NTNU
Det skapende universitet

Rekursiv blokkoppdatering av Isingmodellen

Bjarne Sæther

Master i fysikk og matematikk
Oppgaven levert: August 2006
Hovedveileder: Håkon Tjelmeland, MATH

Norges teknisk-naturvitenskapelige universitet
Institutt for matematiske fag

Oppgavetekst

I denne masteroppgaven vil vi implementere en rekursiv algoritme for å oppdatere sannsynlighetene i en blokk i Isingmodellen.

Oppgaven gitt: 19. januar 2006
Hovedveileder: Håkon Tjelmeland, MATH

Rekursiv blokkoppdatering av Isingmodellen

Bjarne Sæther

9. august 2006

Forord Denne prosjektrapporten er en del av faget TMA4905 Masteroppgave, statistikk. Faget teller 30.0 studiepoeng. Jeg valgte denne masteroppgaven fordi jeg er interessert i Markov Chain Monte Carlo metoder spesielt innen romlige anvendelser, og arbeidet med dette også i faget TMA4700 Matematiske fag, fordypningsemne. Jeg vil benytte anledningen til å rette en stor takk til min veileder, førsteamanuensis Håkon Tjelmeland, for mye hjelp og for å ha vist stor tålmodighet underveis.

Bjarne Sæther, Trondheim, 09.08.2006

Sammendrag I denne rapporten sammenligner vi tre varianter av Markov Chain Monte Carlo (MCMC) - simulering av Isingmodellen. Vi sammenligner enkeltnode-oppdatering, naiv blokkoppdatering og rekursiv blokkoppdatering. Vi begynner med å gi en generell introduksjon til markovfelt og Isingmodellen. Deretter viser vi det teoretiske fundamentet som MCMC-metoder hviler på. Etter det gir vi en teoretisk introduksjon til enkeltnode-oppdatering. Så gir vi en innføring i naiv blokkoppdatering som er den tradisjonelle metoden å utføre blokkoppdatering på. Deretter gir vi en tilsvarende innføring i en nylig foreslått metode for å gjennomføre blokkoppdatering, nemlig rekursiv blokkoppdatering. Blokkoppdatering er en metode som har vist seg nyttig med hensyn på miksing når vi simulerer. Med det menes at blokkoppdatering har vist seg nyttig i forhold til å utforske utfallsrommet til fordelingen vi er interessert i med færre iterasjoner enn enkeltnode-oppdatering. Problemet med naiv blokkoppdatering er imidlertid at vi raskt får en høy beregningsmengde ved at hver iterasjon tar veldig lang tid. Vi prøver også ut rekursiv blokkoppdatering. Denne tar sikte på å redusere beregningsmengden for hver iterasjon når vi utfører blokkoppdatering på et markovfelt. Vi viser så simuleringsalgoritmer og resultater. Vi har simulert Isingmodellen med enkeltnode-oppdatering, naiv blokkoppdatering og rekursiv blokkoppdatering. Det vi sammenligner er antall iterasjoner før markovfeltet konvergerer og spesielt beregningstiden pr iterasjon. Vi viser at beregningsmengden pr iterasjon øker med 91000 ganger med naiv blokkoppdatering dersom vi går fra en 3×3 blokk til en 5×5 blokk. Tilsvarende tall for rekursiv blokkoppdatering er en økning på 83 ganger fra en 3×3 blokk til en 5×5 blokk.

Vi sammenligner også tiden det tar før Isingmodellen konvergerer. Når vi benytter naiv blokkoppdatering finner vi at Isingmodellen bruker 15 sekunder på å konvergere med en 3×3 blokk, 910 sekunder på å konvergere med en 4×4 blokk og 182000 sekunder med en 5×5 blokk. Tilsvarende tall for rekursiv blokkoppdatering er 3.74 sekunder for en 3×3 blokk, 72 sekunder for en 4×4 blokk og 141.2 sekunder for en 5×5 blokk. Når vi benytter enkeltnode-oppdatering bruker feltet 6.6 sekunder på å konvergere.

Innhold

1 Innledning

Innen romlig statistikk ønsker man typisk å modellere romlige fenomener med statistiske modeller. Vi ønsker altså å tilpasse en modell til en mengde data, for deretter, for eksempel, å estimere verdien i noder som ikke er kjente ved hjelp av modellen. En node betegner her en generell stokastisk variabel i en romlig modell. Typiske eksempler på slike problemer kan være hvilken type bergart vi har i en spesiell node i rommet under havoverflaten gitt bergarten i de nærliggende punktene, genetiske sammenhenger mellom fugler gitt deres geografiske posisjon eller intensiteten av kriminalitet i forskjellige deler av en bykjerne.

Markovfelt (MRF) brukes til å modellere et stort antall fenomener innen naturvitenskapen, spesielt der vi ønsker å modellere romlige sammenhenger mellom kvantitative variabler. MRF har den egenskapen at en node i et MRF er betinget uavhengig alle andre noder i et MRF dersom de naboene som er med i nabolaget til den noden er kjente. For å kunne benytte MRF til å modellere sammenhenger som vi nevnte ovenfor må vi benytte bayesiansk inferens og Markov chain Monte Carlo (MCMC) simulering.

Den mest brukte metoden for å utføre MCMC-simulering i MRF har vært enkeltnode-oppdatering. Enkeltnode-oppdatering betyr i praksis å oppdatere en og en node i et MRF om gangen. Vi simulerer hver node i feltet ved hjelp av den fulle betingede fordelingen til det punktet. Problemet med denne metoden er at den typisk gir treg miksing av nodene i feltet, som medfører at sannsynligheten for at en node endrer tilstand i en iterasjon er lav. Dette kommer av at nodene i et MRF i større eller mindre grad er avhengige av de nærmeste naboene i feltet, avhengig av hvilken nabolagsfunksjon vi benytter i modellen. Nabolagsfunksjonen i en slik modell sier hvilken avhengighet det er mellom nabonoder i feltet. En metode som kalles blokkoppdatering er foreslått for å forbedre miksingen i MCMC-simuleringen av MRF. Blokkoppdatering går ut på å beregne den betingede sannsynligheten til en mengde av noder i et MRF gitt resten av nodene i feltet, og oppdatere alle nodene i samme iterasjon i simuleringen. Gjennom empiriske forsøk er det vist at en slik måte å simulere på vil øke antall noder som endrer tilstand i hver iterasjon betraktelig. Dette vil typisk være avhengig av blokkstørrelsen vi velger, altså hvor mange noder vi ønsker å simulere i samme iterasjon, og dette har gitt oss et nytt problem, nemlig uhåndterbare beregningsmengder. Som vi skal se senere i rapporten vil kjøretiden når vi benytter naiv blokkopdatering øke eksponensielt med blokkstørrelsen, der størrelsen av utfallsrommet er grunntallet i eksponenten. I problemer av en viss kompleksitet blir dette fort vanskelig å håndtere. I denne rapporten vil vi derfor utforske en metode som vil kunne redusere denne beregningstiden betraktelig, og som dermed vil gjøre blokkoppdatering mer nyttig innen komplekse problemer. Dette kalles rekursiv blokkoppdatering.

I kapittel 2 vil vi forklare den grunnleggende teorien bak MCMC-simulering. Her vil vi også forklare de tre metodene vi skal sammenligne, enkeltnode-oppdatering,

naiv blokkoppdatering og rekursiv blokkoppdatering. Vi vil videre også forklare hvordan disse metodene kan implementeres for isingmodellen. I kapittel 3 vil vi sammenligne miksing og kjøretid for de tre metodene. I kapittel 4 vil vi diskutere forslag til forbedringer i metoden rekursiv blokkoppdatering.

2 Teori

I denne delen av rapporten vil vi gi en oversikt over den teorien som ligger til grunn for beregningene vi har gjort. I kapittel 2.1 vil vi gi en introduksjon til MCMC-simulering. Deretter vil vi gi en innføring til markovfelt og Isingmodellen. Siden vil vi forklare teorien bak henholdsvis enkeltnode-oppdatering, vanlig blokkoppdatering og rekursiv blokkoppdatering i kapittel 2.3, 2.4 og 2.5. I disse kapitlene vil vi også diskutere hvordan vi kan implementere disse metodene for å simulere fra en bestemt type MRF, nemlig Isingmodellen.

2.1 Isingmodellen

Anta et generelt stokastisk felt X med n noder, som blir nummerert ved $i \in S = \{1, ..., n\}$. La Ω være utfallsrommet til hele feltet. Mengden av noder $\partial(i)$ definert rundt node i definerer et nabolag rundt i hvis $i \notin \partial(i)$ og $i \in \partial(j) \Leftrightarrow j \in \partial(i)$. Dette betyr at en node ikke kan være nabo til seg selv, og at dersom node i er med i nabolaget til node j så må også j være med i nabolaget til i. Mengden av noder i $\partial(i)$ betegner naboene til i.

Et stokastisk felt $x = (x_1, x_2, ..., x_n)$ er et markovfelt med hensyn på et nabolag $\partial = \partial(i) : i \in S$ hvis

$$\pi(X_i = x_i | X_j = x_j; \forall j \in S; j \neq i) = \pi(X_i = x_i | X_j = x_j; j \in \partial(i)). \tag{1}$$

Dette betyr at vi har et markovfelt dersom den betingede fordelingen for en bestemt node i feltet gitt alle andre noder i feltet kun er avhengig av nabolaget til noden. I denne rapporten vil vi konsentrere oss om en spesiell type MRF, nemlig Isingmodellen. I modellen vi skal benytte, bruker vi første ordens nabolag. Dette vil si at hver node kun er avhengig av de fire nærmeste naboene (nord, sør, øst og vest). Simultanfordelingen for Ising-modellen er gitt ved

$$\pi(x) = c \cdot \exp\left(\beta \cdot \sum_{i \sim j} I[x_i = x_j]\right), \tag{2}$$

der $\pi(x)$ angir simultanfordelingen for hele feltet, c er en ukjent normaliseringskonstant, $j \sim i$ betyr at j er nabo til i, og summen i eksponenten sier da at vi skal summere over alle nabopar slik at node j er nabo til node i. Uttrykket i eksponenten kalles en potensialfunksjon og er gitt ved

$$V(x) = \beta \cdot \sum_{i \sim j} I[x_i = x_j]. \tag{3}$$

Vi definerer også $h(x)$ ved

$$h(x) = \exp\left(\beta \cdot \sum_{i\sim j} I[x_i = x_j]\right). \tag{4}$$

Utfallsrommet til Ising-modellen er gitt ved $x_i \in \Lambda = \{0, 1\}$. Den betingede fordelingen til en node gitt alle andre i Ising-modellen kan utledes direkte fra simultanfordelingen, gitt i (2). La x_{-i} betegne alle nodene i et MRF unntatt node i. Vi kan da skrive fra definisjonen av betinget sannsynlighet

$$\pi(x_i|x_{-i}) = \frac{\pi(x)}{\pi(x_{-i})} = \frac{\pi(x)}{\sum_{x_i\in\Lambda} \pi(x = (x_i, x_{-i}))}. \tag{5}$$

I dette uttrykket kan vi nå sette inn (2). Ved å trekke ut de leddene som inneholder x_i fra summene får vi så

$$\pi(x_i|x_{-i}) = \frac{c \cdot \exp\left(\beta \cdot \sum_{j\sim i} I[x_j = x_i]\right) \cdot \exp\left(\beta \cdot \sum_{\substack{u\neq i \\ v\neq i \\ v\sim u}} I[x_u = x_v]\right)}{\sum_{x_i\in\Lambda} c \cdot \exp\left(\beta \cdot \sum_{j\sim i} I[x_j = x_i]\right) \cdot \exp\left(\beta \cdot \sum_{\substack{u\neq i \\ v\neq i \\ v\sim u}} I[x_u = x_v]\right)}. \tag{6}$$

Ved å forkorte de delene som ikke omhandler i og j mot hverandre i (6) får vi til slutt

$$\pi(x_i|x_{-i}) = \frac{\exp\left(\beta \cdot \sum_{j\sim i} I[x_j = x_i]\right)}{\sum_{x_i\in\Lambda} \exp\left(\beta \cdot \sum_{j\sim i} I[x_j = x_i]\right)}. \tag{7}$$

Som vi ser er dermed den betingede fordelingen til en node i Ising-modellen, gitt alle andre nodene i feltet, kun avhengig av nabolaget $\partial(i)$ til noden. Ettersom vi bare har to verdier i utfallsrommet kan vi sette opp de betingede fordelingene til hele utfallsrommet direkte. Vi setter først inn $x_i = 1$ og får

$$\pi(x_i = 1|\partial(x_i)) = \frac{\exp\left(\beta \cdot \sum_{j\sim i} I[x_j = 1]\right)}{\exp\left(\beta \cdot \sum_{j\sim i} I[x_j = 1]\right) + \exp\left(\beta \cdot \sum_{j\sim i} I[x_j = 0]\right)} \tag{8}$$

$$= \frac{1}{1 + \exp\left(\beta\left(\sum_{j\sim i} I[x_j = 0] - I[x_j = 1]\right)\right)}. \tag{9}$$

Dersom vi setter inn tilsvarende for $x_i = 0$ får vi

$$\pi(x_i = 0|\partial(x_i)) = \frac{1}{1 + \exp\left(\beta\left(\sum_{j\sim i} I[x_j = 1] - I[x_j = 0]\right)\right)}. \tag{10}$$

For mer detaljert og komplett informasjon om markovfelt vil vi henvise til Besag (1974).

2.2 Markov Chain Monte Carlo simulering

Direkte metoder for å simulere fra MRF finnes ikke og vi må derfor benytte MCMC-simulering. Ideen bak MCMC-simulering er å lage en markovkjede bestående av stokastiske variabler x med utfallsrom Λ av størrelse k verdier og grensefordeling $\pi(x)$ og deretter produsere (avhengige) realisasjoner fra $\pi(x)$. I denne rapporten skal vi beskrive tre måter å gjøre dette på, men først vil vi forklare teorien litt mer generelt.

En av de mest brukte metodene for å generere realisasjoner fra en vilkårlig grensefordeling $\pi(x)$ er Metropolis-Hastings metoden. I denne metoden konstruerer man en markovkjede som har $\pi(x)$ som sin grensefordeling. Deretter simulerer man markovkjeden lenge, og bruker de simulerte realisasjonene til å estimere de parametrene man er interessert i. Ettersom vi ønsker en markovkjede som skal generere realisasjoner fra π må vi finne en overgangsmatrise P slik at markovkjeden er aperiodisk, irredusibel og oppfyller

$$\pi(y) = \sum_{x \in S} \pi(x) \cdot P_{x,y}, \forall y \in \Omega. \tag{11}$$

Her angir $P_{x,y}$ et element i overgangsmatrisen P, nemlig sannsynligheten for å gå fra tilstand x til tilstand y. $\pi(x)$ angir punktsannsynligheten for å være i tilstand x. Nå er situasjonen at alle verdiene $\pi(x)$ er kjente, mens P fremdeles er ukjent. Vi har også et problem siden vi har mange flere ukjente enn vi har ligninger. Gitt at det er k utfall i utfallsrommet, har vi fra P k^2 ukjente, mens vi kun har k ligninger.

Vi vet at

$$\pi(x) \cdot P_{x,y} = \pi(y) \cdot P_{y,x} \Rightarrow \pi(y) = \sum_{x \in \Omega} \pi(x) \cdot P_{x,y}. \tag{12}$$

En markovkjede som oppfyller (12) kalles tidsreversibel, og vi velger å kreve dette. Til tross for at vi krever tidsreversibilitet har vi imidlertid fremdeles flere ukjente enn ligninger. Nå velger vi overgangssannsynligheter slik at

$$P_{x,y} = Q_{x,y} \cdot \alpha_{x,y}; x \neq y, \tag{13}$$

$$P_{x,x} = 1 - \sum_{x \neq y} P_{x,y}. \tag{14}$$

Her er $Q = \{Q_{x,y}\}_{x,y \in \Omega}$ en vilkårlig overgangsmatrise og $\alpha = \{\alpha_{x,y}\}_{x,y \in \Omega}$ er sansynligheter slik at $\alpha_{x,y} \in [0, 1]$. Fra tidsreversibiliteten har vi at

$$\pi(x) \cdot Q_{x,y} \cdot \alpha_{x,y} = \pi(y) \cdot Q_{y,x} \cdot \alpha_{y,x}, y \neq x. \tag{15}$$

Hvis vi nå definerer en ny variabel $r_{x,y}$ ved

$$r_{x,y} = \pi(x) \cdot Q_{x,y} \cdot \alpha_{x,y}, \tag{16}$$

får vi at $r_{x,y} = r_{y,x}$. Videre får vi på grunn av dette at

$$\alpha_{x,y} = \frac{r_{x,y}}{\pi(x) \cdot Q_{x,y}}. \tag{17}$$

For at markovkjeden skal være tidsreversibel og for at α skal være en gyldig sannsynlighet, ser vi dermed at følgende krav må være oppfylt:

1. $r_{x,y} = r_{y,x}$
2. $\alpha_{x,y} \in [0,1] \Leftrightarrow r_{x,y} \in [0, \pi(x) \cdot Q_{x,y}]$.

Det vi er interessert i når vi gjør en MCMC-simulering er typisk et estimat for forventningsverdien til en fordeling. Vi lar $\hat{\mu}$ betegne en estimator for forventningsverdien vi er interessert i. Fra Peskun (1973) har vi at dersom man har to overgangsmatriser P og Q som begge har $\pi(x)$ som grensefordeling, og $P_{x,y} \geq Q_{x,y} \forall_{x \neq y}$ så vil $\text{Var}[\hat{\mu}_P] \leq \text{Var}[\hat{\mu}_Q]$. Dette resultatet sier i klartekst at dersom vi finner en overgangsmatrise som har gjennomgående høyere sannsynlighet for å endre tilstand enn en annen overgangsmatrise, så vil det gi sikrere estimater for parametrene vi ønsker å estimere dersom vi bruker den overgangsmatrisen. På grunn av dette resultatet er det derfor interessant å finne den verdien av $r_{x,y}$ som maksimerer $\alpha_{x,y}$ og samtidig oppfyller krav 1. og 2. ovenfor. Dette medfører at vi må maksimere $r_{x,y}$, samtidig som krav 2. er oppfylt og dette gir at vi må ha

$$\alpha_{x,y} = \frac{r_{x,y}}{\pi(x) \cdot Q_{x,y}} \leq 1 \Leftrightarrow r_{x,y} \leq \pi(y) \cdot Q_{x,y}. \tag{18}$$

Samtidig vet vi at

$$\alpha_{y,x} = \frac{r_{y,x}}{\pi(y) \cdot Q_{x,y}} \leq 1 \Leftrightarrow r_{y,x} \leq \pi(y) \cdot Q_{y,x}. \tag{19}$$

Vi ser lett at både krav 1. og 2. er oppfylt dersom vi velger $r_{x,y} = \min\{\pi(x) \cdot Q_{x,y}, \pi(y) \cdot Q_{y,x}\}$. Dersom vi setter inn dette i uttrykket for $\alpha_{x,y}$ får vi da at den optimale $\alpha_{x,y}$ er gitt ved

$$\alpha_{x,y} = \frac{1}{\pi(x) \cdot Q_{x,y}} \cdot \{\min \pi(x) \cdot Q_{x,y}, \pi(y) \cdot Q_{y,x}\} = \left\{ \min 1, \frac{\pi(y) \cdot Q_{y,x}}{\pi(x) \cdot Q_{x,y}} \right\}. \tag{20}$$

Ettersom $\pi(x)$ er kjent behøver vi bare å velge Q for å kunne simulere fra $\pi(x)$. Det har selvsagt betydning for simuleringen hvordan vi velger Q, men i prinsippet er det nok at Q er en gyldig overgangsmatrise.

2.3 Enkeltnode-oppdatering

Prinsippet bak enkeltnode-oppdatering er intuitivt. Vi ønsker å oppdatere en og en node i et MRF og når vi gjør dette antar vi at alle andre noder i feltet har kjente, fikserte verdier for hver iterasjon.

Dersom vi ønsker å gjøre en enkeltnode-oppdatering av nodene i et markovfelt, så vil dette bety at vi beregner sannsynligheten for at noden endrer tilstand, gitt naboskapet. Et viktig poeng er at vi kun oppdaterer en og en node om gangen. En vanlig metode å gjøre dette på er ved Gibbssampling. Når vi simulerer ved hjelp av en Gibbssampler så tar vi utgangspunkt i Metropolis-Hastings-algoritmen som vi utledet i forrige kapittel. Vi velger en vilkårlig node i feltet og oppdaterer noden med sannsynlighet $\alpha = 1$.

Anta et vilkårlig markovfelt med n noder, inititert med tilfeldige verdier i tidspunkt 0. Vi skriver da

$$x(0) = (x_1(0), x_2(0), ..., x_{n-1}(0), x_n(0)) \tag{21}$$

hvor $x(0)$ betegner markovkjedens tilstand i tidspunkt 0. Når vi simulerer trekker vi en tilfeldig node som vi betegner ved s. Vi får da en ny foreslått verdi $x_s(t)$, hvor s angir posisjonen i feltet, mens t betegner hvilket tidspunkt verdien er simulert fra. Siden vi alltid aksepterer det nye forslaget til tilstand, så blir da neste tilstand for feltet

$$x(t) = (x_1(t-1), x_2(t-1), ..., x_s(t), ..., x_{n-1}(t-1), x_n(t-1). \tag{22}$$

Det (22) sier er at tilstanden til feltet i tidspunkt t er den nye verdien, $x_s(t)$ kombinert med verdiene i feltet fra forrige tilstand. Merk at $x_s(t-1)$ kan være lik $x_s(t)$. Dersom denne algoritmen kjører lenge nok kan det vises at feltet vi starter med konvergerer mot et MRF med korrekte, statistiske egenskaper. Anta et markovfelt x som over. Anta også at for hver node vi ønsker å oppdatere velger ut denne noden, s, tilfeldig. I t'te iterasjon blir forslaget til ny tilstand da $x(t) = (x_1(t-1), x_2(t-1), ..., x_s(t), ..., x_{n-1}(t-1), x_n(t-1))$. Den tilstanden vi er i ved tidspunkt $t-1$ er $x(t-1) = (x_1(t-1), x_2(t-1), ..., x_s(t-1), ..., x_{n-1}(t-1), x_n(t-1))$. I et generelt markovfelt blir forslag til ny tilstand generert ut ifra overgangssmatrisen til markovfeltet. I Ising-modellen kan hver node bare være i to tilstander. Markovfeltet kan nå gå til tidspunkt t og velge mellom to tilstander i dette tidspunktet. Fra Metropolis-Hastings har vi at markovkjeden endrer tilstand fra $x(t-1)$ til $x(t)$ med sannsynlighet α og at sannsynligheten for at den ikke endrer tilstand er $1-\alpha$. Vi kaller den nye tilstanden t, altså den tilstanden markovkjeden kommer til i tidspunkt $t-1$ dersom den endrer tilstand fra tidspunkt $t-1$. La $Q_{x_s(t-1),x_s(t)}$ angi sannsynligheten i overgangsmatrisen for at markovkjeden endrer tilstand. Fra kapittel 2.2 har vi at

$$\alpha(x(i)) = \min\left(1, \frac{\pi(x_s(t) = t|x_{\partial(s)})}{\pi(x_s(t) = x_s(t-1)|x_{\partial(s)})} \cdot \frac{Q_{x_s(t-1),x_s(t)}}{P_{x_s(t),x_s(t-1)}}\right) \tag{23}$$

Hvis vi nå velger $Q_{x_s(t-1),x_s(t)} = \pi(x_s(i) = x_s(t-1)|x_{\partial(s)})$ og $Q_{x_s(t),x_s(t-1)} = \pi(x_s(t) = t|x_{\partial(s)})$ så ser vi at $\alpha(x(t))$ alltid vil være lik 1. Dette kalles gibbssampling og er den mest vanlige metoden for å utføre enkeltnode-oppdatering i et markovfelt. Det vi oppnår er at vi alltid aksepterer den foreslåtte tilstanden i hver iterasjon.

2.4 Naiv blokkoppdatering

Som vi nevnte i innledningen så er forskjellen på blokkoppdatering og enkeltnode-oppdatering at vi i blokkoppdatering oppdaterer flere noder samtidig, betinget på resten av nodene i feltet, mens vi i enkeltnode-oppdatering oppdaterer en enkelt node, gitt resten av nodene i feltet. Rent prinsipielt gjør vi blokkppdatering med en 1×1 blokk når vi gjør enkeltnode-oppdatering. Grunnen til at blokkoppdateringen gir bedre miksing enn enkeltnode-oppdatering er todelt. For det første så er simultanfordelingen til en blokk mye bredere enn den betingede fordelingen til en node. Dermed er sannsynligheten for å velge en endret verdi mye større enn tilfellet er i tilfellet med en node. For det andre så vil nodene som ligger langt unna kantene i blokken være mindre avhengig av de fikserte nodene utenfor blokken og dermed ha større frihet til å endre tilstand. I sum gir dette bedre miksing.

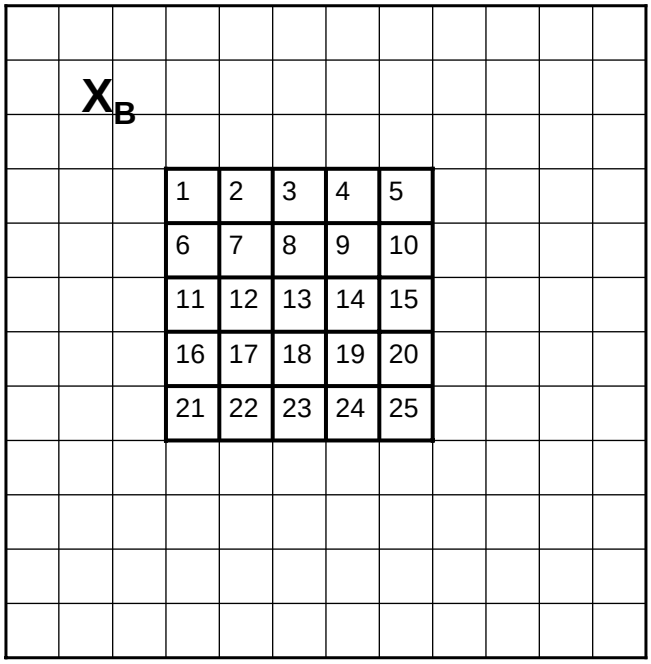
Problemet når vi blokkoppdaterer er å beregne den betingede sannsynligheten til en blokk gitt resten av nodene i feltet. Den fulle betingede fordelingen til ett enkelt punkt gitt resten av feltet er oftest kjent. Dette gjelder ikke for en blokk av vilkårlig størrelse. Derfor tar vi utgangspunkt i simultanfordelingen til et vilkårlig markovfelt.

$$\pi(x) = c \cdot \exp(V(x)). \tag{24}$$

Vi har også at

$$h(x) = \exp(V(x)). \tag{25}$$

Her er c normaliseringskonstanten til fordelingen og $V(x)$ er en nabolagsfunksjon. La n beregne antall noder i blokken og la $n = m^2$. La i og j betegne indekser til noder i et MRF. La x_B betegne alle nodene i feltet som ikke er med i blokken vi ønsker å oppdatere, og la x_A angi nodene som er med i blokken. Nodene i x_B anses som fikserte når vi simulerer. La $x_{<i}$ betegne en mengde noder i et MRF slik at $x_{<i} = \{x_j : j < i\}$. For å gjøre en blokkoppdatering ønsker vi å begynne med den noden i blokken som har lavest indeks-verdi. Vi vil da først simulere en ny verdi i denne noden ved hjelp av sannsynligheten $\pi(x_i|x_{<i}, x_B)$ som vi kjenner verdien av for alle verdier av x_i. Etter at x_i er oppdatert med en ny verdi, vil vi la den nye verdien representere en kjent verdi i feltet og således bli en del av x_B fra figur 1, altså den mengden noder som anses som konstante mens vi oppdaterer blokken. Deretter vil vi beregne $\pi(x_{i+1}|x_{<i+1}, x_B)$ på samme måte som vi beregnet $\pi(x_i|x_{<i}, x_B)$ og oppdatere punktet ved hjelp av sannsynligheten $\pi(x_{i+1}|x_{<i+1}, x_B)$. Vi vil gjenta prosessen for alle punkter i blokken inntil vi har gjort oss ferdige med siste node i blokken. I det øyeblikk vi er ferdig med siste node i blokken har vi utført en blokkoppdatering, finner en ny blokk tilfeldig og gjentar prosessen. Det er viktig å understreke at når vi utfører en blokkoppdatering av nodene i et MRF så beregner vi fremdeles individuelle sannsynligheter for hver node i blokken. Deretter benytter vi denne sannsynligheten til å simulere en verdi i den noden, før vi går videre til neste node i blokken. Til sammen utgjør



Figur 1: Figuren viser et markovfelt hvor de nodene som er kjente tilhører x_B og en 5×5 blokk i feltet.

de enkeltverdiene vi simulerer i blokken et utfall fra den betingede fordelingen til blokken gitt resten av verdiene i feltet, altså fra

$$\pi(x_A|x_B). \tag{26}$$

Hvordan beregner vi så sannsynlighetene $\pi(x_1|x_{<1}, x_B), \pi(x_2|x_{<2}, x_B)$ opp til $\pi(x_n|x_{<n}, x_B)$? I figur 1 ser vi et eksempel på hvordan et MRF kan se ut med en blokk som vi ønsker å oppdatere. Vi betinger på nodene i x_B som i hver iterasjon anses som fikserte. Strategien vi følger går ut på først å beregne $\pi(x_1|x_B)$. For å beregne denne tar vi igjen utgangspunkt i simultanfordelingen for feltet gitt i (2). Ettersom alle noder utenfor blokken i feltet anses for å være konstanter, er denne fordelingen simultanfordelingen til de stokastiske variablene $x_1, x_2, ..., x_n$. For å beregne $\pi(x_1|x_B)$ må vi summere ut de ukjente variablene i $\pi(x)$, ved å summere over utfallsrommet til hver enkelt av de ukjente variablene. Dersom vi lar Λ være utfallsrommet til variablene i markovfeltet så får vi at

$$\pi(x_i|X_B, x_{<i}) = c \cdot \sum_{x_{i+1}\in\Lambda} \sum_{x_{i+2}\in\Lambda} ... \sum_{x_{n-1}\in\Lambda} \sum_{x_n\in\Lambda} h(x_i, x_{i+1}, ..., x_{n-1}, x_n|x_B, x_{<i}) \tag{27}$$

$$\propto c \cdot \sum_{x_{i+1}\in\Lambda} \sum_{x_{i+2}\in\Lambda} ... \sum_{x_{n-1}\in\Lambda} \sum_{x_n\in\Lambda} h(x_i, x_{i+1}, ..., x_{n-1}, x_n, x_B, x_{<i}). \tag{28}$$

Vi har dermed et uttrykk for å beregne de sannsynlighetene vi trenger for å utføre en enkelt blokkoppdatering. Vi mangler imidlertid èn variabel, normaliseringskonstanten c. Vi kan imidlertid beregne den direkte etter at vi har beregnet $h(x_i|x_B, x_{<i})$, $\forall x_i \in \Lambda$ på grunn av at sannsynlighetene må summere til 1, som medfører at $c \cdot \sum_{x_i \in S} h(x_i|x_B, x_{<i}) = 1$. Vi får dermed

$$c \cdot \sum_{x_i \in \Lambda} h(x_i|x_B, x_{<i}) = 1 \tag{29}$$

$$\Rightarrow c = \frac{1}{\sum_{x_i \in \Lambda} h(x_i|x_B, x_{<i})} \tag{30}$$

som medfører at vi har alt vi trenger for å utføre en blokkoppdatering.

Det er imidlertid ikke vanskelig å se at dette er en metode som vil bruke mye beregningstid. For en vilkårlig blokk med n noder og et utfallsrom på størrelse k vil vi for å beregne $\pi(x_1|x_{<i}, x_B)$ være nødt til å summere over k^n kombinasjoner. Med et utfallsrom med 4 elementer og en blokk av størrelse $6 \times 6 = 36$ elementer blir dette altså et tall i størrelsesorden 10^{21}.

2.5 Rekursiv blokkoppdatering

I denne delen av rapporten vil vi gi en oversikt over teorien for rekursiv blokkoppdatering og diskutere nærmere hvordan man kan implementere denne metoden for å simulere MRF.

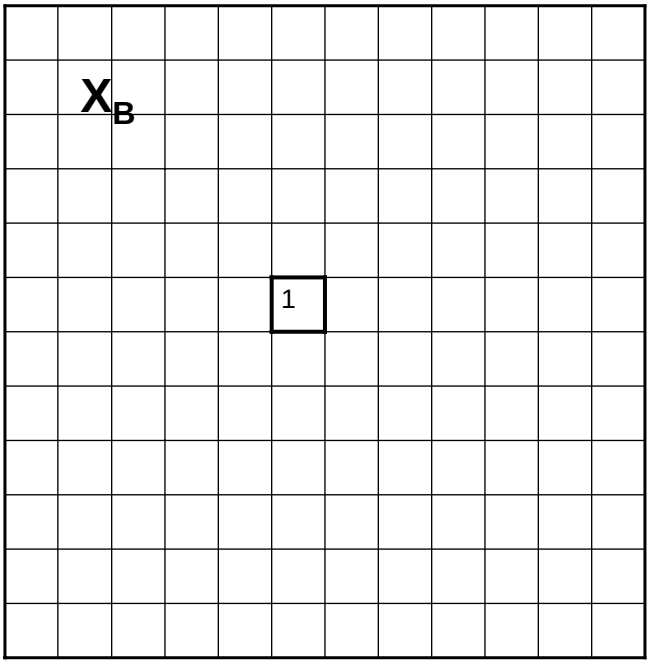
Som vi så i forrige kapittel så er mulighetene for å gjøre naiv blokkoppdatering med blokker av en viss størrelse begrensede på grunn av at beregningstiden blir høy. Dette er et problem siden gevinstene i henhold til miksing i simuleringene jevnt over øker jo større blokk man bruker. I prinsippet ønsker vi å utføre en blokkoppdatering på samme måte som når vi gjør enkeltnode-oppdatering. Vi kan si at enkeltnode-oppdatering er blokkoppdatering der vi benytter en 1×1 blokk. Dette prinsippet illustreres i figur 1 og 2. Som vi ser av figurene så er prinsippet det samme, men det som kompliserer utførelsen av blokkoppdateringen er at de enkeltvise sannsynlighetene for hver node i blokken må beregnes. Når vi gjør enkeltnode-oppdatering kjenner vi de fulle betingede sannsynlighetene til nodene. Dette er ikke tilfelle i en $n \times n$ blokk. Til grunn for den metodikken vi benytter for å utføre rekursiv blokkoppdatering, ligger et teorem fra Bartolucci (2002).

La U, V og W være diskrete, tilfeldige variabler. Da har vi at forholdet mellom dem kan skrives som

$$\pi(u|v) = \left\{ \sum_w \frac{\pi(w|u,v)}{\pi(u|v,w)} \right\}^{-1}. \tag{31}$$

Dette resultatet kan vises ved

$$\pi(w|u,v) = \frac{\pi(w,u|v)}{\pi(u|v)} = \frac{\pi(w|v) \cdot \pi(u|w,v)}{\pi(u|v)}. \tag{32}$$



Figur 2: Figuren viser et markovfelt hvor de nodene som er kjente tilhører x_B og x_A er en 1×1 blokk i feltet.

Videre får vi ved innsetting av (32) i (31) at

$$\left\{ \Sigma_w \frac{\pi(w|u,v)}{\pi(u|v,w)} \right\}^{-1} = \left\{ \sum_w \frac{\pi(w|v) \cdot \pi(u|w,v)}{\pi(u|v) \cdot \pi(u|w,v)} \right\}^{-1} \tag{33}$$

Dersom vi forkorter teller mot nevner på høyre side i (33) står vi igjen med

$$\left\{ \frac{\sum_w \pi(w|v)}{\pi(u,v)} \right\}^{-1} = \left\{ \frac{1}{\pi(u,v)} \right\}^{-1} = \pi(u|v). \tag{34}$$

Vi vil nå anvende dette resultatet på en bestemt måte for å utføre rekursiv blokkoppdatering. La x^j angi alle nodene i et MRF x med indeks mindre eller lik j. Hvis vi nå i en blokk med n elementer i et MRF lar $u = x_i$, $v = x_{<j}$ og $w = x_j$ så får vi at

$$\pi(x_i|x_{<j}, x_B) = \left\{ \sum_{x_j \in \Lambda} \frac{\pi(x_j|x_{<j}, x_B)}{\pi(x_i|x_{<j+1}), x_B} \right\}^{-1}. \tag{35}$$

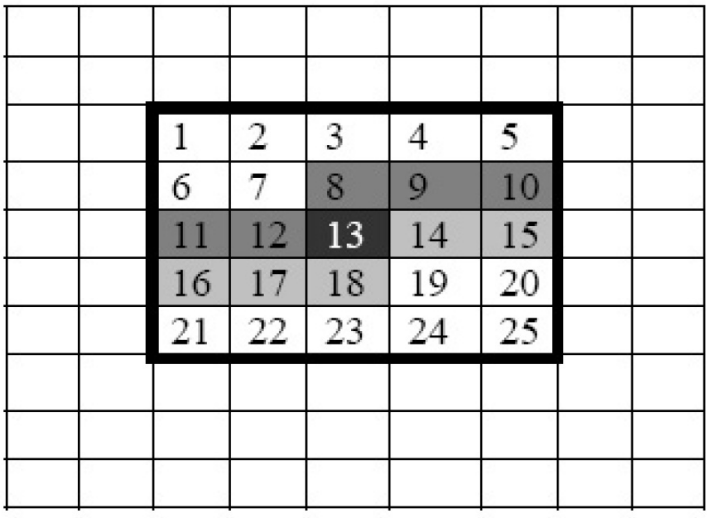
Det som i praksis står her er at den betingede sannsynligheten for en node i blokken gitt alle andre noder i blokken opp til node j, samt de fikserte nodene i feltet utenfor blokken, kan beregnes ved å summere ut x_j. Dette betyr selvsagt at vi må beregne $\pi(x_j|x_{<j})$ og $\pi(x_i|x_{<j+1})$ først. Og ved å erkjenne dette er vi ved

kjernen i hvordan vi kan benytte dette resultatet til blokkoppdatering av et MRF. Hvis vi begynner i node n i en blokk i et MRF, altså den noden som har høyest indeks i blokken, så kan den fulle betingede fordelingen for denne noden beregnes, altså $\pi(x_n|x_{<n}, x_B)$. Deretter beregner vi så den fulle betingede sannsynligheten for node $n-1$ i feltet, altså $\pi(x_{n-1}|x_{<n}, x_B)$ for alle verdier av node n. Vi har da ved å benytte (35) at

$$\pi(x_{n-1}|x_{<n-1}, x_B) = \left\{ \sum_{x_n} \frac{\pi(x_n|x_{<n}, x_B)}{\pi(x_{n-1}|x_{<n}, x_B)} \right\}^{-1}. \tag{36}$$

Hvis vi nå benytter den samme teknikken for node $n-2$ og ned til node 1 i blokken så står vi til slutt igjen med $\pi(x_1|x_B)$. Denne sannsynligheten benytter vi så til å simulere en ny verdi i node 1. Når vi har gjort dette, så kan vi hente ut en sannsynlighet vi allerede har beregnet for å simulere neste node i blokken, altså 2, gitt verdien vi simulerte i node 1. Grunnen til at vi kun henter ut en sannsynlighet vi allerede har beregnet når vi skal simulere den 2. noden i blokken er at når vi beregner oss nedover i blokken fra node n til node 1 så må vi samtidig beregne sannsynlighetene til den noden vi står i betinget på alle noder den er avhengig av. Et eksempel på hvordan en slik avhengighetsstruktur kan se ut er illustrert i figur 3. Her har vi en 5×5 blokk. Vi må nå for hver node i blokken fra node 25 til node 1 beregne $\pi(x_i|x_{<i})$ ved hjelp av (36). For å beregne disse sannsynlighetene riktig må vi begynne med å beregne den fulle betingede sannsynligheten til nodene, og disse må beregnes for alle konfigurasjoner av naboene som er inne i blokken. Det vil si at vi må beregne sannsynlighetene for alle verdier naboene kan ha i utfallsrommet Λ. Generelt må hver sannsynlighet beregnes for alle konfigurasjoner av de nodene som vi betinger på som noden ikke er betinget uavhengig av. Dette kan illustreres med figur 3. I denne illustrasjonen ønsker vi å beregne $P(x_{13}|x_{<13}, x_B)$. De nodene i blokken som x_{13} er avhengig av eller blir avhengig av etterhvert som vi summerer ut nodene ved hjelp av (35) er merket i mørkt grått. Vi må først og fremst beregne $P(x_{13}|x_8, x_{12}, x_{14}, x_{18})$ og deretter benytte denne til å summere ut alle variablene merket med lyst grått, det vil si nodene $14, 15, 16, 17, 18$. Dette må gjøres for alle konfigurasjoner av nodene $13, 14, 15, 16, 17, 18$. I tillegg må vi ta hensyn til de nodene som har lavere indeksverdi enn 13, men som vil komme til å bli avhengige av x_{13}. Dette gjelder nodene fra og med node 8 til og med node 12. Disse er merket i en mørk gråfarge. Dermed er den totale mengden noder som denne beregningen innbefatter 10 noder, i tillegg til noden selv.

Denne fremgangsmåten må følges for alle nodene i blokken. Først må vi identifisere de nodene som vil inngå i beregningen, deretter beregner vi sannsynlighetene $P(x_i|x_{<i}, x_B)$. Disse sannsynlighetene tar vi siden vare på. Grunnen til at vi tar vare på de er at vi benytter dem til å simulere nye verdier i noden de representerer når vi har funnet $P(x_1|x_B)$ og simulerer oss ned til x_{25} i blokken igjen. Disse sannsynlighetene kan da benyttes direkte til å simulere nye verdier i nodene, gitt at vi allerede har simulert verdiene i blokken som har lavere in-

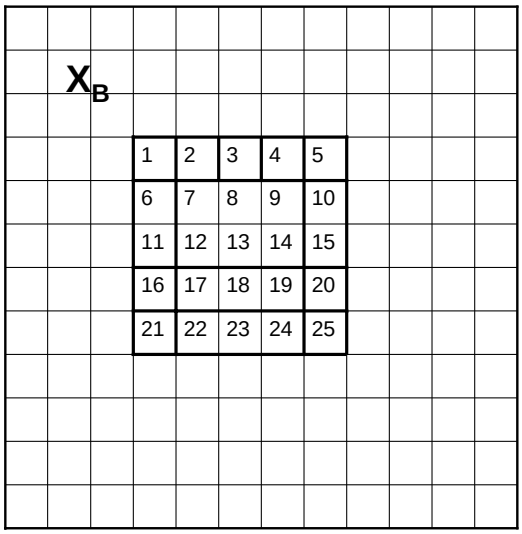


Figur 3: Figuren viser et markovfelt hvor de nodene som er kjente tilhører x_B samt en 5×5 blokk i feltet. De nodene som har lavere indeks enn x_{13} og som samtidig inngår i beregningen av $P(x_{13}|x_{<13}, x_B)$ er merket i en lys grå farge. Vi har merket nodene som har høyere indeksverdi enn 13 i en mørk grå farge. Node 13 er merket spesielt i en veldig mørk farge.

deksverdi enn noden vi står i. Da vi skulle implementere denne metoden forsøkte vi å gjøre dette på en så generell måte som mulig. Det viste seg at mange av nodene i prinsippet kunne beregnes på samme måte. I praksis måtte vi imidlertid gjøre mange spesialtilpasninger i forhold til de individuelle nodene og hvordan de kunne beregnes. Til slutt kom vi frem til et skjema som illustreres i figur 4. Det figuren viser er i praksis at de to nederste radene i en blokk må behandles spesielt. I tillegg må første rad, altså øverste, behandles spesielt. Den delen av blokken som kan være gjenstand for reell generalisering er radene mellom andre og nest siste rad. I disse radene vil nodene og deres tilhørende sannsynligheter behandles likt både i teori og praksis. Det er også slik vi har implementert rekursiv blokkoppdatering. For mer detaljert informasjon om hvordan vi har implementert rekursiv blokkoppdatering henviser vi til Appendiks.

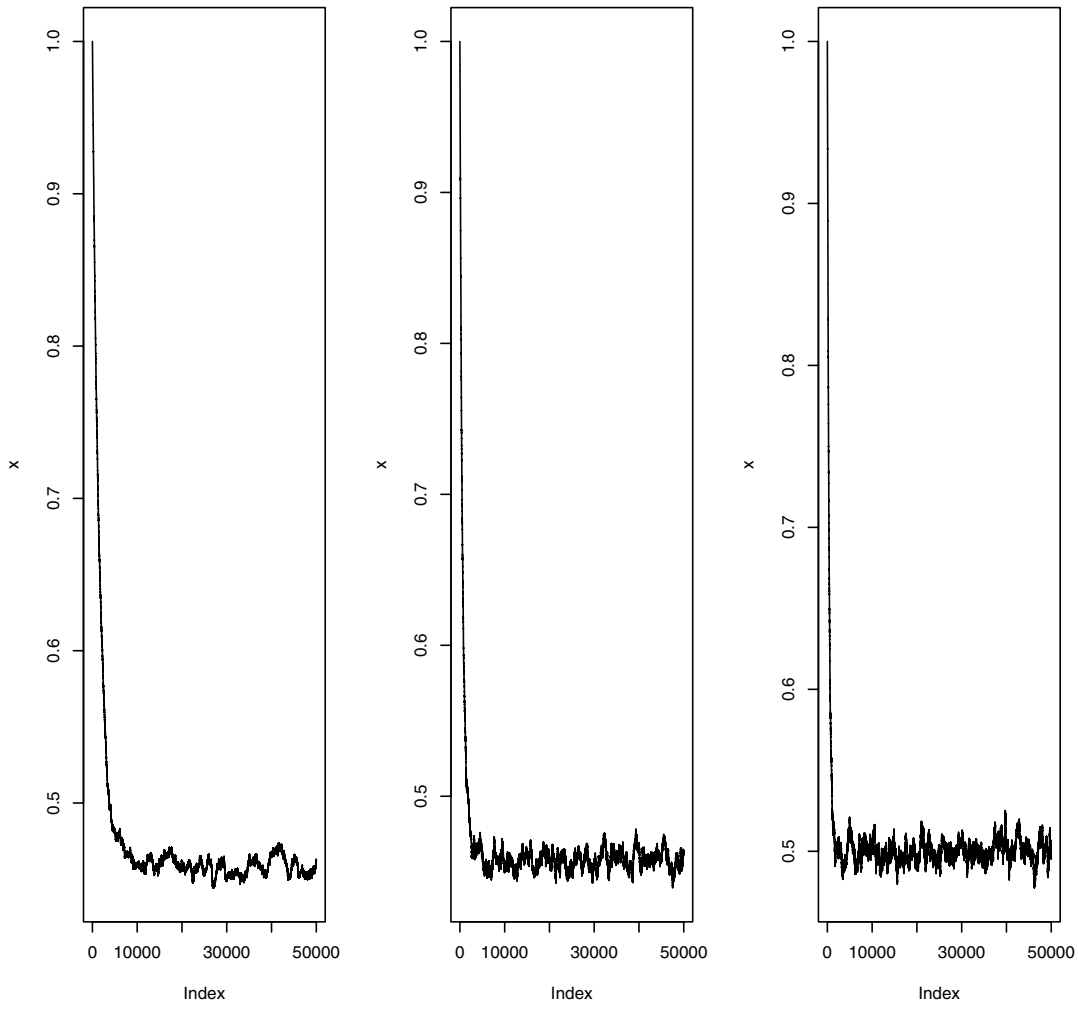
3 Resultater

Vi gjorde alle simuleringer på et markovfelt med 100×100 noder. Vi fulgte samme skjema for de simuleringene vi gjorde med naiv blokkoppdatering og rekursiv blokkoppdatering, hvor vi simulerte for tre forskjellige verdier av β for blokker av størrelse 3×3, 4×4 og 5×5. Figurene viser hvor stor prosent av nodene i feltet som har verdien 1. Grunnen til at vi benytter et slikt mål er at dette gir en indikator på når feltet konvergerer med forskjellige verdier for β. For β verdi

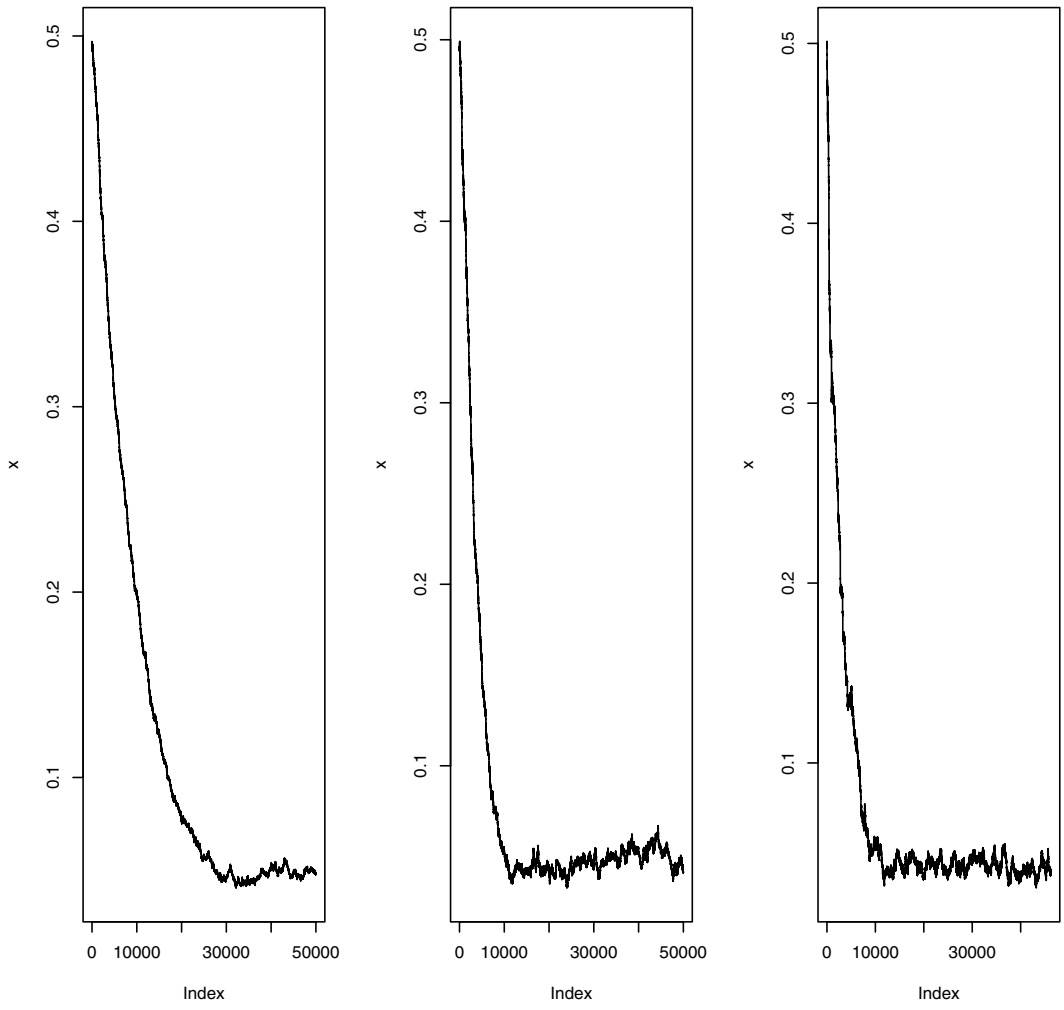


Figur 4: Figuren viser et markovfelt hvor de nodene som er kjente tilhører x_B samt en 5×5 blokk i feltet. De nodene som ble implementert på samme måte er innlemmet i samme rektangel av fet strek.

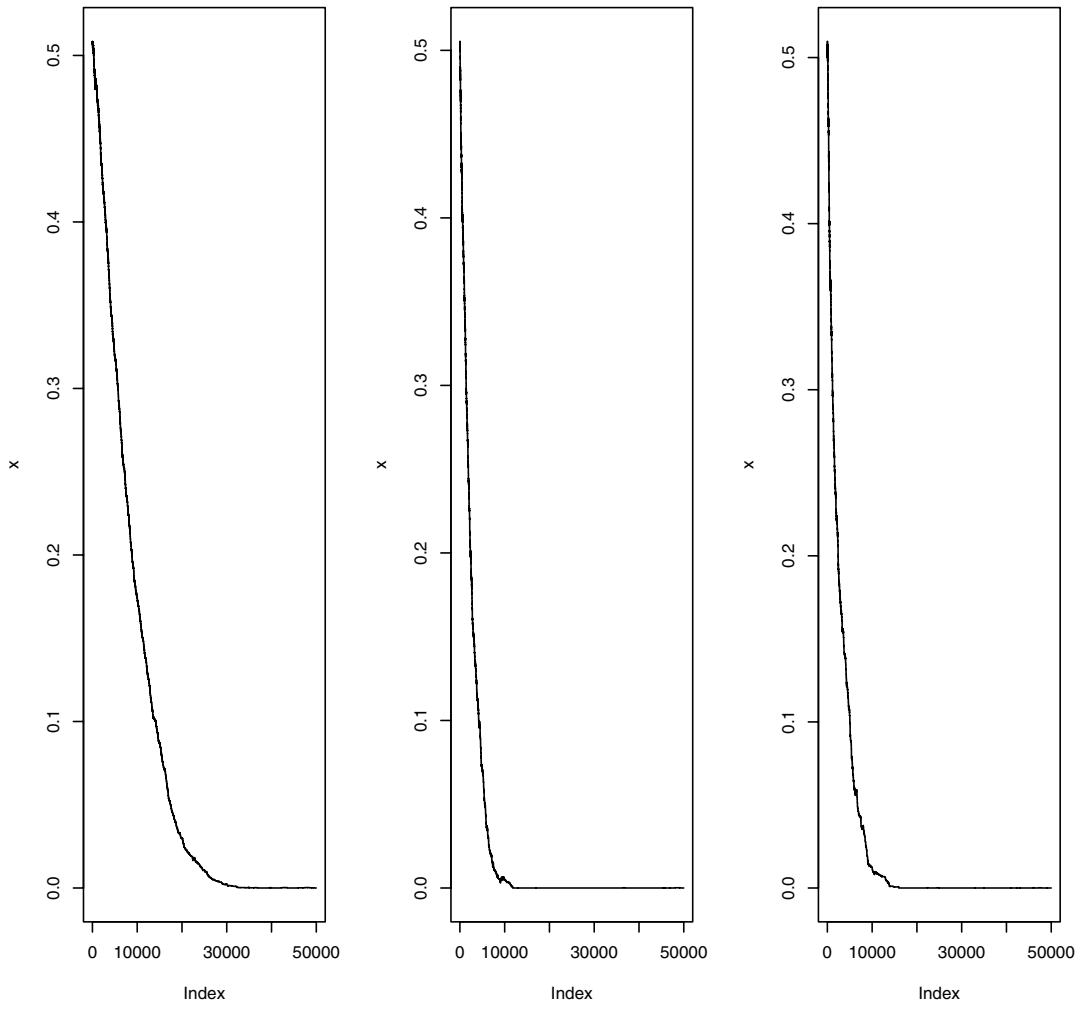
0.1 skal feltet konvergere mot 50% av verdien 0 og 1. Med $\beta = 1$ eller $\beta = 3$ skal feltet konvergere mot tilnærmet 100% av kun en av verdiene. Når vi simulerer med $\beta = 0.1$ bruker vi et felt med samtlige verdier lik 1 som starttilstand for å se hvor raskt feltet konvergerer mot tilnærmet 50% av verdiene til 1. Når vi simulerer med $\beta = 1$ eller $\beta = 3$ bruker vi en starttilstand der feltet er jevnt fordelt mellom 0 og 1 for å se hvor raskt feltet konvergerer mot tilnærmet 100% av kun en av verdiene. I tillegg simulerte vi en serie med enkeltnode-oppdatering for å kunne sammenligne antall iterasjoner som kreves for konvergens. I figur 5 ser vi tre serier med rekursiv blokkoppdatering med $\beta = 0.1$. I første plott ser vi simuleringen for en 3×3 blokk, i den andre for en 4×4 blokk og i det tredje ser vi simulering med en 5×5 blokk. I figur 6 ser vi tilsvarende plott for $\beta = 1$ og i figur 7 ser vi dette for $\beta = 3$. I figur 8, 9 og 10 ser vi de tilsvarende grafene for simuleringer med naiv blokkoppdatering. Til slutt kan vi i figur 11 se en tilsvarende serie for enkeltnode-oppdatering. Merk her at antall iterasjoner må multipliseres med 100. I tabell 1 har vi samlet de dataene vi ønsker å sammenligne mellom de tre metodene, hvor mange iterasjoner som trengs før feltet konvergerer og hvor lang tid hver iterasjon tar.



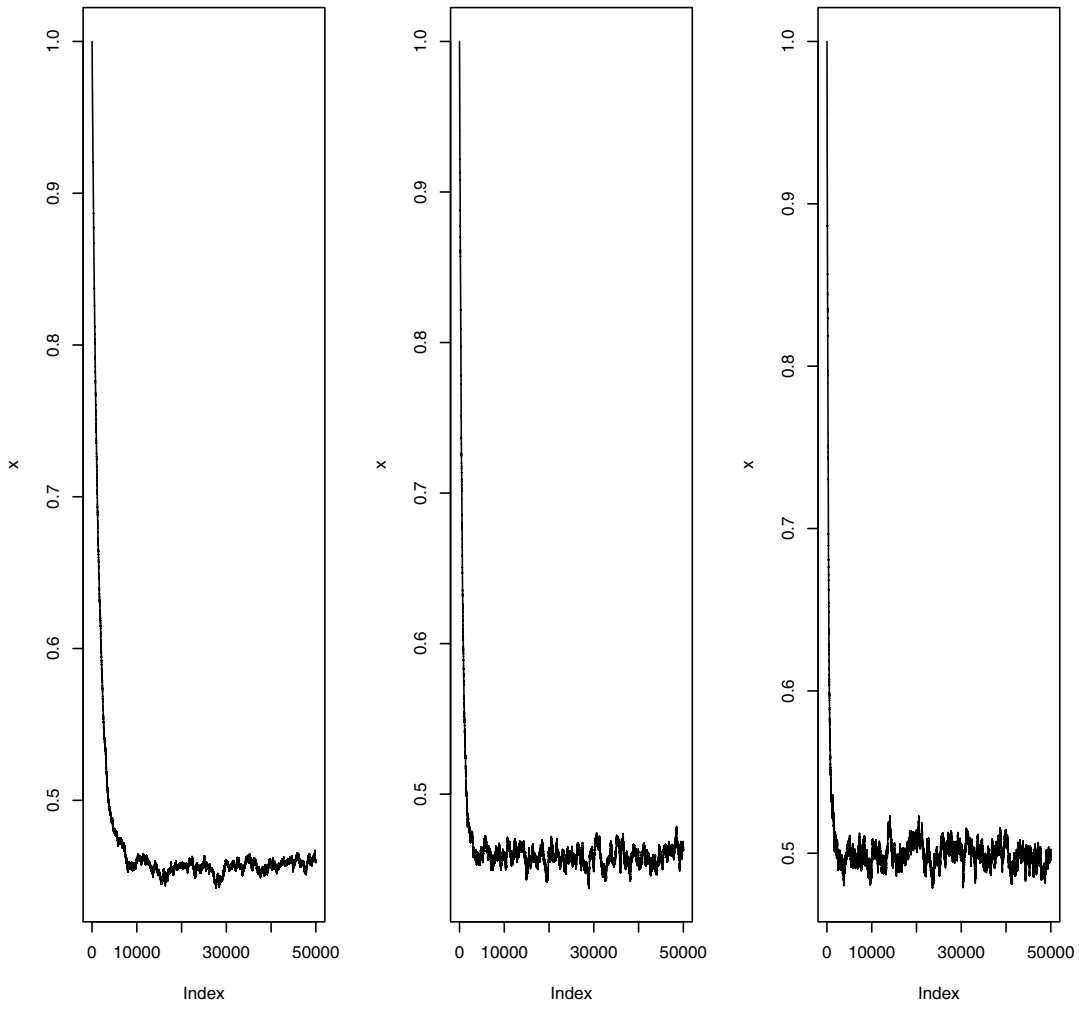
Figur 5: Figuren viser en serie simuleringer med rekursiv blokkoppdatering hvor vi bruker 50000 iterasjoner med henholdsvis en 3×3 blokk, en 4×4 blokk og en 5×5 blokk. Vi har brukt $\beta = 0.1$. Y-aksen viser hvor stor andel av nodene i feltet som har verdien 1.



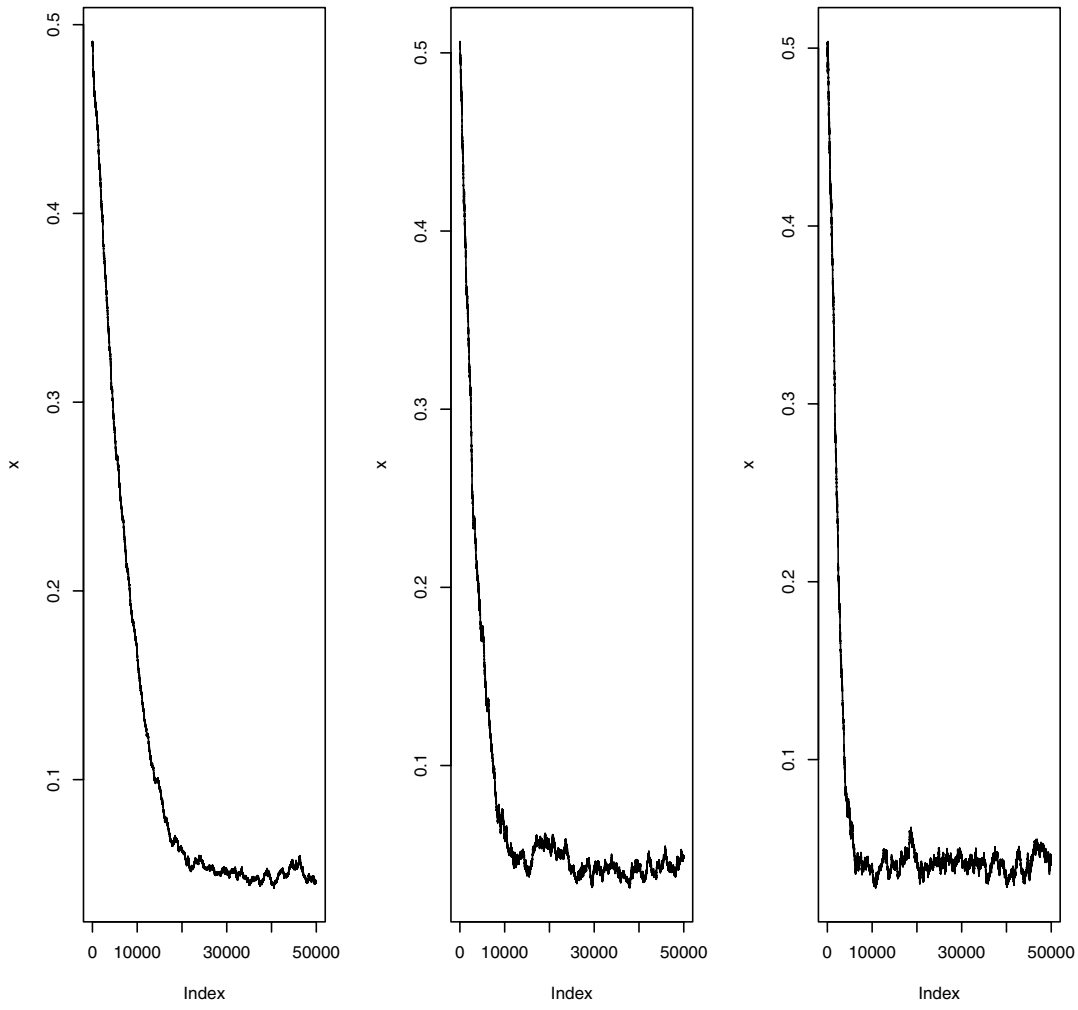
Figur 6: Figuren viser en serie simuleringer med rekursiv blokkoppdatering hvor vi bruker 50000 iterasjoner med henholdsvis en 3×3 blokk, en 4×4 blokk og en 5×5 blokk. Vi har brukt $\beta = 1$. Y-aksen viser hvor stor andel av nodene i feltet som har verdien 1.



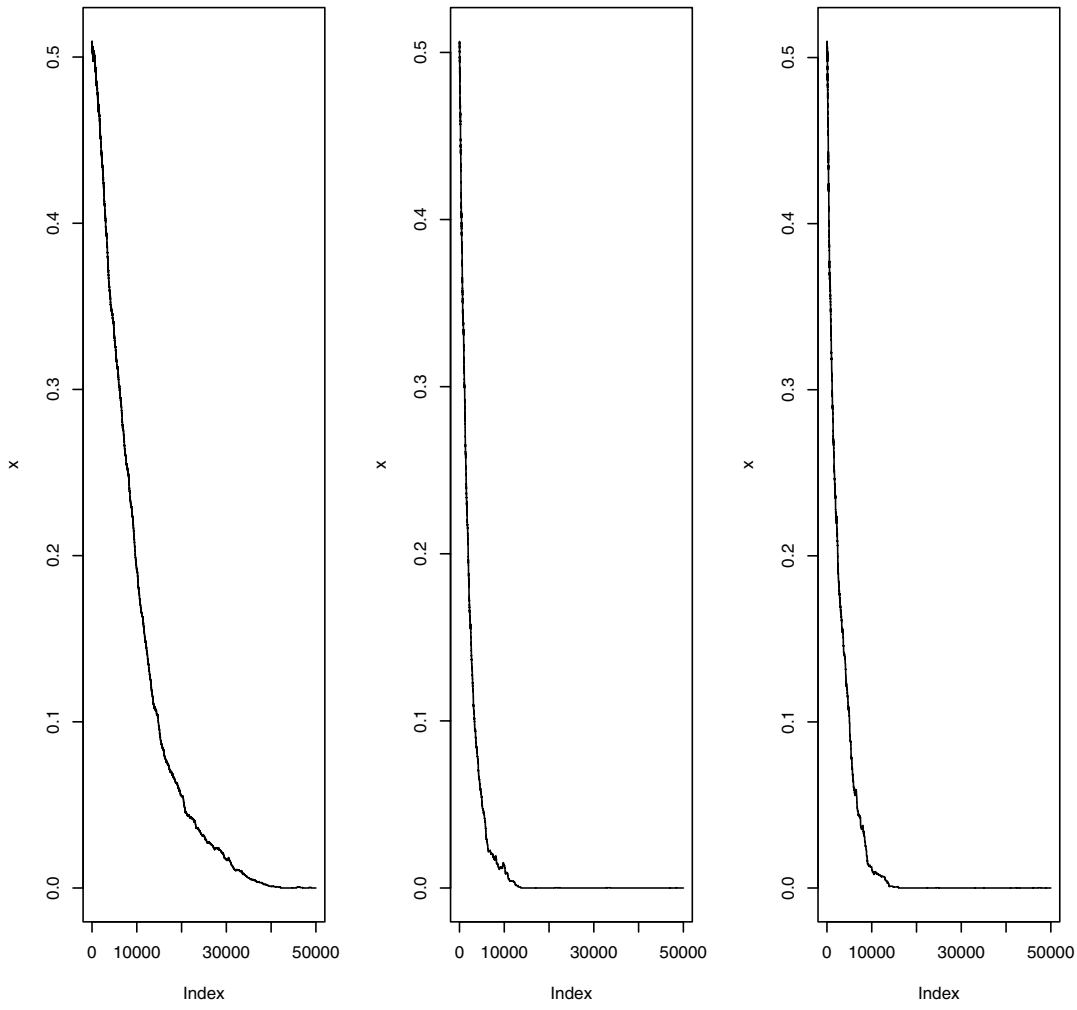
Figur 7: Figuren viser en serie simuleringer med rekursiv blokkoppdatering hvor vi bruker 50000 iterasjoner med henholdsvis en 3×3 blokk, en 4×4 blokk og en 5×5 blokk. Vi har brukt $\beta = 3$. Y-aksen viser hvor stor andel av nodene i feltet som har verdien 1.



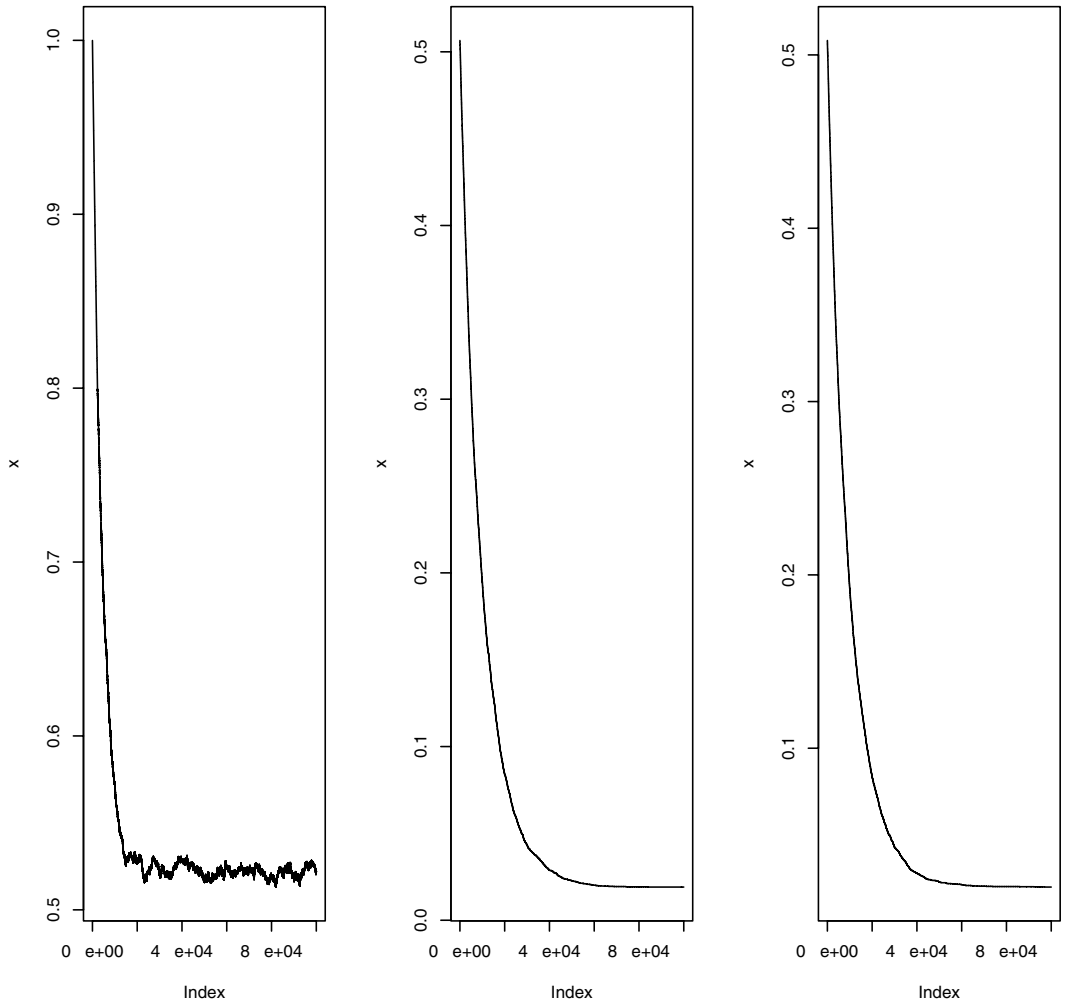
Figur 8: Figuren viser en serie simuleringer med naiv blokkoppdatering hvor vi bruker 50000 iterasjoner med henholdsvis en 3×3 blokk, en 4×4 blokk og en 5×5 blokk. Vi har brukt $\beta = 0.1$. Y-aksen viser hvor stor andel av nodene i feltet som har verdien 1.



Figur 9: Figuren viser en serie simuleringer med naiv blokkoppdatering hvor vi bruker 50000 iterasjoner med henholdsvis en 3×3 blokk, en 4×4 blokk og en 5×5 blokk. Vi har brukt $\beta = 1$. Y-aksen viser hvor stor andel av nodene i feltet som har verdien 1.



Figur 10: Figuren viser en serie simuleringer med naiv blokkoppdatering hvor vi bruker 50000 iterasjoner med henholdsvis en 3×3 blokk, en 4×4 blokk og en 5×5 blokk. Vi har brukt $\beta = 3$. Y-aksen viser hvor stor andel av nodene i feltet som har verdien 1.



Figur 11: Figuren viser en serie med simuleringer hvor vi bruker 100000 iterasjoner med enkeltnode-oppdatering og fra venstre $\beta = 0.1$, $\beta = 1$ og $\beta = 3$. Y-aksen viser hvor stor andel av nodene i feltet som har verdien 1.

Metode	Blokkstørrelse	β	Konvergens	Sek. til konvergens	Sek. pr iterasjon
Enkeltnode	1×1	0.1	50000	5.5	0.00011
Enkeltnode	1×1	1	60000	6.6	0.00011
Enkeltnode	1×1	3	60000	6.6	0.00011
Naiv	3×3	0.1	15000	15	0.001
Rekursiv	3×3	0.1	11000	3.74	0.00034
Naiv	4×4	0.1	7000	910	0.13
Rekursiv	4×4	0.1	9000	71.82	0.00798
Naiv	5×5	0.1	2000	182000	91.0
Rekursiv	5×5	0.1	1500	42.36	0.02824
Naiv	3×3	1	20000	20	0.001
Rekursiv	3×3	1	25000	8.5	0.00034
Naiv	4×4	1	12000	1560	0.13
Rekursiv	4×4	1	9500	75.81	0.00798
Naiv	5×5	1	4000	364000	91.0
Rekursiv	5×5	1	5500	155.32	0.02824
Naiv	3×3	3	25000	25	0.001
Rekursiv	3×3	3	25000	8.5	0.00034
Naiv	4×4	3	12000	1560	0.13
Rekursiv	4×4	3	8500	67.83	0.00798
Naiv	5×5	3	5000	455000	91.0
Rekursiv	5×5	3	5000	141.2	0.02824

Tabell 1: Tabellen viser de data vi hentet ut fra simuleringene vi har vist i figur 5-11

Det vi ser fra tabellen er at enkeltnode-oppdatering faktisk er den raskeste metoden i forhold til kjøretid for Isingmodellen. I dette tilfellet konvergerer feltet etter 60000 iterasjoner med denne metoden. Dersom vi tar den billige kjøretiden pr iterasjon i betraktning ser vi at det billigste med hensyn på kjøretid for denne modellen vil være å benytte enkeltnode-oppdatering. Dette var et uventet resultat. Vi hadde forventet at rekursiv blokkoppdatering skulle være raskere enn enkeltnode-oppdatering. Vi tror imidlertid dette ville vært tilfelle dersom vi hadde simulert rekursiv blokkoppdatering med en større blokk, eller dersom vi hadde prøvd ut denne metoden for en modell hvor enkeltnode-oppdatering har reelle problemer med miksing. Dersom vi sammenligner de to metodene for blokkoppdatering så ser vi at for 3×3 blokken så er naiv blokkoppdatering ca 3 ganger dyrere med hensyn på kjøretid pr iterasjon enn rekursiv blokkoppdatering. For 4×4 blokken er dette forholdet økt til størrelsesorden 10^1. For en 5×5 blokk er dette forholdet økt til 10^3.

Hvis vi sammenligner tiden det tar før feltet konvergerer med de forskjellige metodene så ser vi først og fremst at enkeltnode-oppdatering konvergerer hurtigst for denne modellen. Det som er mer interessant er å sammenligne de to metodene for blokkoppdatering. Her ser vi en drastisk økning i tiden det tar før feltet

konvergerer med naiv blokkoppdatering fra 8.5 til 20 sekunder for en 3×3 blokk til hele 455000 sekunder for en 5×5 blokk. Det er interessant å merke seg at tiden det tar før feltet konvergerer med en 5×5 blokk stiger til maksimalt 155.32 sekunder for en 5×5 blokk med rekursiv blokkoppdatering.

4 Diskusjon

Når vi sammenligner resultatene for de tre metodene vi har undersøkt, enkeltnode-oppdatering, naiv blokkoppdatering og rekursiv blokkoppdatering så ser vi at enkeltnode-oppdatering gjør hver iterasjon på kortest tid. Dette var forsåvidt som ventet, og det samme var det faktum at vi oppnår mindre miksing pr iterasjon ved å benytte denne metoden. I modeller hvor det er betraktelig større avhengighet mellom variablene vi simulerer enn det som er tilfelle i Isingmodellen, er denne metoden ikke mulig å benytte. Hver iterasjon vil fremdeles ta mye kortere tid enn hver enkelt iterasjon med de to metodene for blokkoppdatering, men vi vil måtte simulere mye lenger for at markovfeltet skal konvergere. Derfor er det mest interessant å sammenligne de to metodene for blokkoppdatering. Ut fra resultatene ser vi at gevinsten med å bruke rekursiv blokkoppdatering blir større for hver dimensjon vi øker blokken med. I tabell 1 kan vi se at de to metodene presterer like godt når det kommer til miksing, hvilket også er som ventet ettersom de simulerer feltet på samme måte. For naiv blokkoppdatering så ser vi at kjøretiden øker med 130 ganger fra en 3×3 blokk til en 4×4 blokk, og med 700 ganger for en 5×5 blokk fra en 4×4 blokk. Til sammen øker kjøretiden med 91000 ganger pr iterasjon fra en 3×3 blokk til en 5×5 blokk. For de simuleringene vi utførte med rekursiv blokkoppdatering, øker kjøretiden pr iterasjon med 23 ganger fra en 3×3 blokk til en 4×4 blokk og med 3.53 ganger fra en 4×4 blokk til en 5×5 blokk. Til sammen øker kjøretiden pr iterasjon med 83 ganger fra en 3×3 blokk til en 5×5 blokk når vi simulerer med Isingmodellen.

Som vi kan se så er det stor forskjell på de to metodene når det gjelder kjøretid pr iterasjon. Ut ifra de resultatene vi har, er det naturlig å tro at denne forskjellen vil bli enda større dersom vi øker antall dimensjoner på blokken vi bruker. Dette er interessant også for andre modeller hvor vi simulerer markovfelt. Isingmodellen er en modell som ikke er veldig krevende i forhold til beregningsmengde. Dersom man benytter en modell med større nabolag der det er stor avhengighet mellom variablene og meget treg miksing ved bruk av enkeltnode-oppdatering vil vi få et problem dersom vi prøver å benytte naiv blokkoppdatering. Sannsynligvis vil vi ikke engang kunne simulere med en 5×5 blokk. En annen situasjon som kan være meget interessant med tanke på bruk av rekursiv blokkoppdatering er dersom vi vil simulere markovfelt i 3 dimensjoner. I en slik situasjon vil selv en $2 \times 2 \times 2$ blokk inneholde 8 noder og med de resultatene vi har sett i denne rapporten er ikke en slik blokkstørrelse håndterbart dersom vi benytter naiv blokkoppdatering.

Implementasjonen av de to metodene er også svært forskjellig. Å implementere

naiv blokkoppdatering er forholdsvis ukomplisert. Å implementere rekursiv blokkoppdatering har vært en tidkrevende affære selv for en såpass elementær modell som Isingmodellen. Dette må selvsagt tas i betraktning dersom man ønsker å benytte denne metoden på andre modeller. Vi har beskrevet selve implementeringen av den rekursive blokkoppdateringen i detalj i Appendiks.

Referanser

Besag, J. (1974). Spatial interaction and statistical-analysis of lattice systems, *Journal of the Royal Statistical Society Series B-Methodological* **36**(2).

Peskun, P. H. (1973). Optimum Monte-Carlo Sambling Using Markov Chains, *Biometrika* **64**(3).

Bartolucci, F., Besag, J. (2002). A recursive algorithm for Markov random fields, *Biometrika* **89**(3).

APPENDIKS

I appendikset skal vi forklare mer detaljert hvordan vi implementerte rekursiv blokkoppdatering. Som nevnt i rapporten så har vi 12 distinkte måter å beregne sannsynlighetene i blokken på. Vi vil forklare litt mer detaljert hvordan vi har beregnet disse. Noen av disse måtene er relativt enkle å forstå, mens andre er generalisert for å omhandle så mange noder som mulig. Vi tar her utgangspunkt i en blokk av størrelse $n = m \times m$. Vi ønsker å beregne alle sannsynligheter i blokken som er slik at for en vilkårlig node i så har vi $P(x_i|x_{<i}, x_B)$. Vi indekserer alle disse sannsynlighetene i en matrise med tre dimensjoner, hvor den første dimensjonen angir indeksen til noden vi står i, den andre dimensjonen angir en indeks for hvilken konfigurasjon av nabolaget vi bruker og den tredje dimensjonen angir hvor mange noder vi har igjen å summere ut ved hjelp av (35). Måten vi indekserer på er at vi tar utgangspunkt i blokkstørrelsen og sier at den noden vi står i har rangen m. Når vi beregner indeksen I for konfigurasjonen av noder får vi da

$$I = x_{i-m}\cdot 2^0 + x_{i-m+1}\cdot 2^1 + ... + x_{i-1}\cdot 2^{m-1} + x_i\cdot 2^m + ... + x_{i+m-1}\cdot 2^{m+m-1} + x_{i+m}\cdot 2^{m+m}. \tag{A-1}$$

Selvsagt vil hvilke noder som inngår i indeksen være avhengig av hvilke noder som er med i hver iterasjon. Grunnen til at vi gjør dette er fordi vi ønsker å lagre alle sannsynligheter med en unik identitet. Denne unikheten frembringer vi med å bruke et polynom med grunntall 2. Som vi ser av (A-1) så er denne indeksen avhengig av verdien av nodene som er med i beregningen, slik at ingen konfigurasjon av noder vil gi samme verdi.

Når vi skal beregne sannsynlighetene vi behøver for å gjøre en blokkoppdatering, begynner vi med siste node i blokken, altså den noden som har høyest indeks. Den er plassert nederst til høyre i blokken. Vi skal beregne $P(x_n|x_{<n}, x_B)$. Siden dette er siste node i blokken slipper vi å benytte formelen gitt i (35) og kan beregne sannsynligheten ved å benytte den fulle betingede fordelingen, gitt i (18) og (19). Hvis vi ser tilbake på figur 3, så må vi følge tilsvarende skjema også for x_n og beregne $P(x_n|x_{<n}, x_B) = P(x_n|x_{n-1}, x_{n-m}, x_B)$, altså for alle mulige verdier x_n, x_{n-1} og x_{n-m} kan ta. Indeksen I blir da beregnet ved $x_{n-m}\cdot 2^0 + x_{n-1}\cdot 2^{m-1} + x_n\cdot 2^m$. Til slutt vil vi da lagre $2^3 = 8$ sannsynligheter for $P(x_n|x_{n-1}, x_{n-m}, x_B)$, noe som gjelder uansett blokkstørrelse for Isingmodellen. Grunnen til at vi lagrer alle verdiene er todelt. Den ene grunnen er at vi kommer til å beregne sannsynlighetene for alle mulige konfigurasjoner av nodene som den noden vi ønsker å beregne sannsynligheten for er avhengig av. Dette er en følge av at når vi har beregnet sannsynligheten $P(x_1|x_B)$ for den noden med lavest indeks i blokken, skal vi simulere nye verdier for alle nodene i blokken. Når vi skal gjøre dette og har beregnet sannsynlighetene for alle konfigurasjonene av nodene som den noden vi ønsker å simulere er avhengig av, kan vi hente ut en sannsynlighet vi allerede har beregnet ved å se på de allerede simulerte verdiene i blokken og sette dem

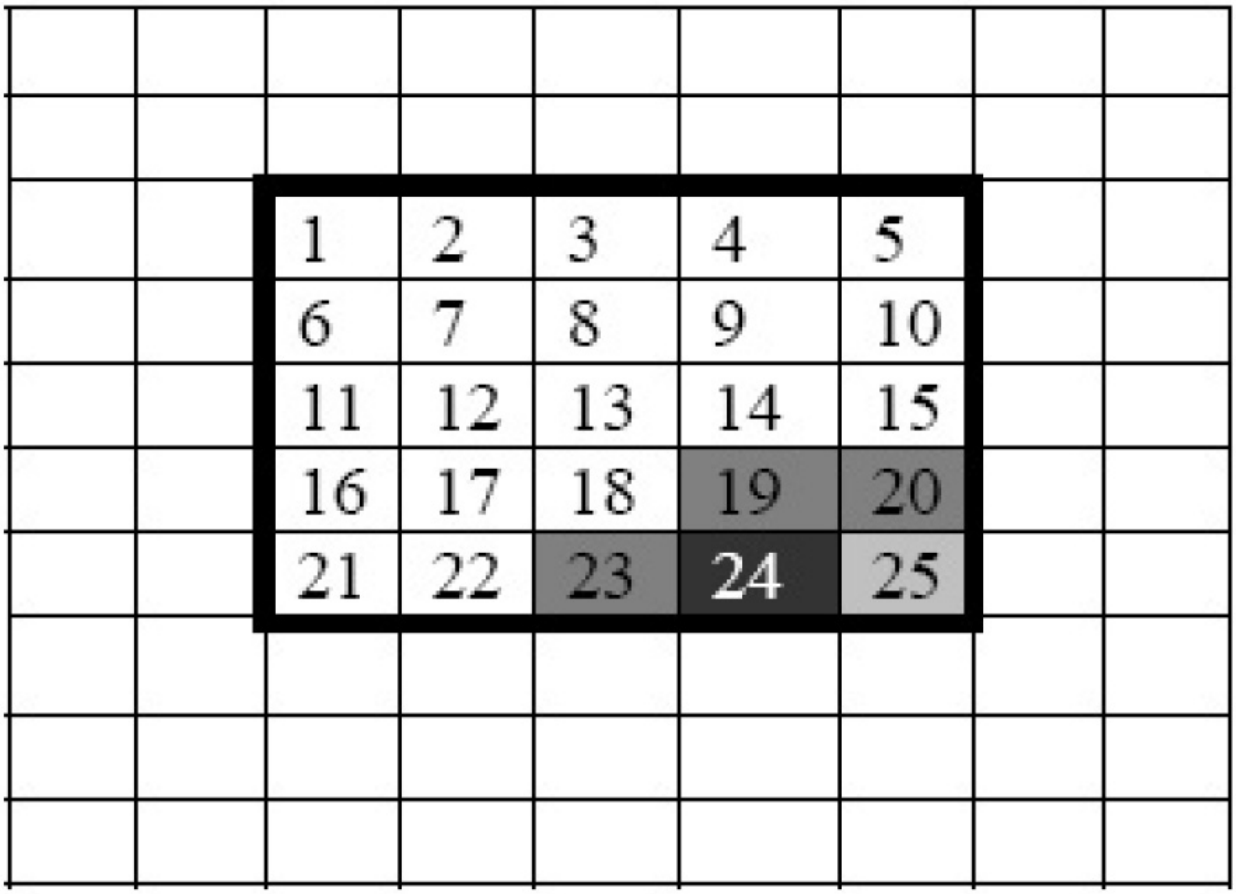
inn i (A-1) og ved hjelp av den unike indeksen hente ut riktig sannsynlighet og simulere en ny verdi for noden vi står i direkte.

Videre så har nodene mellom siste node i blokken og første node på siste rad, $n - m + 1$, like egenskaper når vi skal beregne sannsynligheter. Vi utviklet derfor en generell algoritme for å beregne disse sannsynlighetene som tilpasser seg blokkstørrelsen. Vi lar en ytre løkke gå gjennom de $m - 2$ nodene vi skal beregne sannsynlighetene for. Det som kompliserer denne algoritmen er at nodene får et forskjellig antall noder som de blir avhengige av. Når vi beregner $P(x_{n-1}|x_{<n-1}, x_B)$ behøver vi kun å summere ut node n. Når vi beregner $P(x_{n-2}|x_{<n-2}, x_B)$, må vi summere ut node $n - 1$. Dette medfører at noden vi står i, blir avhengig av tilsvarende flere noder på raden over, som selvsagt må inkluderes i indeksen, I. I figur 12 ser vi hvordan oppsettet blir for den første noden vi beregner av de nodene som er mellom hjørnene på siste rad. Som vi ser av figuren så blir node 24 avhengig av node 20 når vi summerer ut node 25. Tilsvarende blir node 23 avhengig av node 19 og 20, samt 18, når vi summerer ut node 24.

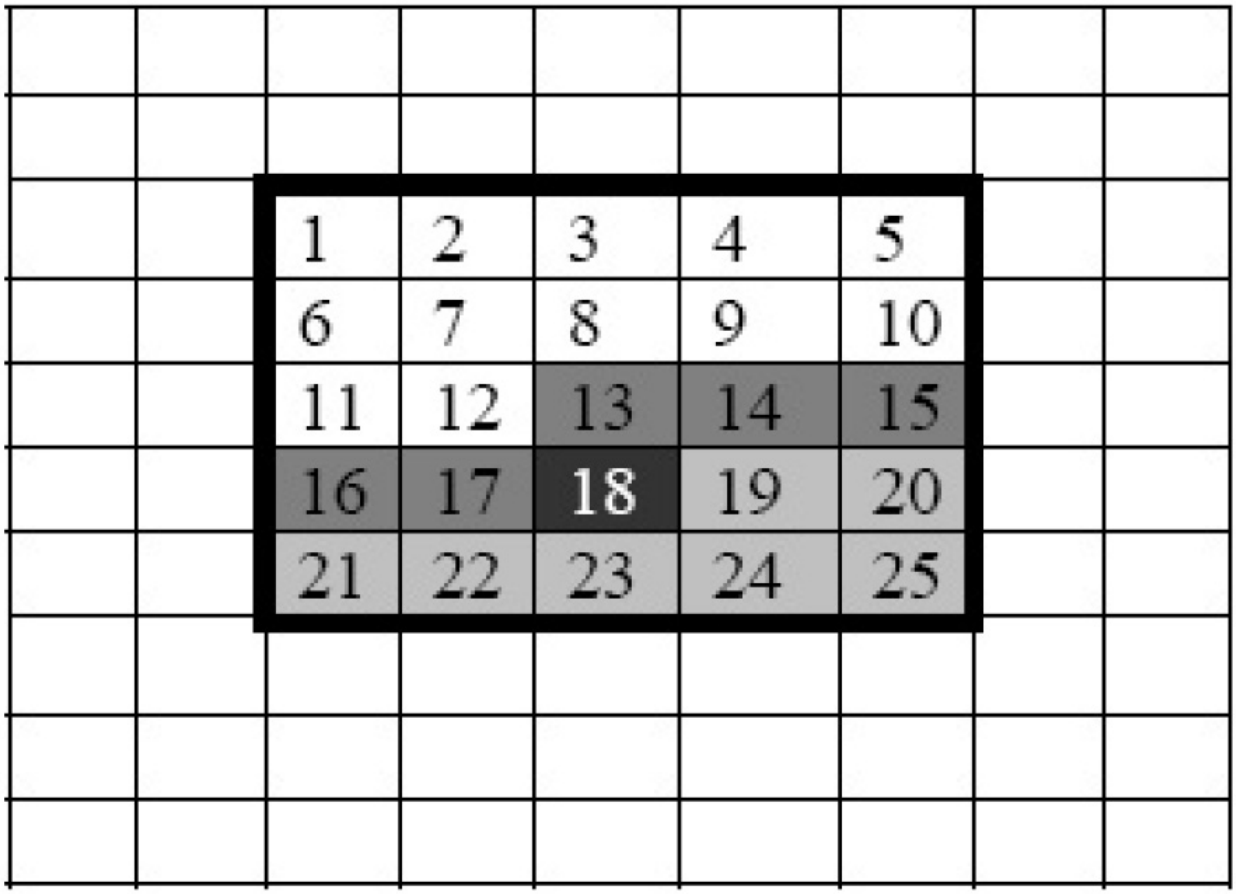
Siste node på siste rad, node $n - m + 1$ i blokken, beregnes spesifikt ved å anvende (35) på akkurat den noden. Vi generaliserer selvsagt slik at beregningen er uavhengig av blokkstørrelsen. Grunnen til at denne beregnes spesifikt og ikke inkluderes i algoritmen for de nodene som ligger fra og med $n-m+2$ til og med $n-1$ er at denne har flere naboer som er fikserte, altså de tilhører B (se figur 1).

Nodene på nest siste rad behandles også spesifikt, det vil si at de algoritmene vi benytter kun gjelder for denne raden i blokken. Dette har sammenheng med at indeksverdiene fra siste rad, som vi benytter på nest siste rad, blir annerledes enn for nodene i resten av feltet. Grunnen til dette er at nodene på siste rad ikke blir avhengige av nodene på samme rad som har lavere indeks enn naboen til vest for den noden vi står i. Dette gjør det vanskelig å lage en generell algoritme for nest siste rad som også gjelder for resten av blokken, ettersom de indeksene vi genererer når vi skal hente ut allerede beregnede sannsynligheter for å anvende i (35) genereres forskjellig fra de indeksene vi genererer når vi ikke skal hente ut sannsynligheter fra siste rad. På denne raden begynner vi med å beregne $P(n - m|x_{<n-m}, x_B)$. Denne beregner vi ved å benytte (35). Vi har igjen gjort beregningen av sannsynligheten uavhengig av blokkstørrelsen. Grunnen til at denne noden beregnes spesifikt er igjen at den er en hjørnenode og har flere noder fra B i nabolaget enn de andre nodene på samme linje.

Nodene fra og med $n - m - 1$ til og med $n - m - m + 2$, altså de som ligger imellom ytterpunktene på raden, er av lik karakter og kan beregnes med en generell algoritme for denne raden. Som for tilsvarende noder på siste rad i blokken, laget vi en ytre løkke som gikk gjennom indeksverdien til nodene i feltet. Vi laget dessuten en indre løkke som passer på hvor mange noder som gjenstår å summere ut. Siste node på raden, $n - m - m + 1$ beregnes ved å anvende (35). I figur 13 ser vi hvilke noder nodene mellom hjørnenodene på nest siste rad er avhengige av. Dersom vi anser øverste rad i blokken som rad 1, så kan radene



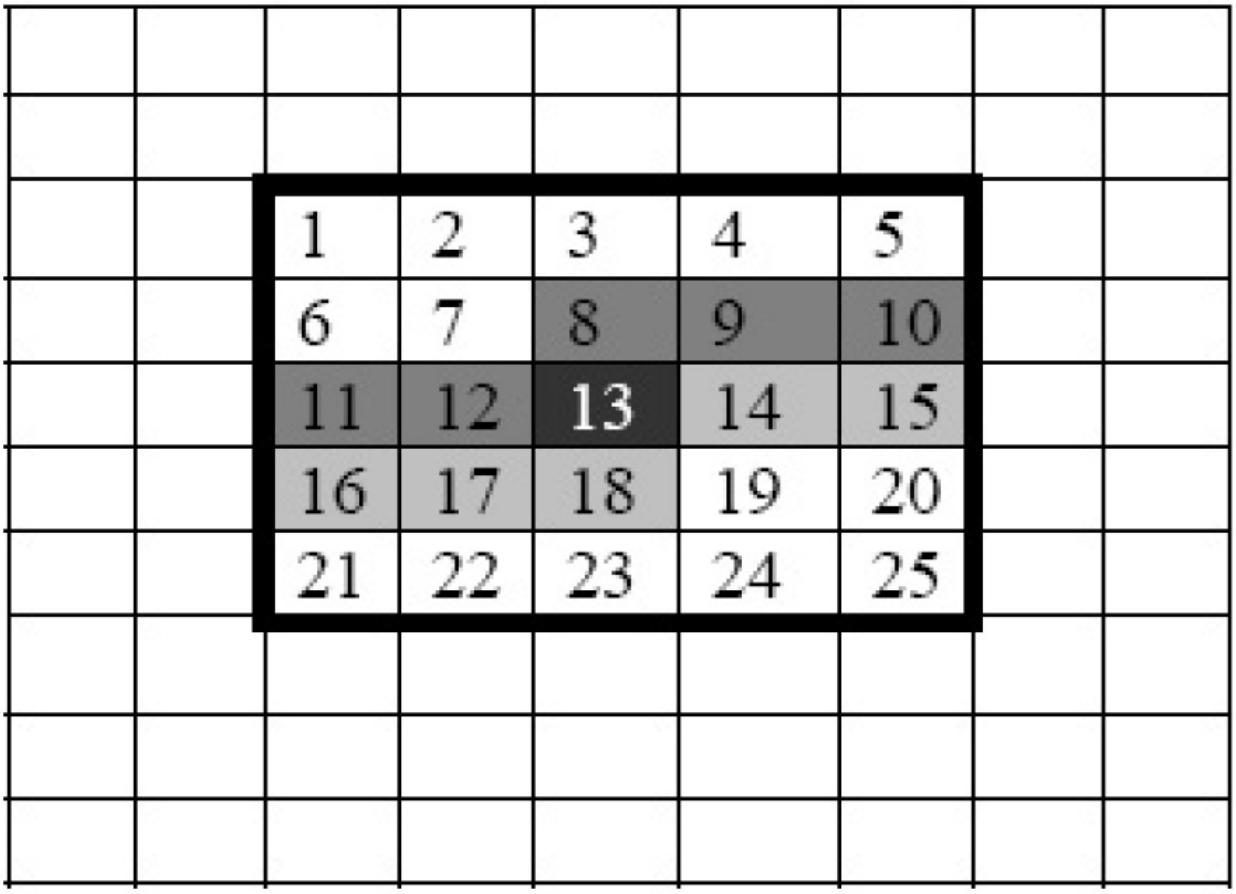
Figur 12: Figuren viser hvilke noder som er involvert når vi beregner sannsynligheten for nest siste node på siste rad. De lyse grå nodene er den noden vi skal summere ut, node 25. De nodene som er merket i mørkt grått er de som node 24 blir avhengige av når vi summerer ut node 25.



Figur 13: Figuren viser hvilke noder som er involvert når vi beregner sannsynligheten for nest siste node på siste rad. De nodene som er merket i mørkt grått er de nodene vi skal summere ut, node 19 til og med node 23. De nodene som er merket i lyst grått er de som node 18 blir avhengige av når vi summerer ut nodene 19 til og med 23. Node 18 er merket som mørkeste node i blokken

2 til og med rad $m-2$ estimeres i en enkelt algoritme som er uavhengig av blokkstørrelsen. Vi lager først en ytre løkke som holder orden på hvilken rad vi står i. Deretter har vi en ny løkke som holder orden på hvilken kolonne vi står i. For hver rad beregner vi først sannsynligheten til hjørnenoden på slutten av linjen, deretter sannsynlighetene som tilhører nodene mellom hjørnenodene på linjen, og til slutt hjørnenoden med lavest indeks. Da kan sannsynlighetene for nodene mellom hjørnene beregnes generelt såfremt vi holder styr på hvor vi står på radene. I figur 14 ser vi hvordan skjemaet for disse nodene blir med hensyn på hvilke noder node 13 er avhengige av.

På siste rad valgte vi å implementere beregningen av sannsynlighetene for hver node hver for seg. Dette hadde sammenheng med at det viste seg vanskelig å finne et generelt forhold i måten de ble beregnet på, samtidig som vi bare implementerte den rekursive blokkoppdateringen opp til en 5×5 blokk.



Figur 14: Figuren viser hvilke noder som er involvert når vi beregner sannsynligheten for nest siste node på siste rad. De nodene som er merket lyst grått de nodene vi skal summere ut, node 14 til og med node 18. De nodene som er merket i mørkt grått er de som node 13 blir avhengige av når vi summerer ut nodene 14 til og med 18. 13 selv er merket i den mørkeste grå fargen.