt slik at antall sensorer er større enn ønsket antall svar. Men dette bør man teste grundigere.

Jeg antar at radioforbindelsen bruker mye strøm. Jeg har ikke noen begrunnelse for denne påstanden, men jeg har erfart at mobiltelefoner for eksempel bruker mer strøm når radioforbindelsen er aktiv. Her er det sikkert rom for forbedringer av radioforbindelsen mellom sensor og base. Et eksempel kan være at sensorene sender data når de skal og lytter når de skal. Samme gjelder basen (synkroniserte sendinger). Kunne vært interessant å se om det hadde hjulpet på batteriforbruket til sensorene.

Goore-spill metoden viser seg å være stabil og litt treg til å nå ønsket antall svar. Men kunne man forsket på initialiseringen av metoden, slik at den ble raskere, så hadde man fått en rask og stabil metode.

6 Konklusjon

Batteridrevne trådløse sensornettverk kan absolutt dra nytte av intelligente metoder for å spare strøm på batteriene. Dette er både fleksibelt og effektivt, med tanke på å håndtere forskjellige støynivåer og forskjellige antall tilgjengelige sensorer. Jeg har sett på tre forskjellige metoder, der man har to ulike typer. Den første metoden Goore-spill er en desentralisert metode der sensorene er Tsetlinautomater som er den lærende part. Mens de andre to metodene, Stochastic Weak Estimators og Bayesiansk estimering, er sentraliserte metoder der læringen skjer i basestasjonen. Resultatene er det som skiller metodene fra hverandre. Det er variasjoner om hvor fort de stabiliserer seg rundt ønsket oppløsning, og hvor stabile de er. Også sensorene som blir brukt i metodene er forskjellige. Goore-spill bruker Tsetlinautomater i sine sensorer og det kan gi utslag på strømforbruket, mens de andre to metodene bruker mindre avanserte sensorer.

Det som vil avgjør hvilken metode som er best, er opp til hvert enkelt scenario. Et scenario vil kreve forskjellige egenskaper av sensornettverkene sine, som batterilevetid, de fysiske påkjenninger og mengde radiostøy. Ved store fysiske påkjenninger vil man kunne regne med at sensorer slutter å operere, og at man må sette inn flere nye sensorer. I et slikt utfordrende scenario kan de to siste metodene foretrekkes fordi de gir raskest tilpassning til endringer i nettverket, som betyr at de tåler støy og tap av sensorer bedre enn Goore-spill metoden. Men skulle det være ønske om et mer stabilt svar fra sensorene, og batterilevetiden kanskje ikke er viktig, kan Goore-spill metoden være den rette metoden for jobben.

Valget av metode er altså avhengig av flere faktorer. Men metoden Stochastic Weak Estimators har ingen store fordeler sammenliknet med Bayesiansk estimering, som bruker mindre iterasjoner og nesten like stabil som Stochastic Weak Estimators. Så da står valget mellom to veldig ulike metoder, Goore-spill og Bayesiansk estimering. Jeg ville valgt Bayesiansk Estimering ettersom at den er meget rask til å estimere riktig oppløsning, og håndterer støy og sensortap meget effektivt. En fleksibel og rask metode med også andre mulige strømbesparende faktorer, som mindre avanserte sensorer.

7 Referanser

[1] QoS Control For Sensor Networks, R. Iyer and L. Kleinrock,
Dept. of Comput. Sci., California Univ., Los Angeles, CA, USA
Communications, 2003. ICC '03. IEEE International Conference 11 – 15 mai 2003

[2] Statistikk og sannsynlighetsregning 2.utgave av Alf Harbitz. Utgitt i 2004
ISBN: 82-7674-535-0

[3] Wikipedia
http://en.wikipedia.org/wiki/Bayes'_theorem.
Besøkt 30.05.07

[4] IKT590 sin hjemmeside
http://fag.grm.hia.no/ikt590/hovedoppgave/lister/lstValgteO1.aspx
Besøkt 23.05.07

8 Vedlegg

8.1 Metode Goore-spill

```
package algoritmer;

import java.util.Random;

public class GooreGame {
	double p;
	double antall_simuleringer = 100.0;
	int onsket_antall_svar = 35;
	int antall_runder = 5000;
	int antall_sensorer = 100;
	int antall_svar = 0;
	double stoy = 0.5; //Støyfaktor

	int[] tilstander = new int[antall_sensorer];
	int[] svar = new int[antall_runder];
	int[] sumSvar = new int[antall_runder];
	int tilstand;

	//MAIN kjører startSimulering()
	public static void main(String[] args) {
		GooreGame b = new GooreGame();
		b.startSimulering();
	}
	public void startSimulering(){


		for (int k=0;k<antall_simuleringer;k++){
			//nuller ut tilstander og svar fra forrige runde
			nullUtArrays();
			//Lager en tilfeldig start
			start();

			for (int i=0;i<antall_runder;i++){

				finn_antall_svar();
				svar[i]=antall_svar;
				sumSvar[i]+=svar[i];
				kalkuler_p();
				belonning_eller_straff();
			}
		}
		for (int j=0;j<antall_runder;j++){
			double a= 1.0*sumSvar[j]/antall_simuleringer;
			System.out.println(a);
```

```
		}
	}
	private void nullUtArrays() {
		for (int g=0;g<antall_runder;g++){
			svar[g]=0;
		}
		for (int i=0;i<antall_sensorer;i++){
			tilstander[i]=0;
		}
	}

	private void start() {

		for (int i=0;i<antall_sensorer;i++){
			if (tilfeldigTall()>=0.5)
				tilstand=0;
			else
				tilstand=-1;

			tilstander[i]=tilstand;
		}
	}
	private void finn_antall_svar() {
		antall_svar=0;

		for (int i=0;i<antall_sensorer;i++)
		{
			if (tilstander[i]>=0 && tilfeldigTall()>stoy)
			{
				antall_svar++;
			}
		}
	}

	public void kalkuler_p(){
	p= 0.2 + (0.8 * Math.exp(-0.002*Math.pow((antall_svar - onsket_antall_svar), 2)));
	}

	private void belonning_eller_straff() {

		for (int f=0;f<antall_sensorer;f++){

			if (tilfeldigTall() <= p)
				reward(f);
			else
				penalize(f);
		}
	}

	public void reward(int index){
```

```
                int j = tilstander[index];

                if (tilstander[index] >=0 && tilstander[index]<2){
                        tilstander[index]=++j;
                        }
                else if(tilstander[index] <0 && tilstander[index]>-3){
                        tilstander[index]=--j;
                }
        }
        public void penalize(int index){

                int j = tilstander[index];

                if (tilstander[index] >=0 && tilstander[index]<=2){
                        tilstander[index]=--j;
                        }
                else if(j <0 && j>=-3){
                        tilstander[index]=++j;
                }
        }
        public double tilfeldigTall(){ //tilfeldig tall k, mellom 0 og 1
                Random generator = new Random();
                double k = generator.nextDouble();
                return k;
        }
}
```

8.2 Stochastic Weak Estimators

```
package algoritmer;

import java.util.Random;

public class StochasticWeakEstimators {

	double antall_simuleringer=100.0;
	int antall_runder=1000;
	int antall_sensorer=100;
	double alpha = 0.95;  //bestemmer hvor mye n fra forrige runde skal påvirke ny n.

	double n=90;			//antatt antall sensorer forrige runde
	int antall_svar;
	int totalt_antall_svar;
	double oensket_antall_svar=35;
	double k;			//tilfeldig tall mellom 0 og 1
	double j;
	double p=1;			//gitt sannsynlighet i prosent fra base til sensorene

	double stoy = 0.5;		//Støy i miljøet

	double[] svar= new double[antall_runder];
	double[] sumSvar= new double[antall_runder];


	public int aktivSensor(){
		Random generator = new Random();
		k = generator.nextDouble();
		j = generator.nextDouble();

		if(k<p && j>stoy)
			return 1;
		else
			return 0;
	}

	public void kalkuler_ant_svar(){

		for (int i=0;i<antall_sensorer;i++){

			if (aktivSensor() == 1)
			antall_svar++;
		}
	}
	public void kalkuler_n(){
		n= (alpha*n)+((antall_svar/p)*(1-alpha));
	}
```

```
	public void kalkuler_p(){
		p=oensket_antall_svar/n ;
	}

	public void tot_kalk(){
		for (int s=0;s<antall_simuleringer;s++){

			p=1.0;
			n=90;

			for (int i=0;i<antall_runder;i++){
				antall_svar =0;
				kalkuler_ant_svar();
				svar[i] =antall_svar;
				sumSvar[i]+=svar[i];
				kalkuler_n();
				kalkuler_p();
			}
		}
		for (int i=0;i<antall_runder;i++){
			double a=sumSvar[i]/antall_simuleringer;
			System.out.println(a);
		}
	}
	public static void main(String[] args) {
		StochasticWeakEstimators b = new StochasticWeakEstimators();
		b.tot_kalk();
	}
}
```

8.3 Bayesiansk estimering

```
package algoritmer;

import java.util.Random;
import cern.*;
import cern.jet.random.Binomial;

public class BayesianskEstimering {

	int antall_simuleringer=100;
	int maxAntall=200; //max antall elementer i Pn
	int n = 100; //Antall sensorer
	double q = 0.0;
	double antall_onsket_svar = 35.0;  // Q
	int antall_svar; //svar man mottar ifra sensorene som er paa
	double sendt_ut_p; // sendt_ut_p=Q/n
	int antall_runder = 1000;
	double k;	//tilfeldig tall mellom 0 og 1
	double j;	//tilfeldig tall mellom 0 og 1
	double stoy = 0.5;
	double[] Pn = new double[maxAntall];//Liste med tall,
                                            //som til sammen er lik 1
	double[] bf = new double[maxAntall];//Liste med den binomiske fordelingen
	int tilfeldigN;
	double[] svar = new double[antall_runder];
	double[] sum=new double[antall_runder];

	public void init(){
		for (int i=0;i<maxAntall;i++){
			Pn[i]=1.0/maxAntall;
		}
	}

	public int trekkUtTilfeldigN(){ //mellom 0 og max
		double r = tilfeldigTall(); //tilfeldig mellom 0 og 1
		double sum = Pn[0];
		int i=0;

		while (r>sum){
			i++;
			sum+=Pn[i];
		}
		return i;
	}

	public void sendtUtP(){
		//p=Q/n
		tilfeldigN = trekkUtTilfeldigN();
```

```
        sendt_ut_p = antall_onsket_svar / tilfeldigN;
        if (sendt_ut_p>=1){
                sendt_ut_p=0.999999999;
        }
        System.out.println(sendt_ut_p);
}
private double multipliserPnbf() {
        double u=0.0;
        for (int i=0;i<maxAntall;i++){

                u = Pn[i]*bf[i]; //Ganger Pn med den binomiske fordelingen bf

                Pn[i]=u;
        }
        return u;
}
public void normaliser(){
        double sum=0;

        for (int i=0;i<maxAntall;i++){
                Pn[i]+=0.000001;
                sum += Pn[i]; //summerer alle verdiene i Pn

        }
        for (int i=0;i<maxAntall;i++){
                Pn[i] = Pn[i]/sum;
        }
}
public void binomiskFordeling(){

        for (int i=0; i<maxAntall;i++){
                if (i < antall_svar) {
                        bf[i] = 0.0;
                } else {
                        Binomial b = new Binomial(i, sendt_ut_p,
                        cern.jet.random.engine.RandomEngine.makeDefault());
                        bf[i]= b.pdf(antall_svar);
                }
        }
}
public double tilfeldigTall(){ //tilfeldig tall k, mellom 0 og 1
        Random generator = new Random();
        double k = generator.nextDouble();
        return k;
}
public int aktivSensor(){
        Random generator = new Random();
        k = generator.nextDouble();
        j = generator.nextDouble();
```

```
                if(k<sendt_ut_p && j>stoy)
                        return 1;
                else
                        return 0;
        }
        public void kalkuler_ant_svar(){
                antall_svar=0;
                for (int i=0;i<n;i++){

                        if (aktivSensor() == 1)
                        antall_svar++;
                }
        }
        public static void main(String[] args) {
                BayesianskEstimering b = new BayesianskEstimering();

                b.startSimulering();
        }
        public void startSimulering() {
                for (int j=0;j<antall_simuleringer;j++){
                        init();

                        for (int i=0;i<antall_runder;i++){
                                sendtUtP();
                                kalkuler_ant_svar();
                                svar[i]+=antall_svar;
                                binomiskFordeling();
                                multipliserPnbf();
                                normaliser();
                        }
                }
                for (int j=0;j<antall_runder;j++){
                        System.out.println(svar[j]/antall_simuleringer);
                }
        }
}
```