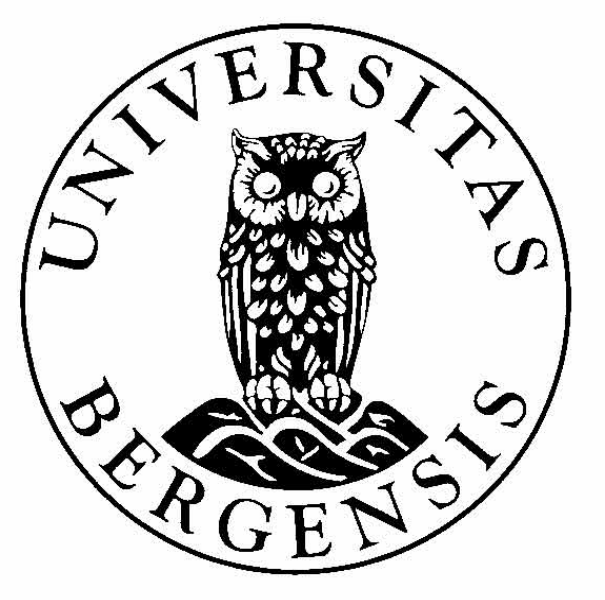
Selvstabiliserende algoritmer for frekvensallokering



Tom Martens Meyer Solem

Institutt for Informatikk

Universitetet i Bergen

20. juni 2004

Forord

Dette er en hovedfagsoppgave til graden Candidatus Scientarum ved Institutt for Informatikk, Universitetet i Bergen. Arbeidet med oppgaven er utført delvis ved Institutt for Informatikk og i Oslo i perioden januar 2003 til juni 2004.

Jeg vil med dette takke mine veiledere Fredrik Manne og Kjell Jørgen Hole. Jeg ønsker å takke dem for konstruktiv kritikk, givende diskusjoner, kommentarer og gode tilbakemeldinger.

Jeg vil takke mine nærmeste for god hjelp til å lese korrektur og moralsk støtte.

Tom Martens Meyer Solem

Bergen, 20. juni 2004

Sammendrag

I denne oppgaven blir det sett på selvstabiliserende algoritmer for tildeling av frekvenser i trådløse *ad hoc* nettverk. Det er laget en generisk simulator for å simulere selvstabiliserende algoritmer i distribuerte systemer.

Eksisterende selvstabiliserende algoritmer blir tilpasset problemstillingen der antall frekvenser er begrenset. Denne begrensningen gjør at ikke alle enhetene kan kommunisere samtidig i nettverket.

En ny selvstabiliserende algoritme for frekvensallokering presenteres. Den nye selvstabiliserende algoritmen blir målt opp mot eksisterende algoritmer (både sekvensielle fargeleggingsalgoritmer og selvstabiliserende). Det blir påvist gjennom simuleringer at den nye selvstabiliserende algoritmen er like bra eller bedre enn de eksisterende algoritmene.

Innhold

Forord i

Sammendrag ii

1 Introduksjon 1

2 Datastrukturer og ad hoc nettverk 6

2.1 Definisjon av en graf 6

2.2 Ad hoc nettverk 8

2.3 Frekvensplanlegging 10

2.4 Graffargelegging 11

2.4.1 Kompleksitet til sekvensielle algoritmer 11

2.4.2 Sekvensielle fargeleggingsalgoritmer 13

2.5 Selvstabiliserende algoritmer 16

3 Simulatoren 19

3.1 Funksjonaliteten til simulatoren 19

3.2 Implementasjon av simulatoren 20

3.3 Generering av datasett: Grafgeneratoren 22

4 Selvstabiliserende algoritmer 25

4.1 Rask fargelegging (RF) 25

4.2 Laveste ledige farge (LLF) 27

4.3 Maksimal, uavhengig mengde (MUM) 29

5 Nye selvstabiliserende algoritmer **32**

5.1 Modifikasjon av rask fargelegging (MRF) 32

5.2 $PekerFlagg$–algoritmen (PF) 35

5.2.1 Motivasjon for en bedre algoritme 36

5.2.2 Begrensninger som en må ta hensyn til 36

5.2.3 Prinsippene bak algoritmen 37

5.2.4 Algoritmen 41

5.2.5 Bevis for stabilitet og kompleksitet 44

6 Simulering av frekvenstildeling **50**

6.1 Datasettet som simuleringene kjøres på 50

6.2 Sammenligning av initielle tildelinger 52

6.2.1 Sammenligning av initielle verdier til MRF–algoritmen 53

6.2.2 Diskusjon av initielle tildelinger til MRF–algoritmen 56

6.2.3 Sammenligning av initielle verdier til TPF–algoritmen 58

6.2.4 Diskusjon av initielle fordelinger til TPF–algoritmen 62

6.3 Sekvensiell fargelegging 63

6.3.1 Sammenligning av SDO og Første ledige 64

6.3.2 Sammenligning av TPF og $LLPF$ 65

6.3.3 Sammenligning av SDO og $LLPF$ 66

6.3.4 Diskusjon 67

6.4 Antall ufargede noder 68

6.4.1 Sammenligning av $LLPF$–algoritmen med MRF og $MLLF$ . . . 68

6.4.2 Diskusjon 69

7 Oppsummering **71**

7.1 Konklusjon 71

7.2 Videre arbeid 72

A Implementasjon av nye regler til simulatoren **73**

B Data fra simuleringene **74**

B.1 Antall noder ufarget ved de ulike algoritmene 74

B.2 Tabeller til sammenligning av initielle verdier MRF, kap 6.2.1, side 53 . . 77

B.3 Tabeller til sammenligning av initielle verdier TPF, kap. 6.2.3, side 58 . . 78

B.4 Simuleringer med TPF–algoritmen. 80

B.5 Simuleringer med $LLPF$–algoritmen. 90

Figurer

2.1 Eksempel på en graf med 3 noder og 2 kanter. 7
2.2 Enheter med sender/mottaker i et rom. 8
2.3 Mulighet for kommunikasjon mellom enhetene representeres ved kanter. . . 8
2.4 Grafen som representerer et *ad hoc* nettverk. 9
2.5 Asymptotisk kjøretid. 12

3.1 Logisk bilde av simulatoren. 21
3.2 To noder kan kommunisere med hverandre. 23
3.3 Kommunikasjonsmuligheter mellom tre noder og tilhørende kanter. 24

4.1 Ved start har nodene samme farge. 26
4.2 Node v_1 har valgt ny farge. 26
4.3 Node v_2 har valgt ny farge. 27
4.4 Node v_4 har valgt ny farge. 27
4.5 Ved start har nodene samme farge. 28
4.6 Node v_1 har valgt ny farge. 28
4.7 Node v_3 har valgt laveste ledige farge. 28
4.8 Node v_2 har valgt ny farge. 28
4.9 Node v_4 har valgt den laveste ledige fargen. 29
4.10 Ved start har noen noder $s(i) = sann$. 30
4.11 Node H er gått ut av mengden τ. 30
4.12 Node I er blitt med i τ. 31
4.13 Node C er blitt med i τ. 31
4.14 Node D har gått ut av mengden τ. 31

5.1 Alle noder har samme farge ved start. 33
5.2 Node v_1 har valgt farge. 33
5.3 Node v_2 har valgt farge. 34
5.4 Node v_3 har valgt farge -1 (ingen farge). 34
5.5 Node v_4 har valgt farge 1. 34
5.6 Utsnitt av en graf der MRF–algoritmen er brukt. 36
5.7 Den initielle tildelingen av farger til nodene. 38
5.8 Node v_3 har brukt R_1 og v_5 har brukt R_1 og R_2. 38
5.9 Node v_4 har ikke flagget oppe, mens v_5 har flagget oppe. 39
5.10 Node v_4 har tatt opp flagget, mens v_5 har tatt det ned. 39
5.11 Nodene v_3 og v_7 peker på unike naboer. 40
5.12 Node v_6 har byttet farge og v_3 fått en farge og tatt vekk pekeren. 40
5.13 Node v_7 flytter pekeren til node v_8. 42
5.14 Ingen noder er privilegert lenger og systemet er stabilt. 42

5.15 Utsnitt av en graf der noen noder peker på andre. 48

6.1 Plott av nodene i grafen som simuleringene kjøres på. 51
6.2 Uniform fargefordeling. 53
6.3 Uniform fargefordeling inkludert -1 (ingen farge). 54
6.4 Alle nodene har den samme fargen. 55
6.5 Alle nodene har -1 fargen ved start. 56
6.6 Uniform fargefordeling. 58
6.7 Uniform fargefordeling inkludert -1 (ingen farge). 59
6.8 Alle nodene har den samme fargen. 60
6.9 Alle nodene har -1 fargen ved start. 61
6.10 Sammenligning av SDO og Første ledige. 64
6.11 Sammenligning av TPF og $LLPF$. 66
6.12 SDO sammenlignet med $LLPF$. 67
6.13 Sammenligning av $LLPF$ og $MLLF$ algoritmene. 68

Tabeller

6.1 Data til grafene generert av grafgeneratoren. 51
6.2 Oversikt over algoritmer som sammenlignes 52
6.3 Uniform fargefordeling. 53
6.4 Uniform fargefordeling inkludert -1 (ingen farge). 54
6.5 Alle nodene har den samme fargen. 55
6.6 Alle nodene har -1 fargen. 56
6.7 Uniform fargefordeling. 59
6.8 Uniform fargefordeling inkludert -1 (ingen farge). 60
6.9 Alle nodene har den samme fargen. 61
6.10 Alle nodene har -1 fargen. 62
6.11 Sammenligning av lavest ledige peker flagg og de to andre. 69

B.1 Antall noder ufarget ved MRF–algoritmen og uniform fordeling inkl. -1. . 74
B.2 Antall noder ufarget ved TPF–algoritmen og uniform fordeling inkl. -1. . . 74
B.3 Antall noder ufarget ved $MLLF$–algoritmen og uniform fordeling inkl. -1. 75
B.4 Antall noder ufarget ved $LLPF$–algoritmen og uniform fordeling inkl. -1. . 75
B.5 Antall noder ufarget ved sekvensiell SDO–algoritmen. 75
B.6 Antall noder er ufarget ved sekvensiell Første ledige farge algoritmen. . . . 76
B.7 Fordeling av farger til MRF–algoritmen ved uniform initiell fordeling. . . 77
B.8 Fordeling av farger til MRF–algoritmen ved den samme fargen til alle nodene initielt. 77
B.9 Fordeling av farger til MRF–algoritmen ved uniform fordeling inkl. -1 initielt. 77
B.10 Fordeling av farger til MRF–algoritmen ved alle noder ufarget initielt. . . 78
B.11 Fordeling av farger til TPF–algoritmen ved uniform fordeling initielt. . . . 78
B.12 Fordeling av farger til TPF–algoritmen ved den samme fargen til alle nodene initielt. 78
B.13 Fordeling av farger til TPF–algoritmen ved uniform fordeling inkl. -1 initielt. 79
B.14 Fordeling av farger til TPF–algoritmen ved alle noder ufarget initielt. . . . 79
B.15 Tabell der TPF velger tilfeldig, ledig farge 81
B.16 Tabell der $LLPF$ velger laveste ledige farge 90

Kapittel 1

Introduksjon

Trådløs kommunikasjon blir stadig mer vanlig. Guliemo Marconi tok patent på å sende telegram ved bruk av radiobølger i 1896. I 1905 ble det første telegrammet sendt over Atlanterhavet med radiobølger [17]. Det ble etterhvert innført trådløs kommunikasjon på skip som seilte på verdenshavene. Under 2. Verdenskrig ble radiokommunikasjon en viktig faktor. Radio ble brukt til å sende hemmelige meldinger til flåter og fly. Alan Mathison Turing, kjent for den teoretiske modellen for datamaskiner, Turing maskinen, arbeidet under 2. Verdenskrig med å ”knekke” hemmelige meldinger [16].

Dr. Martin Cooper utførte den første trådløse telefonsamtalen 3. april 1973 utenfor Hilton Hotell i New York [17]. Dette var starten på en ny tid for telekommunikasjon. I begynnelsen var det få som hadde mobiltelefon, men frem til 2004 er det blitt solgt mer enn 1 milliard mobiltelefoner verden over [17]. Den 3. desember 1992 sendte Neil Papworth en SMS (Short Message Service) til sine kolleger i Vodaphone. Den lød: *”MERRY CHRISTMAS”*. Dette var den første SMS som ble sendt [20].

I 1997 ble de første produktene basert på *IEEE* 802.11b standarden lansert [7]. Dette var begynnelsen på kommersielle trådløse nettverk for datakommunikasjon. De fleste bærebare pc’er som kommer på markedet i dag, har minst én sender og én mottaker for trådløs kommunikasjon.

For at enheter skal kunne kommunisere med hverandre trådløst, må de sende og motta radiobølger på de samme frekvensene. Det finnes i dag mange forskjellige trådløse nettverk som bruker radiobølger til kommunikasjon. Mobiltelefonnettet har en infrastruktur med

basestasjoner som er koblet sammen med det trådbaserte telefonnettet. Basestasjonene kan kommunisere med mobiltelefonene til en gitt avstand, men dersom avstanden blir for stor, blir signalene for svake til kommunikasjon. Området hvor det er dekning kalles dekningsområdet til basestasjonen.

Mobiltelefoner som kommuniserer med samme basestasjon, må bruke forskjellige frekvenser eller dele en for ikke å forstyrre hverandre. For at et større område skal ha dekning til mobilkommunikasjon, og en skal unngå å miste kontakten med basestasjonene når en beveger seg mellom dem, må dekningsområdet til basestasjonene overlappe. Et dekningsområde kalles gjerne en ”celle”. Cellene som overlapper med hverandre, må kommunisere i forskjellige frekvensområder. Hvis alle basestasjonene skulle hatt unike frekvensområder, hadde det ikke vært mulig å plassere ut mange basestasjoner. Siden antall frekvenser er begrenset, er det gjenbruk av frekvensområder. De basestasjonene som ikke har et overlappende dekningsområde, kan bruke det samme frekvensområdet til å kommunisere med mobiltelefonene.

Det er basestasjonene som står for tildeling av frekvenser i mobilnettverket. Disse har blitt tildelt et frekvensområde med et gitt antall frekvenser som de kan bruke til kommunikasjon med mobiltelefoner. Dette er en sentralisert og dynamisk måte å tildele frekvensene på. Mobiltelefoner har mulighet til å kommunisere på alle de ulike frekvensene i frekvensområdet til basestasjonene. Et mer statisk nettverk er TV-nettverket. Alle basestasjonene sender på de samme frekvensene hele tiden. I mobilnettet er det mulig å endre på frekvensene uten at brukerne merker det. Dette kan ikke gjøres med TV-nettet, for da må alle stille inn TV’en på nytt.

En annen type nettverk er de uten fast infrastruktur. De har ikke faste master eller basestasjoner. Det kan være forskjellige grunner til at en ikke ønsker eller har en infrastruktur. Dersom nettverket bare skal brukes i en begrenset periode, kan det bli for dyrt å sette ut basestasjoner. I f.eks. jordskjelvområder trenger kanskje hjelpemannskaper å sette opp et nettverk for å koordinere hjelpen. Det er da gjerne hverken tid eller ressurser til å sette opp basestasjoner. Et annet eksempel kan være militære enheter i en stridssituasjon der de ikke har tilgang til infrastruktur.

Infrastrukturløse nettverk er bygd opp av selvstendige enheter som kommuniserer med hverandre direkte. Dette kan f.eks. være bærbare pc’er som kommuniserer direkte med

hverandre. Når disse pc'ene skal kommunisere med hverandre trådløst, må de danne et trådløst nettverk. Et distribuert system består av selvstendige enheter som blir koblet sammen i et selvorganisert nettverk (f.eks. selvstendige pc'er). Det er ingen sentral kontroll over når kommunikasjon skjer eller hvilke enheter som kommuniserer med hverandre. Tenk deg et rom med mange selvstendige enheter, f.eks. en stor messe med mange personer med bærbare pc'er e.l. Hvis enhetene skal kunne kommunisere med hverandre uten å forstyrre andre enheter i nærheten (de som er i det samme dekningsområdet), må de kommunisere på forskjellige frekvenser.

Hvordan skal de enhetene som er i nærheten av hverandre, bli enige om hvilke frekvenser de skal bruke? Det er ingen sentral enhet som sier at enhet *A* skal bruke frekvens "en", og enhet *B* skal bruke frekvens "to". Ved å bruke selvstabiliserende algoritmer, kan enhetene velge frekvenser selv. Selvstabiliserende algoritmer gir hver enhet en mengde med *regler*. Ved at alle enhetene følger disse reglene, vil de få en frekvens å kommunisere på. Det er to forutsetninger for at alle skal få en frekvens. Den ene er at det er nok frekvenser å velge mellom, og den andre er at reglene er implementert riktig.

Selvstabiliserende algoritmer vil søke å oppnå et stabilt system. Et system er stabilt dersom alle enhetene har en frekvens å kommunisere på og ikke forstyrrer andre nærliggende enheter. (Et system kan være stabilt selv om ikke alle enhetene har frekvenser å kommunisere på. Enhetene kan gi opp å finne en ledig frekvens).

Hvor mange av enhetene kan kommunisere med hverandre med et gitt antall frekvenser? Dette er et av de spørsmålene denne oppgaven skal prøve å besvare. Et av de andre spørsmålene vi ønsker å belyse, er om tildelingen av frekvenser til enhetene ved start har noe å si for utfallet når systemet har blitt stabilt. Det siste spørsmålet er: Vil en "smartere" måte å velge frekvenser på ha en innvirkning på tildelingen til alle enhetene? Kan en enhet f.eks. sende ut en forespørsel til de andre nærliggende enhetene om de kan endre frekvens? Vil det da bli flere enheter som kan kommunisere sammen?

For å kunne si noe om hvor mange enheter som kan kommunisere med hverandre, må det på en eller annen måte prøves ut. En kan prøve dette ut i full skala. Det kan settes opp et rom med mange enheter, og så kan utprøvingen av frekvensvalg starte, eller det kan lages et dataprogram som simulerer dette nettverket. Ved å lage et program som simulerer

infrastrukturløse nettverk, dukker det opp en del problem, f.eks. hvordan de forskjellige enhetene skal simuleres.

I denne oppgaven vil det bli demonstrert et simulatorprogram som bruker én prosessor på en datamaskin. For å simulere flere selvstendige enheter, må det holdes en oversikt over hvilke enheter som til enhver tid gjør hva. Med objektorientert programmering blir dette lett. Simulatoren har en ”demon” som kontrollerer enhetene og velger ut den enheten som skal få tid til å endre frekvens. På denne simulatoren vil det bli utprøvd forskjellige frekvenstildelinger. Disse forskjellige tildelingene vil bli sammenlignet med hverandre.
Denne oppgaven har 3 hovedspørsmål:

I. Hvor mange enheter kan kommunisere med hverandre gitt at det er k frekvenser?

II. Har frekvensen som enhetene har initielt, noen betydning for hvordan resultatet blir til slutt?

III. Finnes det noen ”smarte” løsninger som gjør tildelingen av frekvenser bedre? Bør enheter sende ut en forespørsel om andre nærliggende enheter har mulighet til å endre frekvensen(e) de sender på?

Oppgaven består av 7 kapitler inkludert denne innledningen. De første 2 kapitlene er en introduksjon til problemstillingene og definerer de grunnleggende begrepene.

Kapittel 2 begynner med å definere grafer. Det går videre med å forklare hva et *ad hoc* nettverk er og hvordan grafer kan representere *ad hoc* nettverk. Sammenhengen mellom frekvenstildeling og graffargelegging forklares før noen sekvensielle fargeleggingsalgoritmer vises. Kapitlet avsluttes med en introduksjon til selvstabiliserende algoritmer.

For å simulere tildelingen av frekvenser i trådløse nettverk, trenger vi en simulator. I kapittel 3 blir simulatoren som er laget, forklart. Siste del av kapittel 3 forklarer grafgeneratoren. Den skal lage tilfeldige grafer som representerer *ad hoc* nettverk.

I kapittel 4 forklares tre eksisterende selvstabiliserende algoritmer. Disse tre algoritmene ble brukt til å teste simulatoren. Det er to fargeleggingsalgoritmer, rask fargelegging og laveste ledige farge. Den siste finner en maksimal, uavhengig mengde.

I kapittel 5 blir rask fargelegging tilpasset problemstillingene til denne oppgaven. Dette blir gjort for å ha et utgangspunkt å sammenligne de ”smarte” algoritmene med. De blir

også presentert i kapittel 5.

I kapittel 6 blir de ulike algoritmene brukt i simulatoren. Det er generert datasett fra grafgeneratoren. Et av spørsmålene som vi ønsker se på, er om verdien nodene har ved start påvirker utfallet av simuleringene. Det vil i dette kapitlet bli presentert 4 forskjellige tildelinger ved start, og kjørt simuleringer med de. Det kommer en sammenligning av de 4 ulike tildelingene.

Kapittel 7 er en generell diskusjon og en oversikt over resultatene. Forslag til problemstillinger det kan arbeides videre med blir nevnt.

Kapittel 2

Datastrukturer og ad hoc nettverk

Dette kapitlet tar for seg de grunnleggende begrepene, som vil bli brukt, i denne oppgaven. Grafer og *ad hoc* nettverk defineres for å gi begrep om hvordan vi forholder oss til den virkelige verden. Kapitlet viser videre hvordan en kan se på frekvensplanlegging som et graffargeleggingsproblem.

Det er ulike måter å løse graffargeleggingsproblemet på. Noen algoritmer presenteres for å vise dette. Til slutt kommer den formelle definisjonen på selvstabiliserende algoritmer. Det er denne måten vi ønsker å fargelegge grafer på.

2.1 Definisjon av en graf

En graf er en abstrakt representasjon av relasjoner. En graf G er ett par (V, E) av mengder. Mengden V kalles **noder** og kan være en representasjon av fysiske enheter, f.eks. pda'er[1]. Mengden E er par av noder (i, j). Det er da en relasjon mellom nodene i og j. Relasjonene mellom nodene kalles **kanter**. I en urettet graf har kantene ingen retning, og E består av uordnede par. Hvis det er en kant mellom node i og node j, er de **naboer**, dvs. at paret bestående av node i og node j er i mengden E.

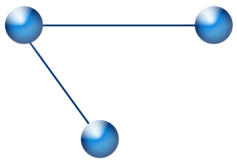
En node kan ha fra null til $|V| - 1$ naboer. Antall naboer en node har, kalles **gradtallet**. Dersom en node $v \in V$ har to naboer, er gradtallet til noden to. Dette betegnes *deg(v)* = 2. Det største gradtallet som forekommer i en graf betegnes Δ. Dvs. Δ er gradtallet til

[1] pda: personal digital assistant. En digital filofax.

den noden i V som har flest naboer av alle nodene i V. Minste gradtall betegnes δ. Det er gradtallet til den noden som har færrest naboer av alle nodene i V. Det gjennomsnittlige gradtallet betegnes $\bar{\delta}$. Dette er $\frac{2m}{n}$ der $|E| = m$ og $|V| = n$, dvs. summen av alle gradtall delt på antall noder. En graf kan være en representasjon av f.eks.:

- En konstruksjon: boreplattform, skipsskrog.
- Et kart: veier og gatekryss.
- Muligheter for kommunikasjon mellom mobile enheter.

Det er det siste punktet som vil bli brukt i denne oppgaven. Alle de n nodene i grafen vil representere fysiske enheter. En enhet kan være en pda som er utstyrt med en sender og mottaker for at den skal kunne kommunisere med andre pda'er. Dette kan f.eks. være et trådløst kort med *IEEE* 802.11b standarden, også kalt *Wi-Fi* [7]. I figur 2.1 ser vi et eksempel på en graf med tre noder og to kanter.



Figur 2.1: Eksempel på en graf med 3 noder og 2 kanter.

En **sti** er en liste av noder, $v_1, v_2, ..., v_l$. Den første noden er starten på stien, og den l-te noden er slutten. Node v_1 har node v_2 som nabo, og node v_2 har i tillegg node v_3 som nabo. Dermed kan en starte i node v_1 og følge listen av noder til man ender opp i node v_l.

En graf er **sammenhengende** hvis og bare hvis det er mulig å følge kantene fra en vilkårlig node til alle andre noder i grafen. Med andre ord eksisterer det en sti fra en vilkårlig node til alle andre noder. Grafen G er **planar** hvis og bare hvis det er mulig å tegne den i planet uten at noen kanter krysser hverandre. I en **komplett** graf er alle nodene naboer med hverandre. Den komplette grafen K_5, bestående av 5 noder, er ikke planar. En **klikk** er en undermengde av en graf som er komplett.

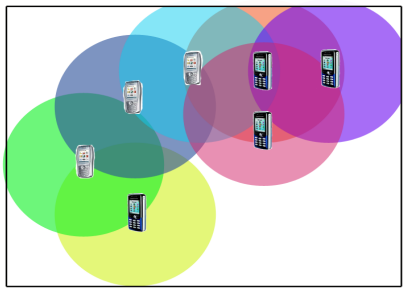
Vi har at en **uavhengig** mengde I er en undermengde av V der to vilkårlige noder i I ikke er naboer i G. Mengden I er en **maksimal**, uavhengig mengde hvis ingen uavhengige mengder inneholder I som en undermengde.

Den **induserte** delgrafen $G[U]$ av G er en graf bestående av mengden med noder $U \subset V$ og alle kantene mellom nodene i U. For mer informasjon om grafer se [21].

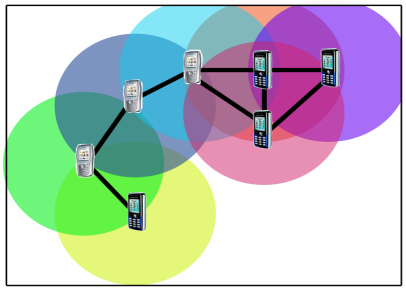
2.2 Ad hoc nettverk

Et *ad hoc* nettverk består av enheter (f.eks. laptop'er, pda'er, mobiltelefoner) som sammen danner et tilfeldig selvorganisert nettverk. Det har ingen infrastruktur slik som basestasjoner eller andre faste holdepunkt. Derfor er det ingen sentral kontroll med hvem som kobler seg på nettet eller hvem som kommuniserer med hvem.

Tenk på børsen i New York. Der er det mange mennesker som selger og kjøper aksjer. Hvis alle har en pda eller pc, vil de kunne danne et *ad hoc* nettverk med de andre personene på børsen. Personene vil bevege seg rundt, og dette gjør nettverket til et dynamisk miljø der endringer skjer hele tiden.



Figur 2.2: Enheter med sender/mottaker i et rom.

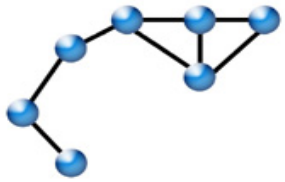


Figur 2.3: Mulighet for kommunikasjon mellom enhetene representeres ved kanter.

Figur 2.2 viser et rom med 7 enheter. Disse enhetene har en sender/mottaker som sender med en gitt effekt målt i watt W. Sendeeffekten er representert med de fargede sirklene rundt enhetene. Disse har en radius på d. Hvis avstanden mellom to enheter er mindre enn d, kan de kommunisere med hverandre. De kan også forstyrre hverandre hvis de sender på samme frekvens samtidig. Vi bruker en graf $G = (V, E)$ til å representere *ad hoc* nettverket. Kantene E viser mulighetene for kommunikasjon mellom nodene V. I figur 2.3 er det

tegnet kanter mellom de enhetene som har avstand mindre enn d mellom seg. Sirklene rundt nodene er det som ble kalt dekningsområdet i kapittel 1.

I figur 2.4 er det en tegning av grafen som representerer *ad hoc* nettverket. Dette er en



Figur 2.4: Grafen som representerer et *ad hoc* nettverk.

ideell representasjon. Den tar ikke høyde for mange av de fysiske hindringene som opptrer i den virkelige verden. Søyler, vegger eller andre fysiske hindringer gjør at området som enheter sender til ikke alltid er en perfekt sirkel. Samtidig kan vegger og vinduer reflektere signaler eller bøye dem av. For mer informasjon om signalforplantning, dekningsområde og andre fysiske problemområder se [17].

I nettverk med infrastruktur blir kommunikasjonen mellom nodene sendt via basestasjonene. I *ad hoc* nettverk kan naboer kommunisere direkte med hverandre. Men hvordan skal en node kommunisere med andre noder som ikke er naboer? Dette er et problem som oppstår i *ad hoc* nettverk. Under kommer noen flere problemer i *ad hoc* nettverk:

- Garantert båndbredde (Quality of Service). Skal noen noder ha en gitt båndbredde til rådighet til enhver tid? Dvs. skal noen enheter kunne se på video-strømmer samtidig, mens de andre ikke kan kommunisere, eller skal alle enhetene få kommunisere litt hver?

- Strømforbruk til enhetene. Hvis routing'en skal være rask, må de forskjellige enhetene sende informasjon om hvem de har som naboer. Det er to ting som gjør dette vanskelig. Kommunikasjon krever strøm. De fleste enhetene bruker batterier, og strøm er derfor en begrenset ressurs. Ekstra kommunikasjon bruker dessuten av den totale tiden systemet kan kommunisere.

2.3 Frekvensplanlegging

I innledningen til denne oppgaven ble det skissert flere måter å tildele frekvenser på. Her kommer en liten oversikt over frekvenstildeling i trådløse nettverk.

I figur 2.2 er det overlapp mellom to eller flere av sirklene til enhetene som skal kommunisere. Dersom disse enhetene skal sende samtidig uten å forstyrre hverandre, må de sende på forskjellige frekvenser. Frekvenser er en begrenset ressurs. De er derfor delt opp i forskjellige bruksområder. Noen områder er satt av til militært bruk og mobiltelefoner, mens andre områder er ”frie”. Alle som vil kan sende i disse områdene hvis de overholder myndighetenes krav. *Wi-Fi* [7] bruker et slikt åpent område. Dette området kalles ISM (Industrial Scientific Medical) båndet og kan brukes av alle dersom en sender på lav styrke. I Norge er det Post- og teletilsynet som bestemmer hvem som kan bruke de forskjellige frekvensområdene (et av ISM–båndene i Norge er fra 5,725 GHZ til 5,925 GHZ. Alle kan sende på under 25 mW i dette båndet). For mer informasjon se [15].

Det er mange teknologier som bruker ISM–båndet, blant annet *Wi-Fi*, Bluetooth (blåtann), trådløse telefoner og mikrobølgeovner. Signaler fra andre teknologier vil av *Wi-Fi* bli oppfattet som støy og dermed forstyrre kommunikasjonen.

Det finnes to former for interferens. Den ene er at to enheter kommuniserer på samme kanal samtidig, og den andre er at de kommuniserer på nærliggende kanaler samtidig. I *Wi-Fi* kan ikke to enheter som er i samme dekningsområde kommunisere på to nærliggende kanaler. Dette er fordi 802.11b ”smører” signalene utover flere kanaler (et større frekvensspekter enn det en kanal er begrenset til) [7].

I den virkelige verden er frekvensplanlegging mer komplisert enn i denne oppgaven. Der er signalforplantningen avhengig av flere faktorer; terreng, frekvensen som brukes, farten til den mobile enheten og interferens fra andre enheter. I et åpent område er signaltapet $\frac{W}{d^2}$, der W er styrken det sendes med og d er avstanden, mens i et byområde er signaltapet mer komplekst å regne ut. Det er vanligvis et større tap enn i åpne områder [17].

2.4 Graffargelegging

Graffargelegging er å tildele en mengde $\{0, 1, ..., k\}$ farger til grafens noder slik at ingen naboer har samme farge. Det minste antall farger som trengs til å fargelegge en graf, kalles det **kromatiske** tallet til grafen og betegnes $\mathcal{X}(G)$.

Noder som er naboer kan forstyrre hverandre hvis de sender på samme frekvens. Ved å velge forskjellige frekvenser/farger til naboene, kan alle i nabolaget kommunisere sammen uten å forstyrre hverandre. Her ser en den klare sammenhengen mellom frekvensplanlegging og graffargelegging. Det er akkurat det samme bare at graffargelegging er en teoretisk tilnærming til det praktiske problemet med frekvenstildeling.

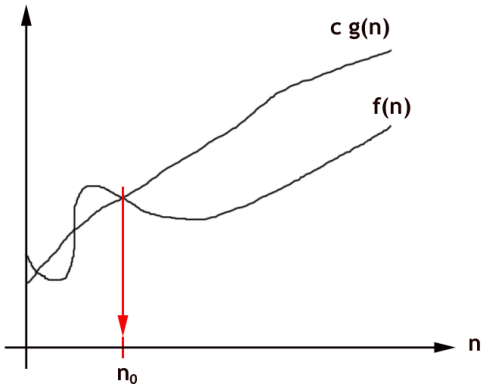
Fargelegging av grafer henger også sammen med å finne den største klikken i en graf. En klikk som har k noder, må ha k farger for å kunne fargelegges korrekt.

Det eksisterer flere ulike algoritmer for fargeleggingsproblemet. Senere i dette kapitlet vil det bli presentert noen fargeleggingsalgoritmer.

2.4.1 Kompleksitet til sekvensielle algoritmer

Graffargelegging er et vanskelig problem. For å gi en forklaring på hvor vanskelig det er, må noen begrep defineres. En algoritme er en formell definisjon av instruksjoner som løser et ønsket problem. Tid og plass (minne) er kostbare ressurser. En ønsker alltid å få svaret på problemet så raskt som mulig.

For å si hvor bra en algoritme er, trenger vi en måte å sammenligne de forskjellige algoritmene på. Tiden en algoritme bruker på å løse et problem, avhenger av størrelsen på det problemet algoritmen får inn (inndata). Vi sier gjerne at kjøretiden til en algoritme er en funksjon av inndata. Hvis inndata er n, er kjøretiden $f(n)$. Den kan f.eks. være $f(n) = 2n^2+3n$. Da utfører algoritmen $2n^2+3n$ operasjoner for å løse et problem av størrelse n. En asymptotisk øvre grense på tiden en algoritme bruker på et problem, betegnes stor $\mathcal{O}$. Ved å si $\mathcal{O}(g(n))$, betyr det at for enhver n denne algoritmen får inn, eksisterer det konstanter c og n_0 slik at funksjonen $f(n)$ alltid vokser saktere enn $cg(n)$. Når $f(n) = 2n^2 + 3n$, kan $g(n) = n^2$. Vi sier at algoritmen har $\mathcal{O}(g(n))$ kjøretid. $0 \leq f(n) \leq cg(n)$ ($f(n)$ ligger mellom 0 og funksjonen $g(n)$ multiplisert med en konstant c for alle $n \geq n_0$). Se figur 2.5.



Figur 2.5: Asymptotisk kjøretid.

$\mathcal{O}$-notasjon brukes til å gi en øvre grense på kjøretiden til en algoritme. Konstanter og lavere grads ledd blir ikke tatt med i $\mathcal{O}$-notasjon. Hvis kjøretiden er $f(n) = n^2+2n+5$, så er denne $\mathcal{O}(n^2)$. Dette er verste tilfelle. Algoritmen vil aldri bruke mer enn $\mathcal{O}(n^2)$ operasjoner.

Kjøretidene deler algoritmene inn i klasser. Klassen P er alle problem som en kan svare ja/nei på innenfor en asymptotisk polynomisk kjøretid. Polynomisk kjøretid er alle funksjoner av typen $f(n) = n^k$, der k er en konstant.

En annen klasse er NP. Dette er klassen av problem der en kan verifisere en løsning i polynomisk tid. Når inndata og utdata til et problem er gitt, kan en i polynomisk tid verifisere at utdata er en riktig løsning på problemet.

En tror at $P \neq NP$. Det er ingen som enda har klart å bevise det ene eller det andre. Dette er et av de viktige åpne spørsmålene i informatikk/matematikk.

En underklasse av NP er klassen bestående av NP–komplette problem. Hvis noen klarer å vise at ett NP–komplett problem er i klassen P, vil $P = NP$. Kan en vilkårlig graf G farges slik at ingen naboer har samme farge med k farger ($k \geq 3$)? Dette er et problem i klassen NP–komplett. Hvis grafen G er planar, er det nok med 4 farger for å få en korrekt fargelegging [21], men det er fortsatt NP–komplett å finne ut om den planare grafen kan farges med 3 farger.

Kjøretiden til en eksakt algoritme for graffargelegging har eksponensiell tid (k^n, der k og n er konstanter). Det er det samme som å prøve alle muligheter. Disse algoritmene

ligger i klassen EXP_{TIME}, dvs. klassen av problem som har eksponensiell kjøretid. Ved eksponensiell kjøretid øker tiden det tar å løse problemet eksponensielt med størrelsen på inndata.

Et eksempel på en eksponensiell algoritme kom Moon og Moser med i 1965. De viste en øvre grense på antall maksimale, uavhengige mengder en graf kan ha [14]. Denne øvre grensen er $3^n/3$, der n er antall noder i grafen. Det finnes eksakte algoritmer som finner en optimal fargelegging i tid $\mathcal{O}((1+3^{1/3})^n) = 2,4422^n$. Dette gjelder for en vilkårlig graf. Selv om det finnes eksakte algoritmer, er de ikke effektive. Det arbeides mye med å forbedre disse eksakte algoritmene. Tiden det tar å finne en optimal eksakt fargelegging, er nå $1,3289^n$ [14]. En kan finne mer informasjon om kompleksitet og eksponensielle algoritmer i [1, 18].

2.4.2 Sekvensielle fargeleggingsalgoritmer

En algoritme som ikke har noen garanti for hvor bra svaret er, kalles en heuristikk. Dette er i motsetning til en aproksimasjonsalgoritme. Den gir en tilnærming (approksimasjon) til en optimal løsning. I graffargelegging sier en gjerne at en α-approksimasjon er en tilnærming som ikke bruker mer enn $\alpha\mathcal{X}(G)$ farger.

Alle grafer kan farges med $\Delta+1$ farger. Den beste approksimasjonsalgoritmen har i dag en ratio (antall farger algoritmen bruker delt på optimalt antall farger) på $\mathcal{O}(n\frac{(log\ log\ n)^2}{(log\ n)^3})$ [10]. Det er også vist at en ikke kan få en bedre ratio enn $n^{1/7-\epsilon}$ for enhver $\epsilon > 0$ der $n = |V|$ [2]. Det finnes forskjellige fargeleggingsheuristikker. Disse heuristikkene kan deles inn i grupper ved å se på måten problemet løses på:

- **Grådige algoritmer.** Et grådig valg er at en node gjør det som er best for seg selv akkurat der og da. Det er ulike måter å velge hvilken node som skal gjøre det grådige valget. Noen algoritmer bruker nodeorden, mens andre bruker maksimale, uavhengige mengder. Grådige algoritmer er raske.

- **Søkealgoritmer.** Den samme algoritmen kjøres flere ganger der hver iterasjon bruker den foregående løsningen og forbedrer denne. Dette kan også kalles lokale forbedringer. Søkealgoritmer er langsommere enn grådige algoritmer, men de gir som oftest et bedre resultat.

- ⇨ **Parallelle algoritmer.** Ved å bruke multiple prosessorer samtidig, kan en løse problemet raskere enn ved å bruke en enkelt prosessor (dette forutsetter at det er ting som kan beregnes samtidig, og at disse beregningene er uavhengige av hverandre).

Grådige algoritmer Den enkleste fargeleggingsheuristikken er å traversere igjennom alle nodene og tildele disse den laveste ledige fargen. Algoritme 2.4.1 er en algoritme for simpel tildeling av farger i en graf. Mer informasjon om grådige algoritmer kan finnes her [5].

Algoritme 2.4.1 *Første ledige farge.*

```
Første ledige farge(inndata: en graf G)
1   n = |V|;
2   gjenta(for i = 1 til n)
3      gi node i den laveste tilgjengelige fargen;
```

Det er mulig å endre denne litt. I linje 3 tildeles den laveste tilgjengelige fargen. Dette kan endres til å være en tilfeldig farge blant de ledige. Tiden algoritmen bruker, er avhengig av gradtallet til hver node. Algoritmen må finne fargene til nodens naboer. Kjøretiden blir $\sum_{k=1}^{n} deg(v_k) = 2m$, i $\mathcal{O}$-notasjon: $\mathcal{O}(m)$, der m er antall kanter.

Algoritme 2.4.1 er en grådig algoritme. Den velger en node, og denne noden gjør sitt grådige valg. Når en node har gjort sitt grådige valg, endres den aldri siden i algoritmen. Algoritme 2.4.2 er også en grådig algoritme, men her er det ulike måter å velge nodene på.

Algoritme 2.4.2 *Grådig fargelegging.*

```
Grådig(inndata: en graf G)
1   U = V;
2   så lenge( U ≠ ∅ )
3     node i = hent node fra U iht. valgmåte;
4     gi i laveste lovlige farge;
5     U = U − {i};
```

Mengden U er de nodene som enda ikke har fått tildelt en farge. I linje 3 velges det ut en node. Her er eksempler på forskjellige måter å velge denne noden på.

- ⇨ Tilfeldig sekvens. Ved hver iterasjon blir en node valgt tilfeldig blant de resterende nodene i U. Denne måten å velge på vil gi forskjellige svar hver gang den kjøres.

- ⇨ Gradtallet til nodene kan også brukes som valgkriterium Det finnes ulike måter å prioritere nodene på vha. gradtallet. Det kan være største gradtall, minste gradtall eller tilfeldig gradtall.

  Tiden algoritme 2.4.2 bruker er avhengig av gradtallet til hver node. Ved bruk av største, minste eller tilfeldig gradtall vil denne algoritmen bruke $\mathcal{O}(m)$ tid.

- ⇨ Antall naboer som har fått farge. En metode som bruker dette kalles SDO (Saturation Degree Ordering). Denne bruker antall naboer som har farge, og hver farge telles en gang. Den noden med flest ulike farger blant naboene blir valgt til å fargelegges. Denne kan implementeres til å kjøre i tid $\mathcal{O}(n^2)$ [8].

Uavhengig mengde En annen måte å fargelegge grafer på er å bruke uavhengige mengder i grafen. Alle nodene som er i den samme uavhengige mengden, kan ha den samme fargen. Algoritmen 2.4.3 deler G opp i maksimale, uavhengige mengder og gir disse en farge hver.

Algoritme 2.4.3 *Pseudokode for maksimal, uavhengig mengde fargelegging.*

```
Uavhengig Mengde(inndata: en graf G)
1  U = V;
2  G' = G;
3  så lenge( G' != ∅ )
4    τ = maksimal, uavhengig mengde i G';
5    gi τ en ny farge;
6    U = U − τ;
7    G' = G[U]; /* den induserte delgrafen til U */
```

Ved neste iterasjon vil ikke nodene i τ være med i grafen G' [13]. Det er også ulike måter å finne uavhengige mengder på. Det kan gjøres grådig ved at en starter i en tilfeldig node og tar denne med i mengden τ. Algoritmen fortsetter med å traversere gjennom nodene og legger de i mengden τ når det er mulig. En annen måte er å søke etter den noden med lavest gradtall som har lov å være med i τ.

Søkealgoritmer Disse algoritmene starter gjerne med en enkel fargeleggingsalgoritme f.eks. 2.4.1. Etter den første fargeleggingen av grafen, blir det gjort lokale søk for å se om det er mulig å bruke færre farger.

En annen måte er å starte med et tall k. Det kjøres en algoritme for å se om grafen kan farges med k farger. Hvis grafen kan farges med k farger, reduseres k med 1, dvs. $k = k - 1$. Slik kan en fortsette med algoritmen til den finner den laveste k som algoritmen kan farge grafen med. Deretter kan en gjøre lokale søk og kanskje forbedre resultatet ytterligere (få en enda lavere k).

Parallelle algoritmer I graffargelegging er det to hovedinnfallsvinkler til parallellisering. Enten kan grafen deles opp og hver prosessor får en del å fargelegge, eller en kan fargelegge maksimale, uavhengige mengder parallelt.

Ved å dele opp grafen blir naboer fordelt på ulike prosessorer. For å hindre at disse har samme farge når grafen settes sammen, må disse naboene kontrolleres opp mot hverandre. En algoritme som bruker denne metoden finnes i [9].

Det er mange forskjellige måter å finne maksimale, uavhengige mengder parallelt. Det er også mange andre heuristikker for graffargeleggingsproblemet. For mer detaljer om forskjellige fargeleggingsalgoritmer se [8].

2.5 Selvstabiliserende algoritmer

Distribuerte systemer (f.eks. *ad hoc* nettverk) utveksler informasjon ved å sende meldinger mellom enhetene. En graf $G = (V, E)$ kan modelere dette distribuerte systemet. Hver node $v \in V$ har en mengde med variabler. Verdiene til variablene avgjør nodens **lokale tilstand**. Tilstanden til hele systemet er den **globale tilstanden**. Dette er unionen av alle de lokale tilstandene.

Hver node kan bare se dens egne og naboenes variabler. Målet til det distribuerte systemet er å komme i en global stabil tilstand. Det har da oppnådd gitte kriterier. Det er også ønskelig at et system skal være robust ovenfor feil, dvs. at det er mulig å komme tilbake til en global stabil tilstand etter en feil. Dette kan i *ad hoc* nettverk være at en node "detter ut" eller forlater dekningsområdet til en annen node.

Dijkstra introduserte selvstabiliserende algoritmer i 1974 [6]. I selvstabiliserende algoritmer kan en node endre den lokale tilstanden ved å utføre en handling h. Algoritmene er

gitt som en mengde regler $\mathcal{R} = \{r_1, r_2, ..., r_g\}$. Hver regel består av et **predikat P** og en **handling** h. Predikatet er et logisk uttrykk basert på variablene til noden og dens naboer. Handlingen er en endring av nodens variabler.

En node blir **privilegert** hvis det eksisterer et predikat som er sant. Noden kan da utføre den korresponderende handlingen.

Det **åpne nabolaget** til node i betegnes $N(i)$ og består av alle naboene til node i. Hvis en tar med node i også, får en det **lukkede nabolaget**. Dette betegnes $N[i]$.

Dersom en node ikke har noen regler med et *sant* predikat, er ikke noden privilegert. Hvis ingen av nodene er privilegert, er systemet **stabilt** (dette forutsetter at alle reglene i $\mathcal{R}$ er korrekt definert). Et distribuert system som blir stabilt etter endelig antall handlinger fra enhver initiell tildeling av verdier til variablene, er selvstabiliserende [6]. For å bevise at et system er selvstabiliserende, er det tre kriterier som må oppfylles [19]:

I. I enhver ustabil tilstand eksisterer det minst en node som er privilegert.

II. I enhver stabil tilstand eksisterer det ingen noder som er privilegerte.

III. For alle mulige initielle konfigurasjoner og for alle mulige måter en node kan velges på er systemet garantert å nå en stabil tilstand etter endelig antall handlinger.

Tiden det tar for et system å bli stabilt, kalles gjerne handlingstiden (handlingskompleksitet). I selvstabiliserende algoritmer måles handlingstiden i antall handlinger som blir utført i hele systemet. Hvis det er n noder og alle disse nodene har to regler hver som blir brukt 1 gang, sier vi at handlingstiden er $2n$. En kan da si at handlingstiden er $\mathcal{O}(n)$. Dette betyr det samme som for sekvensiell kjøretid, dvs. at lavere grads ledd og konstanter er utelatt.

En viktig forutsetning for at det skal være mulig å ha en endelig handlingstid, er at to naboer ikke kan utføre deres handlinger samtidig. Hvis to naboer utfører handlingen på de samme reglene samtidig, kan de velge den samme verdien slik at de fortsatt er privilegerte. Da kan de utføre den samme handlingen en gang til, og det er en uendelig løkke.

I denne oppgaven er det tatt utgangspunkt i en seriell modell av selvstabiliserende algoritmer. Det er en sentral ”Demon” som velger tilfeldig en node blant alle de privilegerte. Noden utfører tilfeldig handlingen til en av de reglene med *sant* predikat. På

denne måten kan en si at selvstabiliserende algoritmer er en endelig sekvens av handlinger, $h_o, h_1, h_2, ..., h_k$, hvor h_k er den k-te handlingen som blir utført. Således blir den globale tilstanden til systemet betegnet S_0 etter at handling h_0 er utført. Flere artikler og informasjon om selvstabiliserende algoritmer kan finnes her [3].

Kapittel 3

Simulatoren

Dette kapitlet omhandler simulatoren. Den ble laget for å simulere selvstabiliserende algoritmer i et *ad hoc* nettverk. Kapitlet begynner med en oversikt over de logiske komponentene til simulatoren før funksjonaliteten til disse forklares. Det går videre med å beskrive implementasjonsdetaljene til de ulike komponentene. Simulatoren består av mer enn 30 klasser og grensesnitt, men det er kun de viktigste klassene som blir beskrevet.

Det er også laget et program for generering av grafer som representerer *ad hoc* nettverk. Siste del av dette kapitlet forklarer hvordan disse grafene genereres.

3.1 Funksjonaliteten til simulatoren

Funksjonaliteten til simulatoren er delt opp i logiske komponenter. De forskjellige komponentene ble valgt for å gjøre simulatoren mest mulig generisk. Simulatoren er delt opp i følgende komponenter:

- Reglene $\mathcal{R}$.
- Nodene V.
- Variablene til nodene.
- Grafen G.
- Demonen. Dette er ”demonen” som styrer nodene.

Simulatoren kan simulere alle typer selvstabiliserende algoritmer. Derfor blir nodene tildelt en mengde med regler $\mathcal{R}$, der $|\mathcal{R}| \geq 1$. Alle nodene får den samme mengden med regler til hver simulering.

Reglene Hver regel har et predikat **P** og en handling h (move). Hvis predikatet er sant, kan reglene utføre handlingen. Reglene ser på nodens variabler og naboenes variabler. Ut ifra disse variablene bestemmer reglene seg for om de er privilegerte eller ikke.

Nodene Nodene har tre lister:

- Regellisten $\mathcal{L}_{regel}$, inneholder alle reglene til noden.
- Variabellisten $\mathcal{L}_{variabler}$, inneholder alle variablene til noden.
- Nabolisten $\mathcal{L}_{nabo}$, inneholder alle naboene til noden.

Variablene Alle nodene har en mengde med variabler. Dette kan også kalles egenskaper. En variabel kan f.eks. være nodens farge. De ulike selvstabiliserende algoritmene bruker forskjellige variabler.

Grafen Dette er grafen som representerer det fysiske rommet. Det er en vanlig urettet graf $G = (V, E)$. Grafen inneholder dataene simuleringene kjører på. Grafen har to lister, en for alle nodene og en for alle kantene. Det vil bli forklart senere hvordan grafene genereres.

Demonen For å kunne simulere flere mobile enheter på en sekvensiell datamaskin, er det laget en ”Demon”. Demonen kontrollerer og simulerer at alle de mobile enhetene er uavhengige av hverandre. Denne velger tilfeldig hvilken av de privilegerte nodene som skal utføre handlingen.

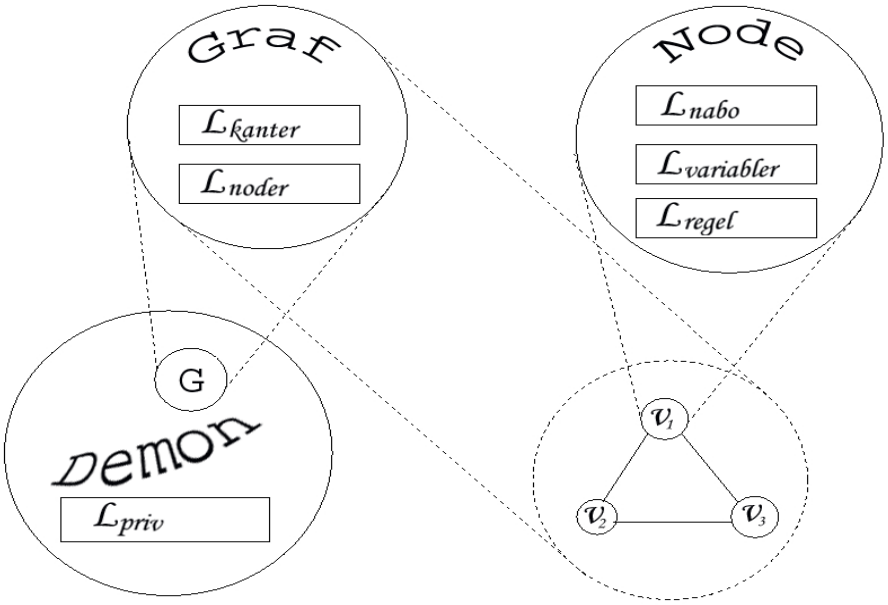
3.2 Implementasjon av simulatoren

Simulatoren er implementert i Java[1] ved hjelp av Borland Jbuilder personal 9.0.125.0 [4]. De ulike komponentene ble implementert med et grensesnitt og en klasse som implementerte

[1] JDK versjon 1.4.1-01-b01

grensesnittet.

Grafklassen har en nodeliste som inneholder nodeklasser. Nodeklassene inneholder en regelliste som inneholder regelklasser. Hele systemet blir kontrollert av Demonklassen. Denne har den overordnede kontrollen over alle de andre klassene. Demonklassen trengs også til å simulere flere uavhengige enheter samtidig. Siden vi har valgt å bruke en seriell modell av selvstabiliserende algoritmer, vil én og én node utføre handlinger. I figur 3.1 ser vi den logiske oppbyggingen av simulatoren.



Figur 3.1: Logisk bilde av simulatoren.

Alle de forskjellige komponentene har grensesnitt mellom seg. Det er derfor mulig å skifte ut alle komponentene med nye, forutsatt at de implementerer grensesnittene. I figur 3.1 ser vi et logisk bilde av simulatoren. Denomen har en liste ($\mathcal{L}_{priv}$) over alle de privilegerte nodene i grafen.

- Reglene: Alle reglene implementerer et *regel*–grensesnitt. Den ene av metodene er *erPrivilegert*. Denne kontrollerer om predikatet **P** er sant eller ikke. Metoden returnerer en boolsk verdi som er sann eller falsk. Den andre viktige metoden er *utførHandling*. Den gjør det som står i handlingen til regelen.

- Alle nodene implementerer et *node*–grensesnitt. De har metoder for å hente ut alle listene, legge til og fjerne naboer.

- ⇨ Systemgrafen G_s er en implementasjon av grafen. Det er grafgeneratoren (se seksjon 3.3) som generer datasettet til systemgrafen. G_s har to lister: nodelisten $\mathcal{L}_{noder}$ og kantlisten $\mathcal{L}_{kanter}$. Listen $\mathcal{L}_{noder}$ består av i_s (systemnoder), og listen $\mathcal{L}_{kanter}$ inneholder par av systemnoder: (i_s, j_s). Systemgrafen har metoder for å finne ut om to noder er naboer, og den kan finne gradtallet til noden. Det finnes også en metode for å finne det største gradtallet i grafen.

- ⇨ Demonen starter med et initielt steg. Reglene til alle nodene i grafen G blir kontrollert. De nodene som har minst én regel $r \in \mathcal{L}_{regel}$ som er privilegert, blir lagt til privilegertlisten $\mathcal{L}_{priv}$ i Demonen. Etter dette initielle steget begynner selve simuleringen. Demonen gjør da følgende:

  I. Velger ”tilfeldig”[2] node i fra $\mathcal{L}_{priv}$.

  II. Henter ut regellisten $\mathcal{L}_{regel}$ til i. For alle r i $\mathcal{L}_{regel}$ ser den om r er privilegert. Hvis r er privilegert, legges denne til den midlertidige regellisten $\mathcal{L}_{mid}$ til Demonen.

  III. Demonen velger ”tilfeldig” en regel r fra $\mathcal{L}_{mid}$.

  IV. Demonen utfører handlingen til r på node i som regelen tilhører. Når regelen er utført, kontrollerer demonen om noen av naboene til node i har blitt privilegert. Hvis det er noen naboer som har blitt privilegert, legges disse til $\mathcal{L}_{priv}$, og de naboene som ikke lenger er privilegert fjernes.

  V. Demonen sjekker om node i, som hadde regel r, har noen andre regler som er privilegerte. Noden skal da ikke ut av $\mathcal{L}_{priv}$. Demonen starter på nytt i punkt I inntil $\mathcal{L}_{priv}$ er tom, dvs. inntil ingen flere noder er privilegerte.

3.3 Generering av datasett: Grafgeneratoren

Grafgeneratoren skal lage datasett som blir brukt i simuleringene. Disse datasettene skal være en representasjon av et plan som inneholder n noder. Nodene skal forestille fysiske enheter med en sender/mottaker. Enhetene sender signaler med en gitt styrke som rekker

[2] Velger vha. java.util.Random klassen.

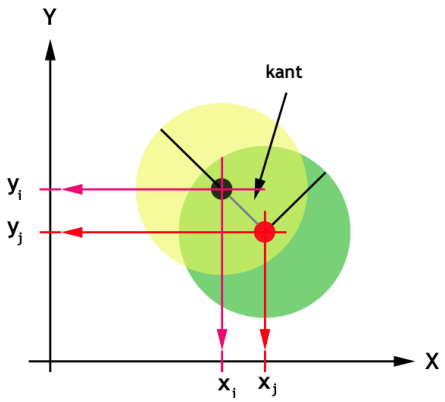
avstand d ut i rommet rundt noden. Dette danner en sirkel rundt noden. I figur 2.2 ser en et slikt plan (”rom”) med 7 enheter og deres dekningsområde.

Planet har en lengde og en bredde, og nodene skal plasseres tilfeldig ut i planet. Dette blir gjort ved å legge et rutenett over planet. Det er $X+1$ linjer i x–retningen og $Y+1$ linjer i y–retningen. Rutenettet består av $(X\ +\ 1) * (Y\ +\ 1)$ punkt (der x– og y–linjene krysser hverandre). For å lage dette planet med n noder, tar generatoren inn følgende parametre:

- $X\ +\ 1$, antall linjer i x–retning.
- $Y\ +\ 1$, antall linjer i y–retning.
- n, antall noder som skal plasseres i planet.
- d, radien til dekningsområdet.
- $filnavn$, navnet på filen grafen skal lagres i.

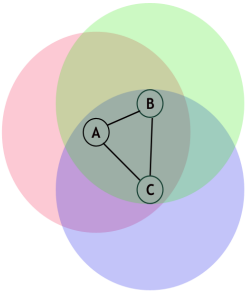
Generatoren gir hver av nodene en tilfeldig posisjon (x, y) vha. *java.util.Random* klassen. Disse posisjonene lagres i en tabell.

Nodene blir så kontrollert om de er naboer med hverandre. Nodene blir naboer hvis de har muligheter for å kommunisere med hverandre. De må da ha en avstand som er mindre enn d. Formelen for avstand mellom to punkt i planet er $dist = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, der punkt (x_i, y_i) representerer node i, og punkt (x_j, y_j) representerer node j. Dersom $dist \leq d$, vil det bli laget en kant mellom node i og node j i grafen.



Figur 3.2: To noder kan kommunisere med hverandre.

Det er også mulig å lage flere grafer der alle nodene er plassert på de samme koordinatene, men der d er forskjellig for hver graf. Da blir x og y koordinatene til alle de n nodene generert, og disse blir tatt vare på. Deretter kan en legge til den radien en ønsker. Alle de n nodene har den samme radien. Dette tilsvarer at alle sender med den samme styrken.



Figur 3.3: Kommunikasjonsmuligheter mellom tre noder og tilhørende kanter.

I figur 3.2 er det illustrert hvordan to noder blir naboer. Når $dist \leq d$, blir node *svart* og node *rød* naboer. Figur 3.3 viser hvordan 3 noder blir naboer og danner en *klikk*. Slik vil generatoren fortsette og legge kanter mellom noder som er mindre enn *dist* fra hverandre. Grafgeneratoren kan ikke kontrollere om grafene, som blir generert, er sammenhengende eller planare. Derfor kan en ikke anta at grafene er noen av delene.

Kapittel 4

Selvstabiliserende algoritmer

I dette kapitlet blir noen eksisterende selvstabiliserende fargeleggingsalgoritmer forklart. Disse er ment som en introduksjon til hvordan en kan bruke selvstabiliserende algoritmer til graffargelegging.

Alle algoritmene blir først forklart med ord, og deretter kommer den formelle definisjonen. Det følger ett eksempel til hver algoritme og til sluttt en beskrivelse av kompleksiteten til algoritmen. Algoritmene utgjør et utgangspunkt som vi skal forbedre senere i oppgaven.

4.1 Rask fargelegging (RF)

Denne algoritmen bruker maksimalt $\Delta + 1$ farger på å fargelegge en graf. Når algoritmen er stabil, har alle nodene ulik farge fra naboene.

Nodene har en variabel $c(i) \in \{0, 1, ...\}$ som representerer fargen til node i. Gradtallet til node i betegnes d_i. Alle nodene i grafen har den samme regelen:

- Regel: *Finn tilfeldig en ledig farge som ikke er brukt av noen naboer og velg denne.*

  Predikat : $\Big(c(i) \in \{ c(j) \mid j \in N(i) \} \Big) \vee \Big(c(i) > d_i \Big)$

  Handling : hvis $\Big\{ c(j) \mid j \in N(i) \Big\} = \Big\{ 0, 1, ..., d_i - 1 \Big\}$

  så sett $c(i) = d_i$

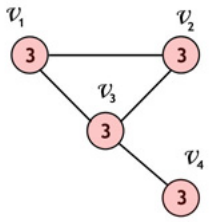
ellers velg $c(i)$ tilfeldig blant $\{0, 1, ..., d_i - 1\} - \{ c(j) \mid j \in N(i) \}$

Algoritmen velger kun $\Delta + 1$ fargen hvis alle de andre fargene er opptatt. Dette kan settes i sammenheng med at den først og fremst vil prøve på en Δ fargelegging, men går ikke det vil den ta i bruk en ekstra farge.

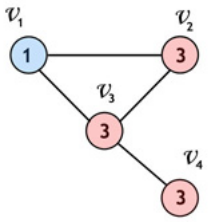
Eksempel Her er et eksempel med rask fargelegging:
Rekkefølgen på nodene som utfører handlingen, er tilfeldig. For å gjøre eksemplet enklest mulig, blir kun én handling utført om gangen på samme måte som i simulatoren.

Alle nodene i figur 4.1 har samme farge ved start. Dermed er alle de fire nodene privilegerte. Det største gradtallet i grafen er 3 ($\Delta = 3$), og det er derfor 4 tillatte farger som kan brukes på fargeleggingen. Vi har da at $|\{0, 1, 2, 3\}| = \Delta + 1$.

Node v_1 ser på naboene i figur 4.1, og den finner de ledige fargene. Når v_1 har funnet



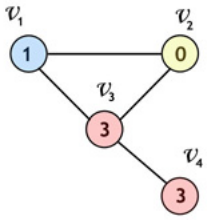
Figur 4.1: Ved start har nodene samme farge.



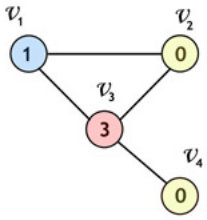
Figur 4.2: Node v_1 har valgt ny farge.

de ledige fargene, velger den en av disse. I figur 4.2 har node v_1 valgt farge 1. Dette valget var et tilfeldig valg blant de ledige fargene {0, 1, (2)}. Node v_1 kan bare velge farge 2 hvis 0 og 1 er opptatt (hvis alle fargene fra 0 til $d_i - 1$ er opptatt, skal d_i velges). I figur 4.2 ser node v_2 på naboene for å finne hvilke farger som er ledige. Node v_2 kan velge i mengden {0, (2)} siden node v_1 valgte 1.

Node v_2 må velge farge 0, som vi kan se i figur 4.3. Det er nå v_4 som skal velge farge. Node v_4 har bare en farge den kan velge, og det er 0. Siden *deg(v_4)* = 1, kan den bare velge d_i hvis alle de lavere fargene er opptatt. Når v_4 har valgt farge 0, er ikke node v_3 lenger privilegert. Den har ikke samme farge som noen av naboene, og fargen den har, er en av de tillatte fargene ($\{0, 1, 2, 3\}$). Det er ingen noder som er privilegert lenger, og systemet er da



Figur 4.3: Node v_2 har valgt ny farge.



Figur 4.4: Node v_4 har valgt ny farge.

stabilt.

Det er også mulig å farge denne grafen med færre enn $\Delta + 1$ farger. Hadde en i figur 4.3 valgt node v_3 istedenfor v_4, kunne v_3 valgt farge 2 (den måtte ha endret farge siden v_4 har samme farge). Node v_4 kunne fortsatt ha valgt farge 0. Algoritmen ville da ha brukt Δ farger.

Teorem 4.1.1 *Gitt en graf med n noder, vil RF–algoritmen finne en $\Delta + 1$ fargelegging i maksimalt n handlinger.*

For bevis av teorem 4.1.1 se [11]. Hovedidéen i beviset er at etter enhver handling utført av node i, er fargen til node i mellom 0 og d_i. Når en node har utført en handling, blir den korrekt farget og forblir det. Ingen av naboene kan endre dette. Videre vises det at en node bare kan gjøre maksimalt en handling, og dermed har denne algoritmen en kjøretid på n handlinger.

4.2 Laveste ledige farge (LLF)

Denne algoritmen er veldig lik RF–algoritmen. Forskjellen er at denne vil velge den laveste fargen som er ledig, og ikke en tilfeldig blant de ledige. LLF–algoritmen bruker heller ikke mer enn $\Delta + 1$ farger. Når algoritmen er stabil, har alle nodene den laveste ledige fargen, og fargen er mindre enn $\Delta + 1$. Akkurat som i RF har nodene en variabel $c(i)$ som er fargen til node i. Algoritmen har en regel:

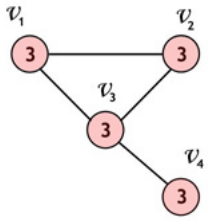
- Regel: *Finn den laveste ledige fargen som ikke er brukt av noen naboer og velg denne.*

  Predikat : $c(i) \neq min \left\{ l \geq 0 \mid (\forall j \in N(i))(c(j) \neq l) \right\}$

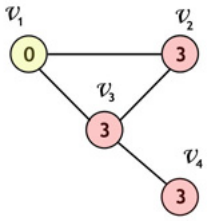
$$\textbf{Handling :} \text{ sett } c(i) = min\left\{ l \geq 0 \mid (\forall j \in N(i))(c(j) \neq l) \right\}$$

Eksempel Her følger et eksempel med LLF–fargelegging:

I figur 4.5 har alle nodene samme farge. Alle nodene er da privilegert. Node v_1 ser på naboene i figur 4.5 og finner fargene til dem. Node v_1 velger den laveste ledige fargen som er 0. Node v_3 er neste til å utføre handlingen. Fargene 0 og 3 er opptatt av naboene. Node

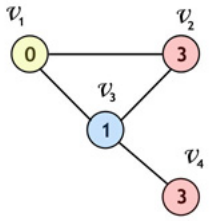


Figur 4.5: Ved start har nodene samme farge.

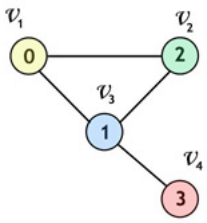


Figur 4.6: Node v_1 har valgt ny farge.

v_3 velger da 1, som er den laveste tilgjengelige fargen. I figur 4.7 ser vi at node v_3 har valgt farge 1. Node v_2 har farge 3, og naboene til v_2 har brukt fargene 0 og 1. Node v_2 må derfor



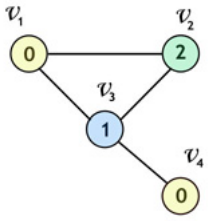
Figur 4.7: Node v_3 har valgt laveste ledige farge.



Figur 4.8: Node v_2 har valgt ny farge.

velge farge 2. Det er nå node v_4 som skal velge farge. I figur 4.8 har v_4 farge 3. Naboen v_3 har farge 1. Node v_4 kan derfor velge farge 0, som er den laveste fargen det er mulig å velge. I figur 4.9 har v_4 valgt den laveste fargen som var tilgjengelig. Alle nodene i figur 4.9 har forskjellig farge fra naboene, og det er den laveste fargen de hadde mulighet til å velge. Det er ingen noder som er privilegert lenger, og systemet er dermed stabilt.

I dette eksemplet bruker LLF–algoritmen ikke mer enn Δ farger. Det finnes eksempel



Figur 4.9: Node v_4 har valgt den laveste ledige fargen.

der denne må bruke $\Delta + 1$ farger. F.eks. den komplette grafen K_4.

Teorem 4.2.1 *Gitt en graf med n noder og m kanter, vil LLF–algoritmen finne en LLF–fargelegging i maksimalt n + 2m handlinger.*

For bevis av teorem 4.2.1 se [11]. Hovedidéen er som følger. Etter enhver handling av en node, er fargen mellom 0 og d_i. En node i kan gjøre maksimalt $d_i + 1$ valg. Noden kan bare gjøre ett økende fargevalg som må skje i den første handlingen, og deretter kan det bare velges en lavere farge. Det økende valget kan bare skje dersom den ikke var ordentlig fargelagt i den initielle tilstanden til systemet. Det vises at hver node kan gjøre maksimalt $d_i + 1$ handlinger, og dermed er det en begrensning på antall handlinger utført. Det kan derfor vises at algoritmen har en kjøretid på $n + 2m$ handlinger.

4.3 Maksimal, uavhengig mengde (MUM)

Algoritmen finner en maksimal, uavhengig mengde. En node kan enten være med i denne mengden eller ikke. Nodene har en boolsk variabel, som betegnes $s(i)$. Betegnelsen $s(i)$ betyr at node i er med i den maksimale, uavhengige mengden τ, og $\overline{s(i)}$ betyr at node i ikke er med i τ.

Denne algoritmen er brukt til å teste om simulatoren kan håndtere flere regler. Den vil ikke bli brukt til noe annet i denne oppgaven. Algoritmen har to regler:

- R_1: *Hvis denne noden og alle naboene ikke er med i mengden τ, sett denne noden til å være med i τ.*

  Predikat : $\overline{s(i)} \wedge (\forall\, j \in N(i) \mid \overline{s(j)})$

Handling : sett $s(i) = sann$.

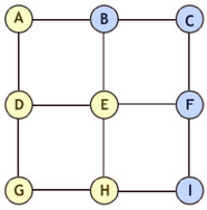
- R_2: *Hvis denne noden er med i τ og det eksisterer en nabo som er med i τ, sett denne noden til ikke å være med i τ.*

  Predikat : $s(i) \wedge (\exists\, j \in N(i) \mid s(j))$

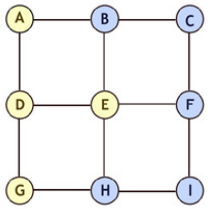
  Handling : sett $s(i) = falsk$.

Eksempel Her kommer et eksempel med MUM–algoritmen:

Ved start i figur 4.10 har nodene A, D, E, G og H satt variabelen $s(i) = sann$, og nodene B, C, F og I har satt $s(i) = falsk$. Ikke alle nodene er privilegerte ved start. Nodene A, D, E, G og H har alle en nabo som er med i τ. De er derfor privilegerte i følge regel 2. I figur 4.11



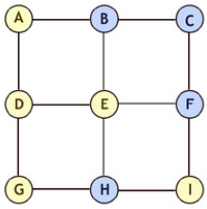
Figur 4.10: Ved start har noen noder $s(i) = sann$.



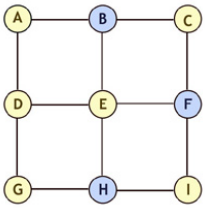
Figur 4.11: Node H er gått ut av mengden τ.

har H utført R_2. Noden har satt seg selv ut av mengden τ. Denne noden ble valgt tilfeldig blant de privilegerte. Etter at H har gått ut av τ, er I blitt privilegert iht. R_1.

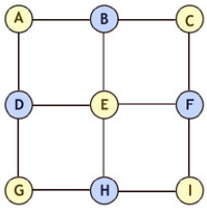
Node I er den neste noden som ble valgt tilfeldig. Siden ingen av naboene til I er med i mengden τ, setter I seg til å være med i mengden τ (se figur 4.12). I figur 4.13 har C brukt R_1 for å bli med i τ siden den i figur 4.12 ikke var med i τ og heller ingen av naboene var med. Node D bruker R_2 for å gå ut av mengden τ. I figur 4.14 er nodene A, C, E, G og I med i den maksimale, uavhengige mengden τ. Det er nå ingen noder som er privilegert lenger, og systemet er stabilt.



Figur 4.12: Node I er blitt med i τ.



Figur 4.13: Node C er blitt med i τ.



Figur 4.14: Node D har gått ut av mengden τ.

Teorem 4.3.1 *Maksimal, uavhengig mengde algoritmen finner én maksimal, uavhengig mengde med maksimalt 2n handlinger.*

Teorem 4.3.1 er bevist i [12]. Hovedidéen er som følger: En node som blir med i mengden τ, vil aldri gjøre flere handlinger. En node kan aldri bruke samme regel to ganger etter hverandre. Dermed vil en node maksimalt bruke to regler. En node som var med i τ ved start, kan gå ut av τ og etterpå gå inn i τ.

Kapittel 5

Nye selvstabiliserende algoritmer

Dette kapitlet omhandler modifiserte og nye selvstabiliserende fargeleggingsalgoritmer. I kapittel 4 ble det presentert algoritmer som bruker opp til $\Delta + 1$ farger. Vi ønsker å se hva som skjer hvis en har mindre enn $\Delta + 1$ farger eller frekvenser. Algoritmene i dette kapitlet blir gitt et fast antall tillatte farger.

Den første algoritmen er en modifikasjon av RF–algoritmen fra seksjon 4.1. Den neste er *Peker flagg–algoritmen* som er en ny selvstabiliserende fargeleggingsalgoritme. Denne har muligheter til å forhandle frem fargebytte mellom noder.

5.1 Modifikasjon av rask fargelegging (MRF)

I denne algoritmen innføres det en ny variabel $maks$ som er den største tillatte fargen en node kan ha. RF–algoritmen bruker maksimalt $\Delta + 1$ farger på å fargelegge en graf, mens her skal det brukes $maks$ farger som gjerne er mindre enn $\Delta + 1$.

Algoritmen er lik RF, men er det ikke noen ledig farge å velge, settes fargen til -1 (ingen farge). Den består av en regel med et 3–delt predikat. Predikatets 3 muligheter for å bli *sant* er: Noden har samme farge som en nabo, fargen er høyere enn tillatt eller fargen er -1 (*ingen farge*) og det er ledige farger å velge.

- Regelen: *Finn tilfeldig en ledig farge som ikke er brukt av noen naboer og velg denne. Er det ikke noen ledige farger velg -1 (ingen farge).*

  Predikat : $c(i) \in \left\{ c(j) \mid j \in N(i) \right\}$

$$\bigvee$$

$$c(i) >= maks$$

$$\bigvee$$

$$c(i) = -1 \bigwedge \Big\{c(j) \mid j \in N(i)\Big\} \neq \Big\{0, 1, ..., (maks - 1)\Big\}$$

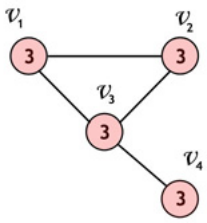
Handling : hvis $\Big\{c(j) \mid j \in N(i)\Big\} = \Big\{0, 1, ..., (maks - 1)\Big\}$

så sett $c(i) = -1$

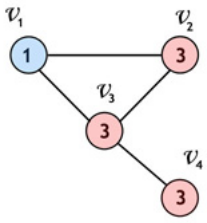
ellers velg $c(i)$ tilfeldig blant $\Big\{0, 1, ..., (maks - 1)\Big\}$ - $\Big\{c(j) \mid j \in N(i)\Big\}$

Denne algoritmen kalles modifisert tilfeldig rask fargelegging (MRF). Den er tilfeldig fordi fargen blir valgt tilfeldig blant de ledige fargene. MRF–algoritmen vil bli brukt i kapittel 6. Der vil det bli kjørt simuleringer på datasett med denne regelen.

Eksempel Her følger et eksempel med MRF–algoritmen:
La oss si at det er 2 farger tilgjengelig ({0, 1}). Ved start har alle nodene den samme ulovlige fargen og er dermed privilegert. Node v_1 starter med å se på naboene og finner fargene de bruker. Både farge 0 og 1 er ledige. Den velger tilfeldig blant de ledige fargene. I figur 5.2

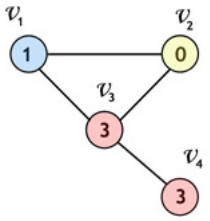


Figur 5.1: Alle noder har samme farge ved start.

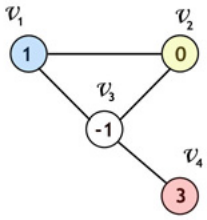


Figur 5.2: Node v_1 har valgt farge.

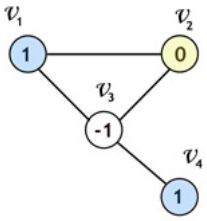
har v_1 valgt farge 1 tilfeldig. Node v_2 skal nå velge farge. Siden v_1 har farge 1, må v_2 velge 0. I figur 5.3 har v_2 valgt farge 0. Det er nå v_3 som skal velge farge. Den har samme farge som v_4. De to andre naboene til v_3 har brukt opp de lovlige fargene {0, 1}. Derfor må v_3 velge -1 (*ingen farge*). Etter at v_3 har valgt -1, har v_4 fortsatt en farge som ikke er lovlig.



Figur 5.3: Node v_2 har valgt farge.



Figur 5.4: Node v_3 har valgt farge -1 (ingen farge).



Figur 5.5: Node v_4 har valgt farge 1.

Den må velge en av de lovlige fargene, og velger farge 1 tilfeldig. Når v_4 har valgt farge, som vi ser i figur 5.5, er det ikke lenger noen noder som er privilegert. Systemet er nå stabilt, siden ingen noder har en regel med et predikat som er *sant* lenger.

Algoritme 5.1.1 *Her kommer pseudokoden for handlingen til MRF–algoritmen:*

Handling:

```
Hent nabolisten L_nabo til noden;
bruktFarge = ny boolsk[maksfarge]; /* alle blir satt til falsk */
gjenta(for alle naboene)
  bruktFarge[naboen sin farge] = sann; /* fargen er opptatt */

muligeFarger = ny liste;

gjenta(for i = 0 til lengden på bruktFarge)
    hvis(bruktFarge[i] != sann)
       muligeFarger legg til i;

hvis(muligeFarger sin størrelse > 0)
  velg en tilfeldig farge fra muligeFarger og
  sett denne fargen til noden;
ellers
  Det er ingen ledige farger å velge,
  sett noden til ufarget;
```

I linje 09 til algoritme 5.1.1 er det en kontroll på om det er noen ledige farger. Det er en grunn til at fargen -1 (*ingen farge*) ikke er med i *muligeFarger*-listen. Hvis den hadde vært med, er det en mulig sannsynlighet for en uendelig løkke. Her er et tilfelle fra eksemplet:

Node v_1 i figur 5.1 kan velge mellom {0, 1} $\cup$ {*ingen farge*}. Hvis v_1 velger {*ingen farge*}, er den fortsatt privilegert. Det er andre farger som er bedre enn -1 (*ingen farge*) som v_1 kan velge. Node v_1 er fortsatt privilegert etter at den har valgt en ny farge. Den kan dermed velge -1 (*ingen farge*) på nytt og vi har starten av det som kan bli en uendelig løkke.

Denne algoritmen kan også velge den laveste ledige fargen. I linje 06 går den igjennom de ledige fargene. Den kan da velge den laveste fargen istedenfor å legge alle de ledige fargene i listen. Den vil da bli en modifikasjon av laveste, ledige farge ($MLLF$) .

Handlingstiden Hvis $maks$ (antall tillatte farger) $= \Delta + 1$, er tiden denne algoritmen bruker den samme som rask fargelegging. Det kan derfor argumenteres for at MRF–algoritmen ikke bruker lenger tid enn rask fargelegging. Fra teorem 4.1.1 har vi at RF–algoritmen ikke bruker mer enn n handlinger. MRF–algoritmen vil ikke lage en korrekt fargelegging med mindre det er nok farger tilgjengelig.

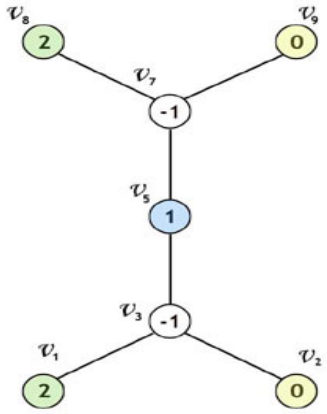
5.2 $PekerFlagg$–algoritmen (PF)

I innledningen til oppgaven var det 3 spørsmål som skulle besvares. Ett av dem var om en algoritme der nodene har muligheter til å forespørre naboene om fargeendring ville være bedre enn en enkel algoritme.

I denne delen blir det presentert en ny algoritme som har muligheter til å forespørre naboene om fargeendring. Motivasjonen bak algoritmen blir først presentert. Deretter kommer noen begrensninger som en må ta hensyn til. Algoritmen blir presentert regel for regel med eksempler. Dette vil gjøre innføringen i prinsippene bak algoritmen enklere. Til slutt vil kompleksiteten til algoritmen bli analysert.

5.2.1 Motivasjon for en bedre algoritme

Ved å se på begrensningene til det en ønsker å forbedre, kan motivasjonen for en ny algoritme komme bedre frem. I figur 5.6 er det tillatt å bruke 3 farger. Figuren viser et utsnitt av en graf der tilstanden er stabil. Dette er etter at MRF–algoritmen har utført handlingene til



Figur 5.6: Utsnitt av en graf der MRF–algoritmen er brukt.

de reglene som var privilegerte. Nodene v_3 og v_7 er ufarget. De har -1 (ingen farge). Hvis v_5 hadde endret farge til f.eks 0, kunne både v_3 og v_7 fått en lovlig farge.

Dersom enten v_3 eller v_7 kunne forespurt v_5 om å endre farge, ville v_3 og v_7 fått en lovlig farge. Motivasjonen for den nye algoritmen er nettopp dette. Vi ønsker å få til en forespørsel mellom noder for fargeendring.

5.2.2 Begrensninger som en må ta hensyn til

Algoritmen må vite hvilke av naboene som har mulighet til å endre farge. Det er ikke vits å forespørre en nabo som ikke kan bytte farge. Det må derfor være et signal som indikerer at en node kan bytte farge.

Hvis en node har to naboer med samme farge, men kun en av dem kan bytte farge, kan ikke noden forespørre denne om fargeendring. Det vil ikke bli frigjort en farge hvis bare den med ledige farger bytter.

Algoritmen skal også være selvstabiliserende. En må derfor aldri komme tilbake til en tidligere tilstand. Vi må hindre at en node forespørr en nabo om fargeendring, og senere i algoritmen har denne naboen byttet tilbake til fargen den hadde tidligere. Igjen forhører

den samme noden naboen om fargeendring. Dette vil forklares nærmere i eksemplene til reglene.

5.2.3 Prinsippene bak algoritmen

Denne delen forklarer prinsippene og reglene til algoritmen. For å signalisere at det er ledige farger å velge og at en node ønsker å få endret fargen til en av naboene, innføres det noen variabler.

- $maks$: Antall tillatte farger. Dvs. fargene i intervallet $\{0, 1, ..., maks - 1\}$ er tillatte farger.
- $c(i)$: $Fargen$ til node i er enten -1 (ingen farge) eller en av de lovlige fargene. (Kan være vilkårlig farge ved start, også større enn $maks$).
- $f(i)$: $Flagget$ er enten *sant* eller *falskt*. Denne variablen forteller om en node har ledige farger å velge.
- $p(i)$: $Pekeren$ er enten $null$ eller den peker på en annen node. Denne noden ønsker da at den det pekes på skal bytte farge.
- id : $Identiteten$ er et unikt nummer som identifiserer noden. Det antas at alle nodene i grafen har en unik id. Dette er det samme som å nummerere nodene fra 1 til n.

Det må lages regler som endrer pekeren, flagget og bytter farge. Vi har løst dette med å innføre 7 regler. Det er to regler for å bytte farge, to regler for å endre flagget og to regler for å endre pekeren. Til slutt en siste regel for å bytte farge fra en farge til en annen. Vi antar at regel R_i ikke kan utføres før reglene $R_1 \ - \ R_{i-1}$ er utført. Dette kan implementeres ved at regelen R_i sjekker om R_{i-1} er privilegert. Hvis R_{i-1} er privilegert, skal ikke R_i være privilegert. Dette blir en ekstra sjekk til hvert predikat til hver regel. F.eks. kan ikke R_3 utføres hvis R_2 er privilegert, og R_2 kan ikke utføres hvis R_1 er privilegert. De to første reglene er for valg av farge.

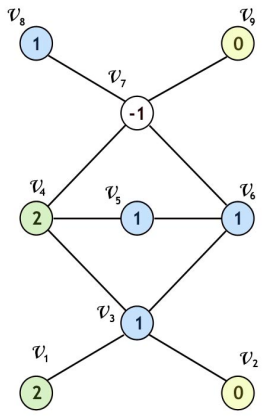
R_1**:** Hvis { ($c(i)$ er samme som en nabo og $c(i) \ \neq \ -1$) }

$$c(i) \ = \ -1$$

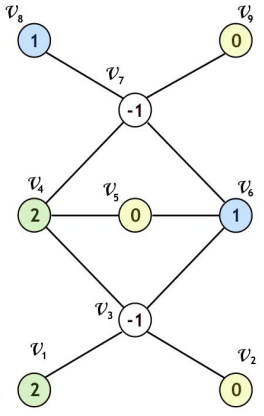
R_2**:** Hvis { ($c(i) = -1$) og (det er ledig farge) }

$c(i)$ = tilfeldig ledig farge

Kommentar til R_1: To naboer kan ha -1 fargen, som gjør at de har den samme fargen, uten å være privilegert. I figur 5.7 har nodene v_3 og v_5 samme farge som en av deres naboer. Det er 3 tillatte farger i figuren. Nodene v_3 og v_5 er derfor privilegert i henhold til predikatet til R_1. Begge nodene vil utføre handlingen til R_1 og sette fargen til -1. Etter at v_5 har satt



Figur 5.7: Den initielle tildelingen av farger til nodene.



Figur 5.8: Node v_3 har brukt R_1 og v_5 har brukt R_1 og R_2.

fargen til -1, blir dens R_2 privilegert. Den har -1 fargen og det er ledige farger å velge. Node v_5 utfører handlingen til R_2. I figur 5.8 har v_5 utført begge fargeleggingsreglene, mens v_3 har utført den første.

De to første reglene utgjør en vanlig selvstabiliserende fargeleggingsalgoritme. Regelen R_1 setter alle ulovlig fargede noder til ufarget (-1), mens R_2 endrer fargen fra -1 til en lovlig farge dersom det er ledige farger. De resterende 5 reglene er utelukkende med for å redusere antall ufargede noder.

De neste to reglene er peker–reglene, men en kan ikke peke på noen noder før de har tatt opp flagget som indikerer at de har mulighet for å bytte farge. Derfor presenteres flagg–reglene først.

R_5**:** Hvis { ($c(i) \neq -1$) og (har ledige farger) og (ingen peker på meg) og (flagg ikke heist) }

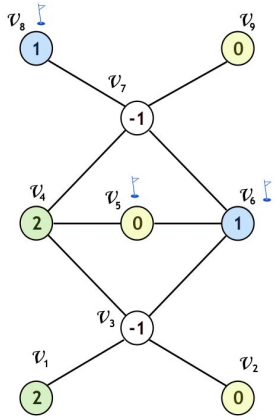
heis flagg

R_6**:** Hvis { (flagg heist) og (($c(i) = -1$) eller (har ikke ledige farger)) }

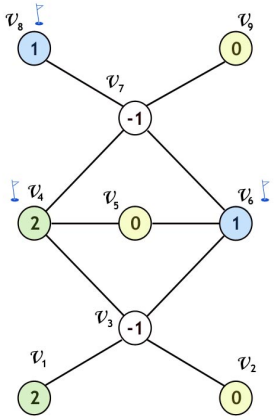
ta ned flagg

Kommentar til R_5: Ingen noder kan peke på den som skal ta opp flagget. Hvis det var noen som pekte på noden, er det en mulig sannsynlighet for en uendelig løkke ved alternerende utførsler av R_5 og R_7.

I figur 5.9 har noen noder flaggene oppe. Node v_5 har flagget oppe, men den har ikke noen ledige farger som kan velges. Derfor vil v_5 bruke R_6 til å ta ned flagget. Node v_4 har ikke flagget oppe i figur 5.9. Den vil derfor bruke R_5 til å ta opp flagget. I figur 5.10 har



Figur 5.9: Node v_4 har ikke flagget oppe, mens v_5 har flagget oppe.



Figur 5.10: Node v_4 har tatt opp flagget, mens v_5 har tatt det ned.

v_4 tatt opp flagget, mens v_5 har tatt det ned. Vi kan anta at de nodene som ikke har flagget oppe i 5.10 har noen naboer som hindrer dem i å ta opp flagget.

Det er fortsatt to noder som er ufarget (har -1 fargen). For at disse skal ha mulighet til å få en farge, må de kunne forespørre naboene om bytte av farge. Til dette bruker de peker–reglene:

R_3**:** Hvis { ($c(i) = -1$) og (har unik nabo m/flagg) og (peker ikke satt til en unik nabo m/flagg) }

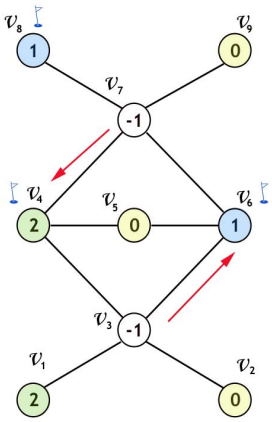
peker = unik nabo m/flagg

R_4: Hvis { (peker $\neq$ null) og (($c(i)$ $\neq$ -1) eller (peker ikke på unik nabo m/flagg) }

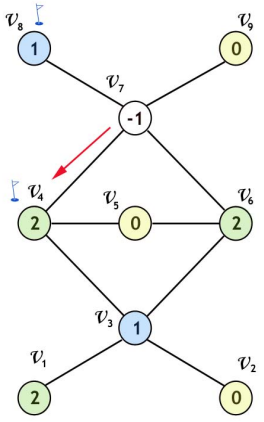
peker = null

I regelen R_3 blir en nabo definert til å være *unik* hvis fargen den har er unik i nabolaget til noden som undersøker. I figur 5.11 har nodene v_3 og v_7 satt pekerne til naboer som er unike for dem med flagget oppe. Node v_3 peker på v_6, og node v_7 peker på v_4. Da har begge to brukt R_3.

Når nodene v_3 og v_7 har satt pekerne på henholdsvis v_6 og v_4, kan det oppstå et problem. Hvis v_4 og v_6 bytter farge (de må da bytte til hverandres farger), kan pekerne flytte seg. Dvs. v_3 vil peke på v_4, og v_7 vil peke på v_6 (dette pga. fargene til de andre naboene til peker–nodene). Igjen kan de nodene det pekes på bytte farge, og vi har starten på en uendelig løkke. Dette unngår vi ved å innføre en unik id til hver node. Dette gir en prioritering mellom de nodene som har felles naboer og pekere på seg. I figur 5.11 har nodene v_4 og v_6 felles



Figur 5.11: Nodene v_3 og v_7 peker på unike naboer.



Figur 5.12: Node v_6 har byttet farge og v_3 fått en farge og tatt vekk pekeren.

naboer og pekere på seg. Noden v_6 ser at noen av dens naboer har peker ulik null. Siden noden med den laveste id'en av de naboene med pekere ulik null peker på v_6, kan v_6 utføre handlingen (bytte farge). I node v_4 sitt tilfelle er det motsatt. Den har også to naboer som

peker på andre, men her peker ikke den noden med lavest id på v_4. Dette fører til at v_4 ikke kan utføre handlingen.

Det er også en annen mulighet for at det oppstår en uendelig løkke. Hvis v_7 peker på v_8 i figur 5.11, vil v_8 bytte farge. Den fargen som v_8 byttet fra er fortsatt ledig for v_8, og det er ingen andre kriterier som har endret seg. Dermed er v_8 fortsatt privilegert og kan bytte tilbake til den fargen den hadde tidligere. Dette er starten på en uendelig løkke. Hvis noden som bytter farge samtidig tar ned flagget, vil denne mulige uendelige løkken bli unngått. Den siste regelen blir derfor slik:

R_7**:** Hvis { (flagg er oppe) og (noen peker på meg)

og

(det er ingen naboer med lavere id som peker på andre) }

bytt farge

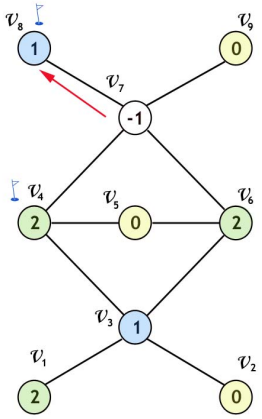
og

ta ned flagget

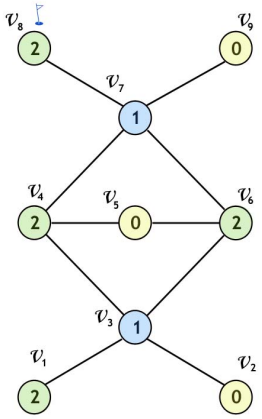
I figur 5.12 har v_6 brukt regel R_7 og byttet farge (Fordi $3 < 7$, kan ikke v_4 bytte farge før v_3 tar vekk pekeren. Se figur 5.11). Den har da også tatt ned flagget. Node v_3 har brukt to regler. Først ble R_2 utført, og den valgte en ledig, lovlig farge. Deretter ble R_4 utført, og pekeren ble tatt vekk. Siden v_6 endret farge i figur 5.12 til 2, er ikke v_4 lenger unik for v_7. Regelen R_3 til v_7 vil bli privilegert og v_7 flytter pekeren over til v_8, som vi ser i figur 5.13. Etter dette vil v_8 bruke R_7 til å bytte farge. Node v_7 får da en ledig farge og bruker R_2 til å velge den. Deretter bruker v_7 regel R_4 til å ta vekk pekeren fra v_8. Da kan v_8 ta opp flagget sitt (R_5). Til slutt vil v_4 ta ned flagget (R_6). Dette var den siste regelen som var privilegert i systemet. I figur 5.14 ser en den stabile tilstanden til systemet der alle nodene har fått en lovlig farge.

5.2.4 Algoritmen

Her kommer den formelle definisjonen til hver regel. Først kommer en forklarende tekst, og deretter kommer den formelle regelen. For å gjøre fremstillingen av reglene lettere, definerer



Figur 5.13: Node v_7 flytter pekeren til node v_8.



Figur 5.14: Ingen noder er privilegert lenger og systemet er stabilt.

vi u til å være en unik nabo med flagget oppe.

- R_1: *Hvis denne noden har samme farge som en nabo, velg -1 (ingen farge).*

  Predikat : $\Big(c(i) \in \{ c(j) \mid j \in N(i) \} \Big) \wedge \Big(c(i) \neq -1 \Big)$

  Handling : sett $c(i) = -1$

- R_2: *Hvis denne noden har -1 fargen og det er ledige farger, velg en ledig farge.*

  Predikat : $\Big(c(i) = -1 \Big) \wedge \Big(\{0, 1, ..., (maks - 1)\} - \{ c(j) \mid j \in N(i)\} \neq \emptyset \Big)$

  Handling : velg $c(i)$ tilfeldig blant $\{0, 1, ..., (maks - 1)\} - \{c(j) \mid j \in N(i)\}$

- R_3: *Hvis denne noden har -1 fargen og det eksisterer en unik nabo med flagg og pekeren ikke er satt til en unik nabo med flagg, sett peker til unik nabo med flagg.*

  Predikat : $\Big(c(i) = -1 \Big) \wedge \Big(\exists\, u \in N(i) \wedge p(i) \neq u \Big)$

  Handling : $p(i) = u$

- R_4: *Hvis denne noden peker på noen og enten har en farge ulik -1 (ingen farge) eller ikke peker på en unik nabo med flagg, så ta vekk pekeren.*

  Predikat : $\Big(\ p(i) \ \neq \ null \ \Big) \bigwedge \Big(\ c(i) \ \neq \ -1 \bigvee p(i) \ \neq \ u \Big)$

  Handling : $p(i) \ = \ null$

- R_5: *Hvis denne noden har farge og den har ledige farger og ingen peker på den og flagget ikke er oppe, ta opp flagget.*

  Predikat : $\Big(\ c(i) \ \neq \ -1 \ \Big) \bigwedge \Big(\ \{0, 1, ..., (maks - 1)\} \ - \ \{ \ c(j) \mid j \ \in \ N(i)\} \ \neq \ \emptyset \ \Big)$

  $$\bigwedge \Big(\ \nexists \, p(j) \ = \ i \, \forall \, j \ \in \ N(i) \ \Big) \bigwedge \Big(\ f(i) \ = \ falsk \ \Big)$$

  Handling : $f(i) \ = \ sann$

- R_6: *Hvis denne noden har flagget oppe og enten har -1 fargen eller ikke har ledige farger å velge, ta ned flagget.*

  Predikat : $(\ f(i) \ = \ sann \ \Big) \bigwedge$

  $$\Big(\ \Big(\ c(i) \ = \ -1 \ \Big) \ \bigvee \ \Big(\ \{0, 1, ..., (maks - 1)\} \ - \ \{ \ c(j) \mid j \ \in \ N(i)\} \ = \ \emptyset \ \Big) \ \Big)$$

  Handling : $f(i) \ = \ falsk$

- R_7: *Hvis flagget er oppe og noen peker på meg og ingen naboer med lavere id enn den med lavest id av de som peker på meg, peker på noen andre, velg ny farge og ta ned flagget.*

Predikat : $\Big(f(i) \ = \ sann \Big) \bigwedge \Big(\exists \, j \ \in \ N(i) \mid p(j) \ = \ i \Big)$

$$\bigwedge \Big(\forall \, j \ \in \ N(i) \mid p(j) \ = \ i$$

$$\nexists \, x \ \in \ N(i) \mid (id(x) \ < \ id(j) \ \wedge \ p(x) \ \neq \ i \ \wedge \ p(x) \ \neq \ null) \Big)$$

Handling : $\Big($ velg $c(i)$ tilfeldig blant $\{0, 1, ..., (maks - 1)\} -$

$$(\ \{\ c(j) \mid j \ \in \ N(i)\} \ \cup \ \{\text{tidligere farge}\}\ ) \text{ og } f(i) \ = \ falsk \Big)$$

5.2.5 Bevis for stabilitet og kompleksitet

I denne delen skal vi bevise at PF–algoritmen er selvstabiliserende og bestemme kompleksiteten til den. Innledningsvis ble det nevnt 3 punkt som en måtte bevise for at et system skulle bli stabilt (se side 17). Disse 3 kravene viser bare til konvergensen av systemet. Vi vil bevise at systemet stabiliserers ved å begrense antall ganger hver regel kan bli utført. Dermed får vi stabiliteten ved kompleksiteten (husk at kompleksiteten til selvstabiliserende algoritmer måles i antall handlinger som blir utført). Først kommer noen observasjoner. Når systemet er stabilt, er det ingen noder som er privilegerte. Det er da ingen noder som har

- Samme farge som en nabo, R_1.
- $c(i) \ = \ -1$ og ledige farger, R_2.
- Peker satt til en unik node med flagg oppe. Hvis pekeren er feil, R_4. Hvis flagget er feil, R_6. Hvis alt er rett, R_7.

For alle nodene gjelder:

- Alle pekerne er satt til null.
- Alle naboene til en node med flagg har $c(i) \ \neq \ -1$.
- En node med $c(i) \ = \ -1$ har ingen unike naboer med flagget oppe.

Det er bare R_2 og R_7 som kan sette fargen til en node ulik -1. Siden R_2 og R_7 bare velger tillatte farger, følger det at når en node først har fått en lovlig farge (ulik -1), vil den være lovlig farget resten av algoritmen. Når en node har utført R_1, vil den deretter aldri bli ulovlig farget. Dette betyr at hvis algoritmen stabiliserer seg, vil den ha en lovlig farging.

Stabilisering og kompleksitet Her kommer først noen lemma som vil bli brukt i beviset. Deretter følger beviset for stabilitet og kompleksitet.

Lemma 5.2.1 *Reglene R_1 og R_2 kan bare utføres en gang pr. node.*

Bevis: Alle nodene som har samme farge som en nabo, kan utføre R_1. De nodene som har ledige farger å velge, kan i tillegg utføre R_2. □
Vi kan som en konklusjon si at R_1 og R_2 ikke utføres mer enn totalt $\mathcal{O}(n)$ ganger.

Lemma 5.2.2 *Dersom en node x utfører en regel som er avhengig av at en av dens naboer skal utføre R_1 eller R_2, kan den høyst utføre denne regelen $deg(x)$ ganger.*

Bevis: Reglene R_1 og R_2 kan kun utføres en gang pr. node, og x har $deg(x)$ naboer. □
Summerer en opp alle gradtallene i grafen, får vi $\sum_{i=1}^{n} deg(i) = 2m$, der $m = |E|$. Totalt i hele grafen blir dette $\mathcal{O}(m)$. Fra lemma 5.2.2 følger det at en regel som er avhengig av at naboen utfører enten R_1 eller R_2, ikke kan utføres mer enn $\mathcal{O}(m)$ ganger totalt i hele grafen.

Lemma 5.2.3 *Dersom noden x er en node som utfører en regel fordi en nabos nabo utfører R_1 eller R_2, kan den gjøre dette $2 * deg_2(x)$ ganger, der $deg_2(x)$ gir antall naboer en node har på avstand 2.*

Bevis: Vi har fra lemma 5.2.1 at det kun kan utføres $\mathcal{O}(n)$ R_1 og R_2 regler. Hver node x har bare $deg_2(x)$ naboer på avstand 2. Alle naboens naboer kan bare utføre R_1 og R_2 én gang. Dermed blir det $2 * deg_2(x)$.□
Summeres dette over hele grafen får vi at $\sum_{i=1}^{n} deg_2(i) = 2m$. Ved å definere $\delta_2 = \frac{\sum_{i=1}^{n} deg_2(i)}{n}$, vil det totalt for hele grafen ikke bli utført mer enn $n * \delta_2$[1] slike handlinger. Dvs. δ_2 er gjennomsnittlig antall naboer en node har på avstand 2.

[1] $\bar{\delta}$ er vanlig gjennomsnitt, se seksjon 2.1 side 7.

Vi skal nå vise hvor mange regler fra R_3 til R_6 som kan bli utført før en R_7 regel må bli utført. Til slutt vil vi begrense det totale antall ganger en R_7 regel kan bli utført.

Vi begynner med reglene R_3 og R_4. Reglene R_3 og R_4 alternerer på hver node. Avhengig av initialiseringen kan algoritmen starte med enten R_3 eller R_4. Når en node x peker på en annen node y (x utfører R_3), kan det skje to ting:

- Enten kan naboen y ta ned flagget.
  - a. Flagget kan enten bli tatt ned ved regel R_6.
  - b. Eller flagget blir tatt ned av regel R_7.
- Eller en annen nabo w av x kan ha byttet til samme farge som y. Dermed er ikke noden som x peker på lenger unik. Naboen w kan bytte farge ved to regler:
  - c. Naboen w bytter farge med R_2.
  - d. Naboen w bytter farge med R_7.

Det er nå 4 ulike punkt ($a,\ b,\ c,\ d$) som kan skje etter at en node har utført R_3. Vi må nå vise for alle disse punktene hvor mange R_3 og R_4 som kan bli utført. Vi begynner med punkt c.

Punkt c: Vi har fra lemma 5.2.2 at totalt antall R_3 og R_4 som drives av R_2, er $\mathcal{O}(m)$. Dette begrenser antall R_3 og R_4 som utføres, ved at en nabo utfører R_2.

Punktene $a,\ b$ og d: Hvis R_3 og R_4 skal utføres mer enn $\mathcal{O}(m)$ ganger, må de drives videre av enten R_6 eller R_7. Dette gir oss følgende lemma:

Lemma 5.2.4 *Det blir ikke utført flere enn $\mathcal{O}(m + \#[2]\ R_6 + \#\ R_7)$ R_3 og R_4 regler.*

Bevis: Reglene R_3 og R_4 kan utføres $\mathcal{O}(m)$ ved c, og vi har antall R_6 og R_7 fra a, b og d.□
Vi har nå begrenset antall R_3 og R_4, men denne begrensningen er avhengig av R_6 og R_7. Nå skal vi begrense antall ganger R_6 utføres. Det er to tilfeller der regelen R_6 kan utføres.

A. Regelen R_6 utføres ved start (fordi flagget er feil).

[2] # betyr antall. Dvs. # R_6 = antall ganger R_6 utføres

B. Eller R_6 utføres fordi y ikke har ledige farger lenger.

Punkt B kan bare skje dersom en nabo q til y tar fargen y hadde ledig. Dette kan skje hvis q utfører en av følgende regler:

C. Enten bytter noden q farge med R_2.

D. Eller så bytter q farge med R_7.

Dette gir oss tre punkt, nemlig punktene A, C og D (C og D er underpunkt til B). Dermed må vi begrense antall R_6 som utføres ved hvert punkt.

Punkt A: En utførsel av R_6 ved start kan bare skje en gang pr. node, og dette blir totalt $\mathcal{O}(n)$ R_6 handlinger.

Punkt C: Utførsel av R_6 motivert ved at en nabo til y utfører R_2 kan bare skje en gang pr. nabo til y. Lemma 5.2.2 gir at R_6 kan drives videre totalt $\mathcal{O}(m)$ ganger. Vi har nå vist punktene A og C. Punktet D er avhengig av antall R_7. Det gir oss følgende lemma for R_6.

Lemma 5.2.5 *Det kan ikke bli utført mer enn* $\mathcal{O}(n + m + \#R_7)$ R_6 *regler.*

Bevis: Fra A har vi $\mathcal{O}(n)$ en gang pr. node. Ved C og lemma 5.2.2 har vi $\mathcal{O}(m)$. Til slutt har vi punkt D som gir oss like mange ganger som R_7.□
Vi har nå vist at reglene R_3, R_4 og R_6 er avhengig av antall ganger R_7 utføres. Siden R_5 veksler med enten R_6 eller R_7, får vi følgende lemma:

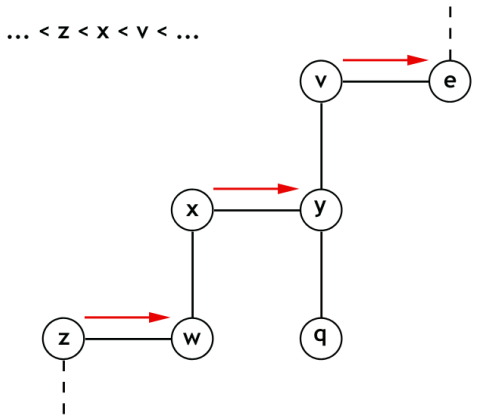
Lemma 5.2.6 *Regelen* R_5 *kan ikke utføres flere ganger enn* $\mathcal{O}(n + m + \#R_7)$.

Kommentar til lemma 5.2.6: Regelen R_5 kan utføres etter en nabo utfører R_1. Vi har fra lemma 5.2.2 at dette ikke kan gjøres mer enn $\mathcal{O}(m)$ ganger.

Vi har nå vist at reglene R_3 - R_6 er avhengige av antall ganger R_7 utføres. Det som gjenstår er å begrense antall R_7 som utføres. Regelen R_7 kan påvirke alle naboer opp til avstand 2. Vi ser først på hvor mange R_7 som kan utføres, før det må skje en R_2.

Når R_7 utføres første gang (la oss si på e, se figur 5.15), og ingen skal klare å fange den frigjorte fargen (med en R_2), må i hvert fall en nabo y til den noden v med lavest id som peker på e utføre R_7 og dermed stjele fargen fra v.

For at y skal kunne stjele fargen til v med R_7, må y ha en nabo x som har lavere id enn v, og den peker på y. Dette argumentet kan videreføres, og vi kan få en sekvens av R_7 utført på noder med avstand 2 imellom seg hvor de mellomliggende nodene må ha synkende id. En slik sekvens kan maksimalt bestå av $\mathcal{O}(n)$ R_7 handlinger (det kan ikke være flere



Figur 5.15: Utsnitt av en graf der noen noder peker på andre.

enn antall noder i grafen). Fargen som frigjøres av den siste R_7 handlingen, vil dermed være tilgjengelig for at en node skal kunne utføre R_2. Antall R_2 er $\mathcal{O}(n)$. Totalt gir dette $n * n = \mathcal{O}(n^2)$ R_7. Siden R_7 har en effekt både på den nærmeste naboen og på den to plasser lenger vekke, har vi totalt $\mathcal{O}((\delta_2 + \bar{\delta})\, n^2)$ handlinger. Dermed får vi følgende lemma for antall handlinger som R_7 kan utføre:

Lemma 5.2.7 *Regelen R_7 vil ikke utføre mer enn totalt $\mathcal{O}((\delta_2 + \bar{\delta})\, n^2)$ handlinger.*

Vi kan nå oppsummere følgende:

- Reglene R_1 og R_2 kan ikke utføres mer enn $\mathcal{O}(n)$ ganger ved lemma 5.2.1.
- Fra lemmaene 5.2.5 og 5.2.7 får vi at R_6 ikke vil utføre mer enn $\mathcal{O}(n + m + (\delta_2 + \bar{\delta})\, n^2)$ handlinger.
- Regelen R_5 kan ikke utføres mer enn $\mathcal{O}(n + m + (\delta_2 + \bar{\delta})\, n^2)$ ganger. Vi får dette fra lemmaene 5.2.6 og 5.2.7.
- Reglene R_3 og R_4 vil ikke utføre mer enn $\mathcal{O}(n + m + (\delta_2 + \bar{\delta})\, n^2)$ handlinger. Dette ved lemma 5.2.4, 5.2.5 og 5.2.7.

➪ Til slutt har vi at R_7 ikke utfører mer enn $\mathcal{O}((\delta_2 + \bar{\delta})\, n^2)$ handlinger.

Teorem 5.2.8 *PF–algoritmen vil ikke utføre mer enn $\mathcal{O}(m + (\delta_2 + \bar{\delta})\, n^2)$ handlinger.*

Bevis: Dette følger fra lemmaene 5.2.1, 5.2.4, 5.2.5, 5.2.6 og 5.2.7. □

Det er to faktorer som driver handlingstiden til PF–algoritmen. Det er antall kanter i grafen og antall noder. Hvis $m < (\delta_2 + \bar{\delta})n^2$, vil algoritmen drives av antall noder i grafen. Når $m > (\delta_2 + \bar{\delta})n^2$, vil handlingstiden drives av antall kanter i grafen. I kapittel 6 vil vi kjøre simuleringer med PF–algoritmen. Vi kan da sammenligne kjøretiden fra simuleringer med den asymptotiske kjøretiden som er vist her.

Kapittel 6

Simulering av frekvenstildeling

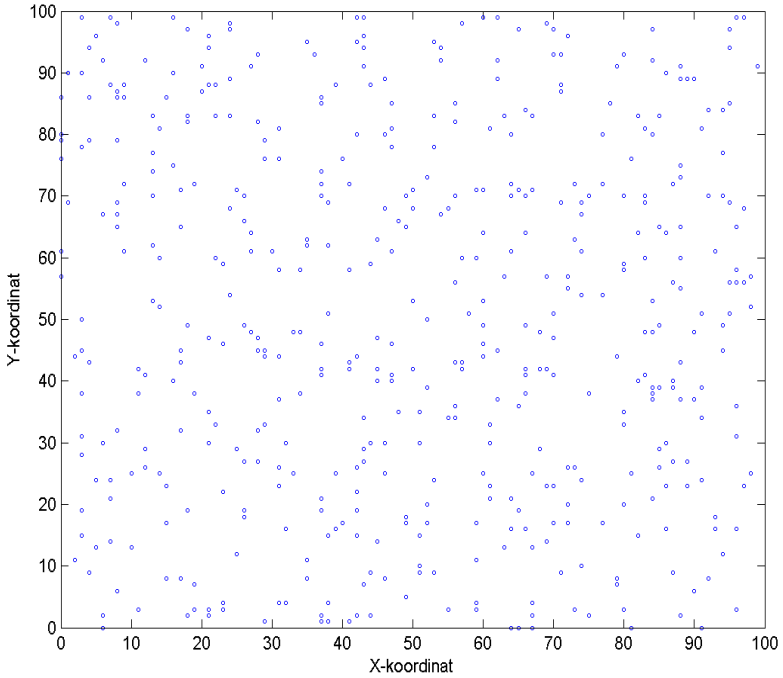
I dette kapitlet blir det presentert simuleringer med de ulike selvstabiliserende algoritmene fra kapittel 5. Vi ønsker å sammenligne disse for å se om det har noen hensikt å forhandle om fargeendring. I tillegg vil vi studere om den initielle tildelingen av farger påvirker utfallet til algoritmene.

Kapitlet gir først en oversikt over grafene det skal simuleres på. Disse er laget med grafgeneratoren fra kapittel 3. For å få en indikasjon på hvor mange farger en trenger til hver graf, blir de fargelagt med to ulike sekvensielle algoritmer.

Det blir kjørt simuleringer med ulikt antall tillatte farger for å se på hvordan fordelingen av antall ufargede noder blir påvirket av dette.

6.1 Datasettet som simuleringene kjøres på

Vi har generert grafer med 100 X-koordinater, 100 Y-koordinater og 500 noder. Alle nodene har de samme posisjonene i hver graf, dvs. node i har koordinat (x_i, y_i) i alle grafene. Det ble laget en graf for hver radius $d = \{2, 3, 4, ..., 9\}$, til sammen 8 grafer. For å se hvordan antall ufargede noder ble påvirket av antall tillatte farger, ble det kjørt simuleringer med antall tillatte farger (*maks*) satt til $2, 3, 4, 5, 7, 9, 11$ og 13. Til sammen 8 ulike $maks$ verdier. I figur 6.1 er alle de 500 nodene merket av i planet. I plottet kan en se at nodene er jevnt fordelt utover hele planet. I tabell 6.1 er det en oversikt over egenskapene til de 8 forskjellige grafene. En kan se at når radien øker, øker også det gjennomsnittlige gradtallet til alle



Figur 6.1: Plott av nodene i grafen som simuleringene kjøres på.

nodene, og det maksimale gradtallet går opp. Vi ser også antall farger som blir brukt ved de to sekvensielle algoritmene SDO og Første ledige farge (se tabell 6.2).

Radius	2	3	4	5	6	7	8	9
Gj.snitt gradtall	0.5	1.25	2.12	3.13	4.88	6.43	8.59	11.07
maks gradtall	4	7	9	11	12	15	18	19
Gj.snitt antall farger SDO:	4.0	5.0	6.5	8.4	9.0	10.5	11.4	12.7
Gj.snitt antall farger Første ledige:	4.0	5.0	6.3	8.4	9.2	10.4	11.6	12.6

Tabell 6.1: Data til grafene generert av grafgeneratoren.

Kommentar til grafene: Posisjonene til nodene i grafene er de samme for alle de 8 grafene. Det er kjørt simuleringer med andre posisjoner til nodene som bekrefter de resultatene vi presenterer her. SDO og Første ledige farge ble simulert 10 ganger der en valgte en tilfeldig node å starte med. Dermed fikk vi ulike resultater fra simuleringene og det er derfor ikke heltall i tabell 6.1.

I dette kapitlet vil det bli sammenlignet mange ulike algoritmer. Her kommer en oversikt over de ulike algoritmene.

Tabell 6.2: Oversikt over algoritmer som sammenlignes

Forkortelsen og navn på algoritmene		Måten fargen/noden velges på
Sekvensielle algoritmer		
SDO	Saturation Degree Ordering	Velger den noden med flest forskjellige farger på naboene (se seksjon 2.4.2 side 15).
FL	Første Ledige	Grådig algoritme som velger en tilfeldig node å fargelegge (se seksjon 2.4.2 side 14).
Selvstabiliserende algoritmer		
MRF	Modifisert Rask Fargelegging	Velger en tilfeldig ledig farge (se seksjon 5.1 side 32).
$MLLF$	Modifisert Laveste Ledige Farge	Velger den laveste ledige fargen (se seksjon 5.1 side 35).
TPF	Tilfeldig Peker Flagg	PF–algoritmen der den velger tilfeldig ledig farge (se seksjon 5.2 side 35).
$LLPF$	Laveste Ledige Peker Flagg	PF–algoritmen der den velger laveste ledige farge.

6.2 Sammenligning av initielle tildelinger

I denne delen skal vi se på den initielle tildelingen av verdier til nodene. Vil ulike tildelinger ved start ha en effekt på resultatet til slutt?

Vi ønsker å se både på tildelingen til den selvstabiliserende algoritmen som bruker flagg og til den som ikke bruker flagg. Først vil vi presentere 4 ulike initielle tildelinger med MRF–algoritmen ved 4 tillatte farger ($maks = 4$). Deretter blir TPF–algoritmen presentert med de samme tildelingene og like mange tillatte farger. De fire ulike tildelingene er:

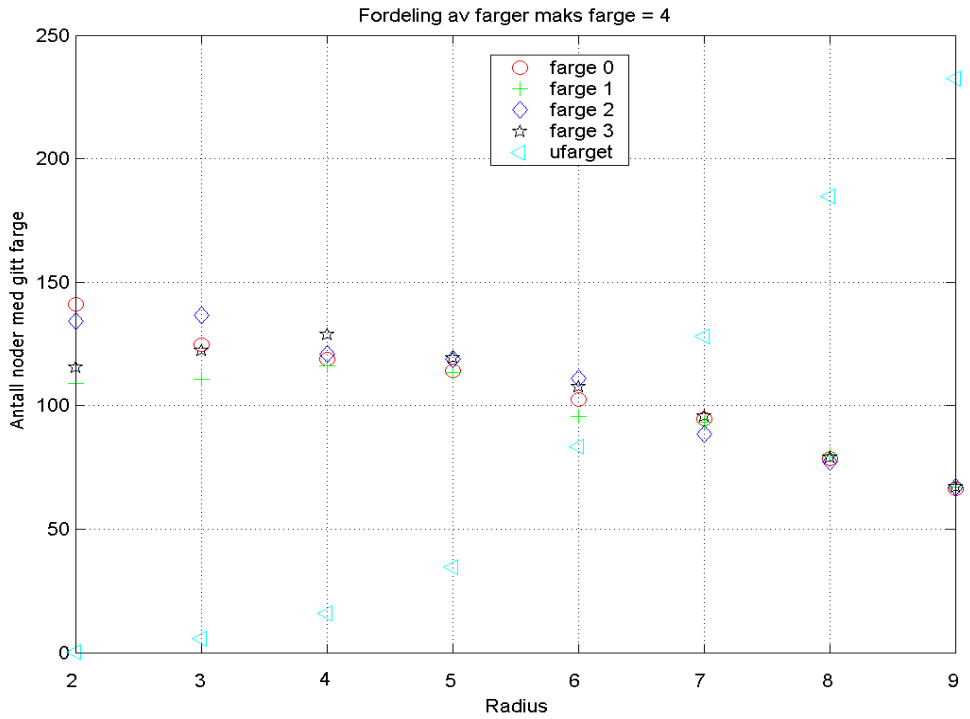
I. Uniform fargefordeling. Nodene fikk tildelt farger uniformt fra verdiene $\{0, 1, ..., (maks - 1)\}$ ved start.

II. Uniform fargefordeling inkludert -1 (ingen farge). Nodene ble tildelt verdier uniformt fra $\{0, 1, ..., (maks - 1)\} \cup \{ingenfarge\}$.

III. Alle nodene har samme farge ved start.

IV. Alle nodene har -1 fargen (ingen farge) ved start.

6.2.1 Sammenligning av initielle verdier til MRF–algoritmen

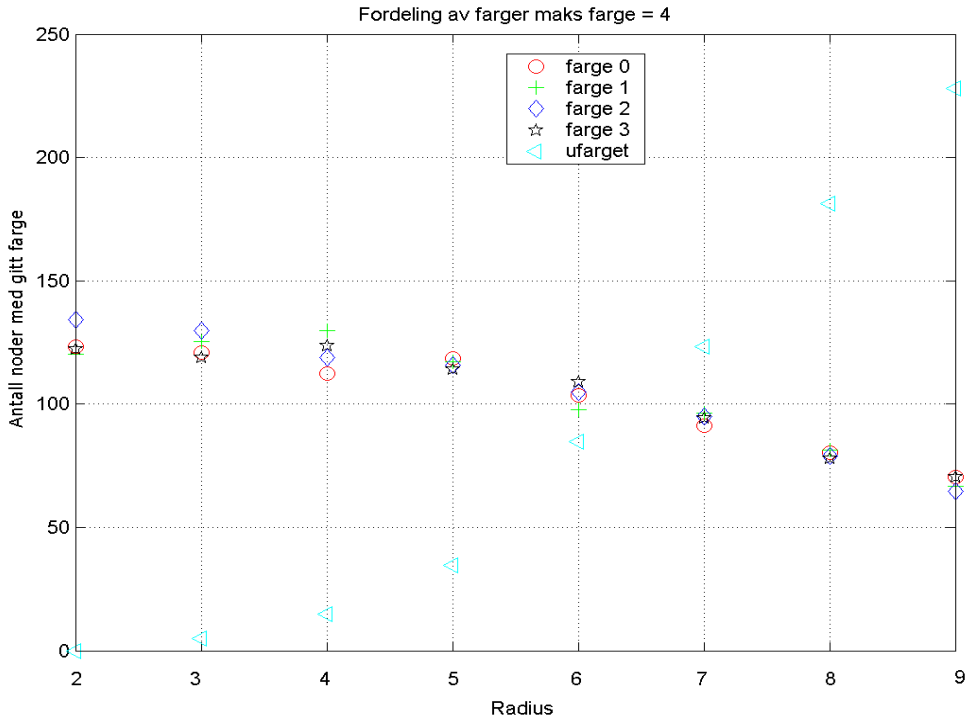


Figur 6.2: Uniform fargefordeling.

I) Uniform fargefordeling. I figur 6.2 har alle nodene i grafen blitt tildelt en av de fire forskjellige fargene. I tabell 6.3 ser en at når d er liten, er det veldig få regler som blir brukt; gjennomsnitt på 36,4 for $d = 2$. Det er to ting som gjør at det er få regler som blir utført når d er liten. Det ene er at gradtallet til hver node er lavt. Dermed er det ikke mange naboer som kan legge beslag på fargene. Det andre er at ved en uniform tildeling av fargene, er det sannsynlig at naboer blir tildelt forskjellige farger.

Radius	2	3	4	5	6	7	8	9
maks. uten farge	1	7	19	39	87	131	189	238
Gj.snitt uten farge	0.1	5.5	15.9	34.6	83.4	127.9	184.7	232.5
min. uten farge	0	4	13	31	78	124	180	225
Gj.snitt regler brukt	36.4	58.7	108.8	159.8	205.2	274.0	324.0	379.6

Tabell 6.3: Uniform fargefordeling.

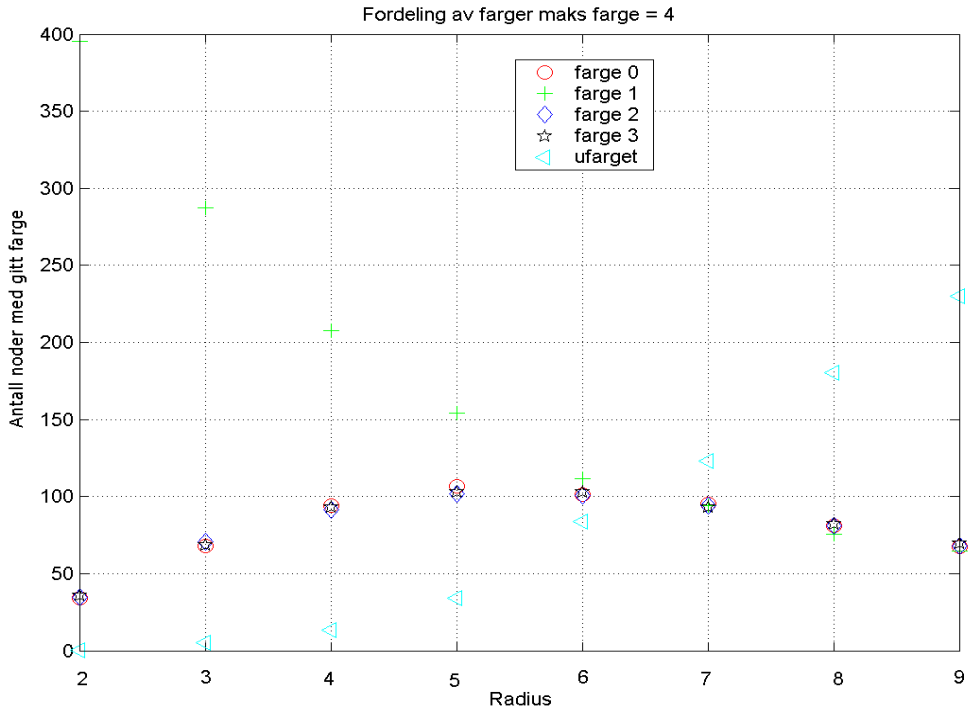


Figur 6.3: Uniform fargefordeling inkludert -1 (ingen farge).

II) Uniform fargefordeling inkludert -1 (ingen farge). Alle noder blir tildelt en farge fra utvalget {0, 1, 2, 3} ∪ {ingen farge} med en uniform sannsynlighetsfordeling. Her er det også få regler som blir brukt når d er liten, men antallet øker når d øker. Antall ufargede noder øker med d og er mellom 222 - 235 når $d = 9$. Når d går mot 9, går antall noder med de tillatte fargene fra 110 - 140 til mellom 60 - 80. Ved å inkludere -1 (ingen farge) i de uniforme tildelingene, er det flere regler som blir utført når d er liten (108,9 mot 36,4), mens når d øker er det færre regler som blir utført (291,1 mot 379,6). Når d er liten, er det få naboer rundt hver node. De nodene som får tildelt -1 (ingen farge) ved start, kan da endre til en lovlig farge.

Radius	2	3	4	5	6	7	8	9
maks. uten farge	0	6	18	39	91	128	188	235
Gj.snitt uten farge	0.0	4.9	15.0	34.7	84.8	123.4	181.2	228.1
min. uten farge	0	4	13	28	80	119	177	222
Gj.snitt regler brukt	108.9	140.3	165.4	199.1	226.1	224.7	278.4	291.1

Tabell 6.4: Uniform fargefordeling inkludert -1 (ingen farge).



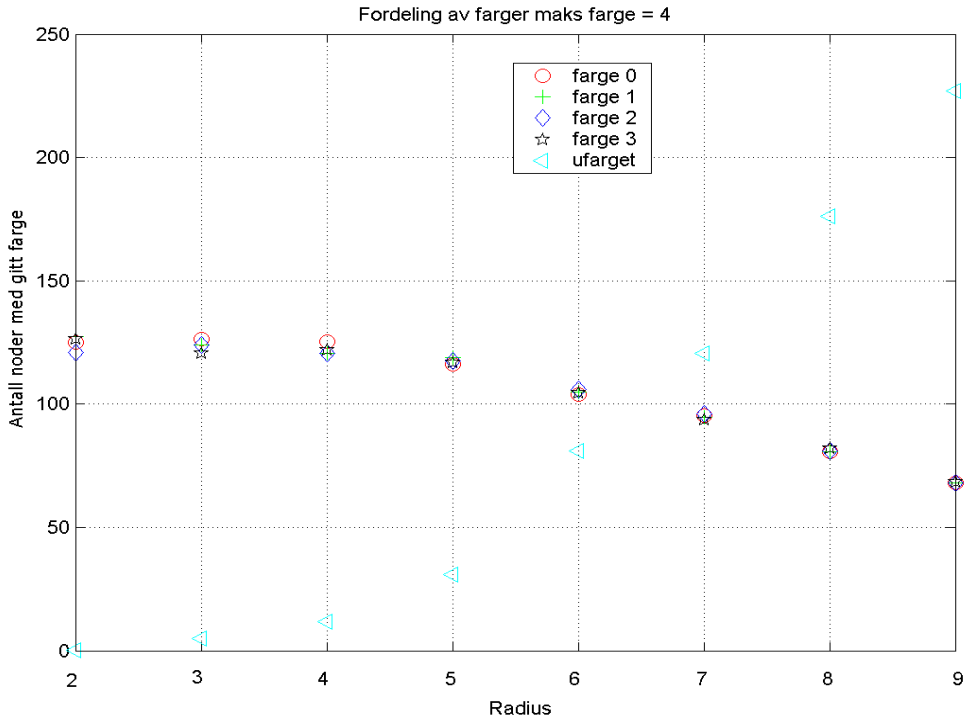
Figur 6.4: Alle nodene har den samme fargen.

III) Alle nodene har den samme fargen. I figur 6.4 har alle nodene den samme fargen ved start. Dette gir utslag i fordelingen av farger etter en kjøring. Når d er liten, er det mange noder som har den samme fargen som de hadde ved start (ca 400), mens når d øker, blir fargene fordelt jevnt mellom de 4, og det er et stort antall ufargede noder (gj.snitt 230,0). Dette støttes også av tallene i tabell 6.5.

Radius	2	3	4	5	6	7	8	9
maks. uten farge	1	7	15	39	89	134	185	236
Gj.snitt uten farge	0.2	5.3	13.1	34.0	83.7	123.2	180.2	230.0
min. uten farge	0	4	10	29	79	117	176	222
Gj.snitt regler brukt	104.5	213.5	294.7	351.6	412.2	435.1	461.6	475.7

Tabell 6.5: Alle nodene har den samme fargen.

IV) Alle nodene har -1 fargen. I figur 6.5 har alle nodene farge -1 (ingen farge) ved start. Det som er interessant her, er at når d er liten, er gjennomsnittet på antall regler brukt høyt. F.eks kan en se i tabell 6.6 at når $d = 2$, er gjennomsnittlig antall regler brukt 499,9.



Figur 6.5: Alle nodene har -1 fargen ved start.

Dvs. at nesten alle nodene utfører en regel. Når d øker, går gjennomsnittlig antall regler ned. I alle de tidligere tabellene (6.3, 6.4, 6.5) øker antall regler utført, når d blir større.

Radius	2	3	4	5	6	7	8	9
maks. uten farge	1	6	15	37	86	130	184	232
Gj.snitt uten farge	0.1	5.0	11.7	30.7	81.0	120.6	176.0	227.1
min. uten farge	0	4	9	23	77	109	168	221
Gj.snitt regler brukt	499.9	495.0	488.3	469.3	419.0	379.4	324.0	272.9

Tabell 6.6: Alle nodene har -1 fargen.

6.2.2 Diskusjon av initielle tildelinger til MRF–algoritmen

Det som skulle undersøkes med disse simuleringene, var om verdiene nodene ble tildelt ved start, hadde noe å si på utfallet til simuleringene. Den første simuleringen var uniform fargefordeling. Antall noder gitt -1 (ingen farge) gikk fra 0 til 233,5 når d gikk fra 2 til 9. Ser vi på de tillatte fargene, går disse fra mellom 141,2 og 115,6 ved $d = 2$ til mellom

66,5 og 67,1 noder pr. farge ved $d = 9$. Det er en større differanse på antall noder pr. farge når d er liten enn når d er stor. Andre simuleringer har vist at differansen mellom antall noder pr. farge er forskjellig for hver simulering. Når det er simulert på de samme grafene flere ganger, blir resultatene ulike. Det samme gjelder også for grafer der nodene har andre plasseringer.

Gjennomsnittlig antall regler brukt ved liten d, er markant mindre enn ved alle de 3 andre simuleringene. Denne har 36,4, og de andre har henholdsvis 108,9, 104,5, og 499,9. Når d er liten, er denne tildelingen å foretrekke med hensyn på antall regler som blir brukt.

Ved å ta med fargen -1 (ingen farge) i den uniforme fargetildelingen, øker gjennomsnittlig antall regler fra 36,4 til 108,9 ved liten d. Her øker også gjennomsnittlig antall regler utført, når d øker, fra 108,9 til 291,1. Antall noder som har -1 fargen (ingen farge) går fra 0 til 228,1. De tillatte fargene ligger her i mellom 120,2 og 134,1 ved $d = 2$. Vi ser at det er noen mindre spredning ved liten d her enn ved den første simuleringen. Differansen mellom antall noder pr. tillatte farge minker med økende d. Ved $d = 9$ ligger fargene i intervallet 66,4 til 70,3. Dette så vi også i den første simuleringen.

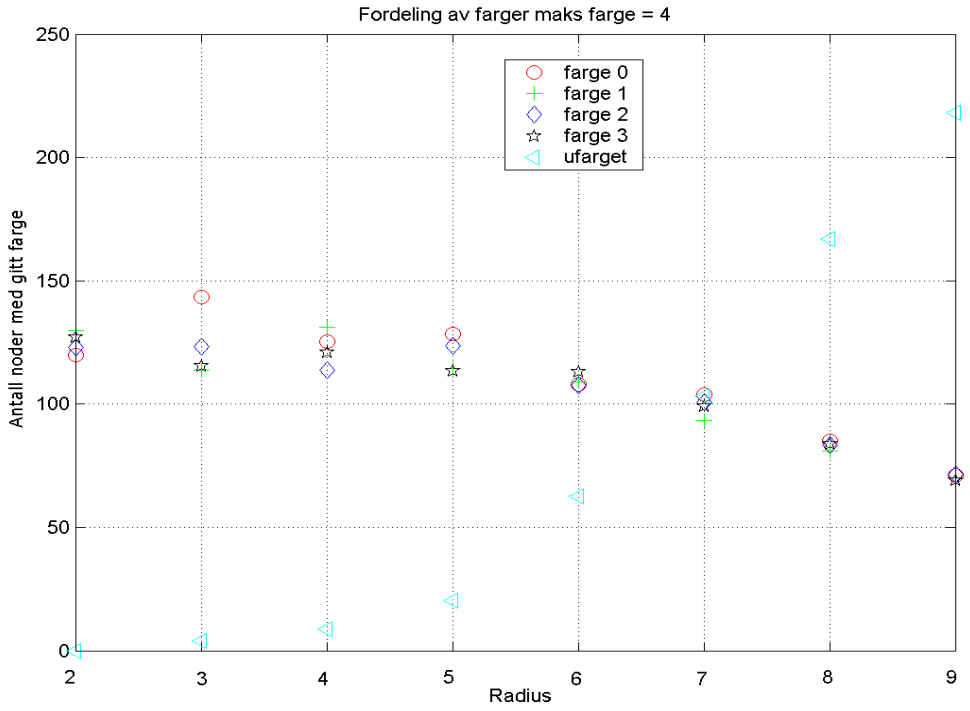
I den 3. simuleringen, der alle nodene har samme farge ved start, går antall noder som har -1 fargen (ingen farge) fra 0 til 222 - 236. Gjennomsnittlig antall regler brukt går fra 104,5 til 475,7, og øker når d øker. I denne simuleringen var det mange noder som beholdt den fargen de ble tildelt ved start, når d var liten. Ved $d = 2$ er det i gjennomsnitt 395,5 noder som har farge 1. Antall noder med fargen 1 blir det samme som de andre tillatte ved $d = 7$. Da ligger antall noder pr. tillatte farge rundt 95.

Den siste simuleringen var tildelingen der alle nodene hadde -1 fargen (ingen farge) ved start. Her er det en forskjell fra alle de andre simuleringene. Gjennomsnittlig antall regler brukt, minker når d øker. Ved $d = 2$ er det 499,9 regler i snitt som blir utført, mens når $d = 9$ er snittet nede i 272,9. Dette er en (markant) forskjell fra de 3 andre simuleringene. De hadde henholdsvis 379,6, 291,1 og 475,7. Her som i de 3 andre tildelingene, øker antall noder uten farge med økende d. De går fra 0,0 til 217,2 i gjennomsnitt. Antall noder pr. tillatte farge ligger mellom 120,1 og 131,9 ved liten d, mens når d blir stor, samler antall noder pr. tillatte farge seg rundt 70.

Vi kan som en konklusjon si at fordelingen av antall noder med -1 fargen ikke påvirkes

av den initielle fordelingen. Ser vi på antall handlinger som blir utført, kan vi trekke følgende konklusjon. Fordelen med uniform fordeling av fargene ved liten d, er at gjennomsnittlig antall regler brukt blir mindre enn når alle noder er ufarget ved start. Når d er stor, er det en fordel at alle noder har fargen -1 (ingen farge) ved start.

6.2.3 Sammenligning av initielle verdier til TPF–algoritmen



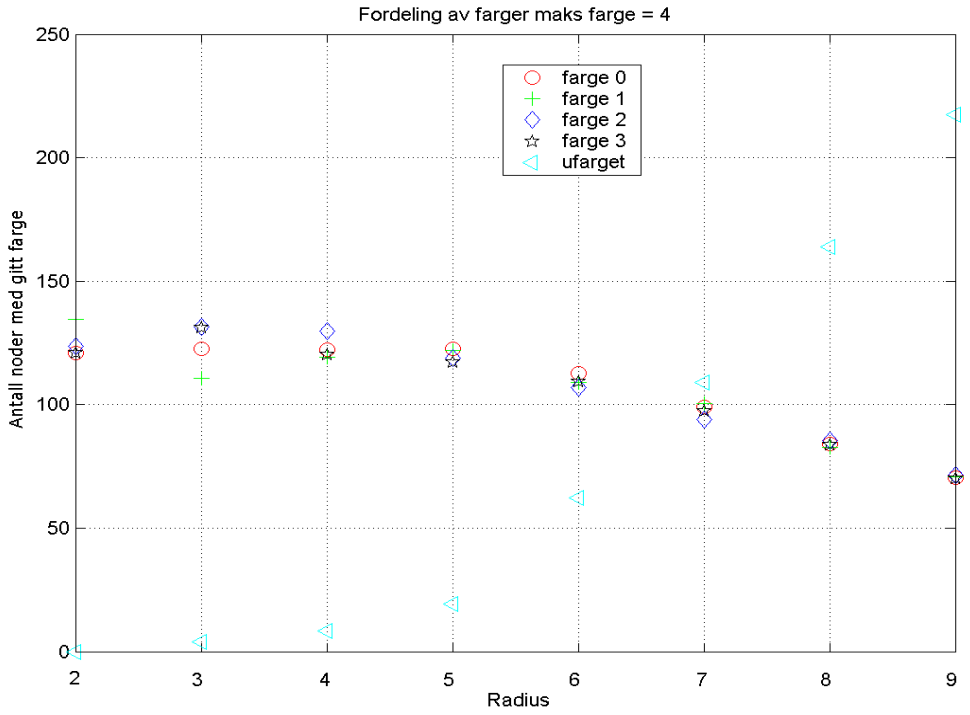
Figur 6.6: Uniform fargefordeling.

I) Uniform fargefordeling. Alle nodene ble tildelt en av de 4 tillatte fargene med en uniform fordeling. Gjennomsnittlig antall ufargede noder øker når d øker. Det går fra 0 til 218,9. Reglene øker opp mot $d = 6$, og deretter minker de ned mot $d = 9$. Antall regler utført er størst ved $d = 6$. Gjennomsnittet er da 808,3.

Når $d = 3$, er forskjellen i antall noder pr. farge 27,7 noder fra største til minste (143,3 - 115,6). For mer informasjon om fordelingen av noder til hver farge se tabell B.13, side 79.

Radius	2	3	4	5	6	7	8	9
maks. uten farge	0	4	9	22	66	107	175	222
Gj.snitt uten farge	0.0	4.0	8.6	20.2	62.5	103.2	166.9	218.2
min. uten farge	0	4	8	19	58	99	159	213
Gj.snitt regler brukt	568.2	630.3	699.4	750.0	808.3	732.2	651.6	608.2

Tabell 6.7: Uniform fargefordeling.

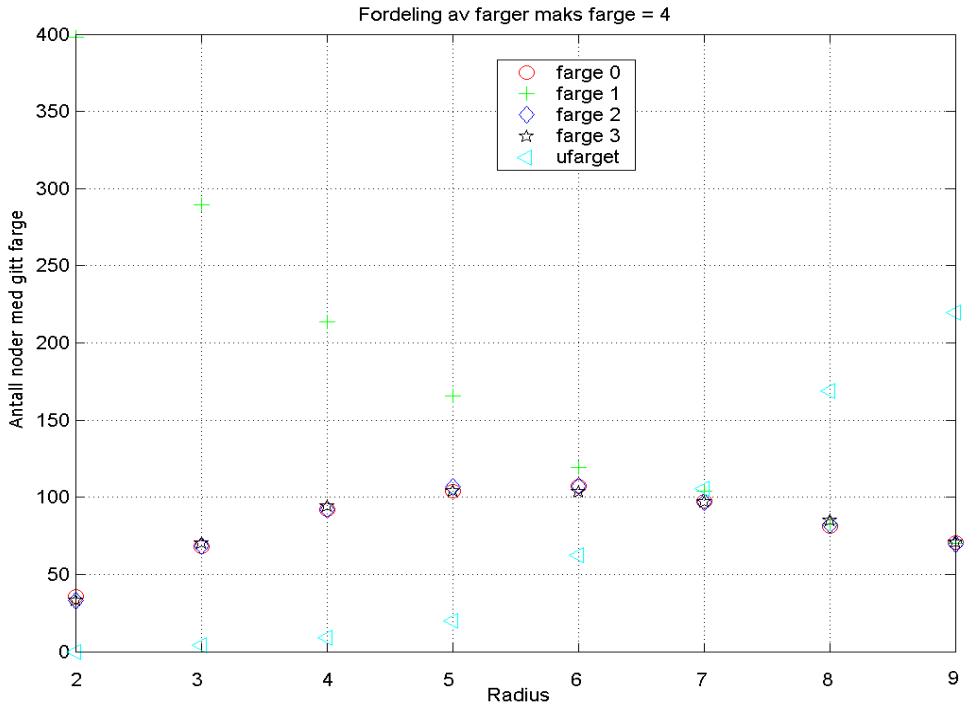


Figur 6.7: Uniform fargefordeling inkludert -1 (ingen farge).

II) Uniform fargefordeling inkludert -1 (ingen farge). Nodene ble her tildelt farger uniformt fra utvalget $\{0, 1, 2, 3\} \cup \{\text{ingen farge}\}$. Antall regler brukt er her 631,8 ved $d = 2$. Ved økende d går gjennomsnittlig antall ufargede noder fra 0 til 217,4. Gjennomsnittet til reglene øker opp til $d = 6$ for deretter å avta opp til 9. Når $d = 0$, brukes det i gjennomsnitt 631,8 regler. Ved $d = 6$ er gjennomsnittlig antall regler brukt oppe i 770,5. Det går så ned til 555,4 når $d = 9$.

Radius	2	3	4	5	6	7	8	9
maks. uten farge	0	4	10	21	65	115	167	222
Gj.snitt uten farge	0.0	4.0	8.4	19.3	62.2	108.9	164.0	217.4
min. uten farge	0	4	8	18	59	105	161	213
Gj.snitt regler brukt	631.8	655.2	717.3	727.7	770.5	652.6	619.3	555.4

Tabell 6.8: Uniform fargefordeling inkludert -1 (ingen farge).

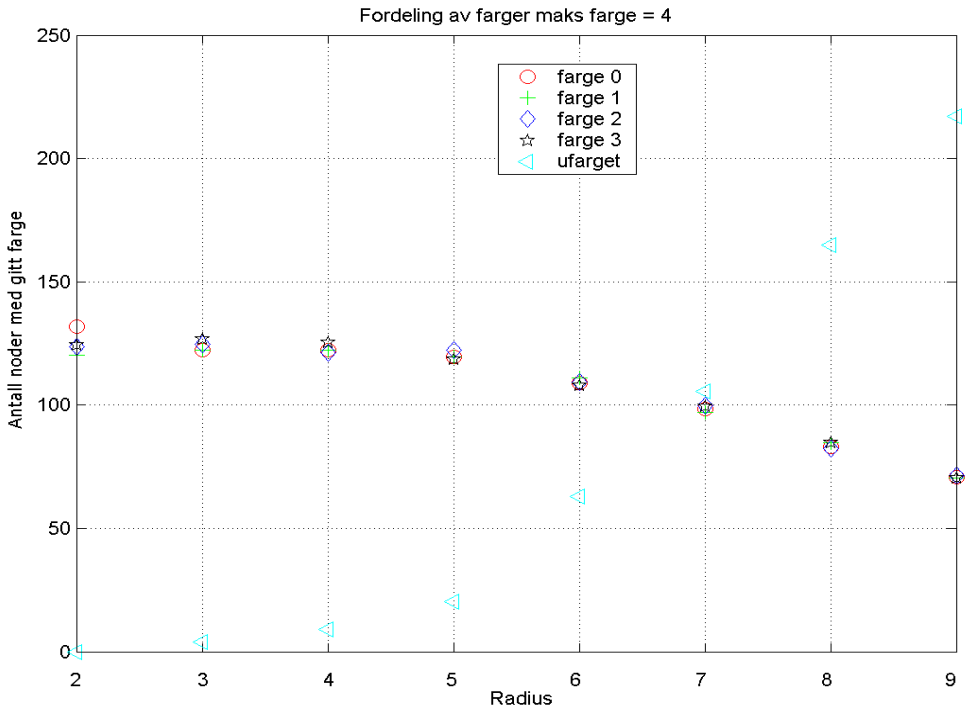


Figur 6.8: Alle nodene har den samme fargen.

III) Alle nodene har den samme fargen. Alle nodene har den samme fargen ved start i figur 6.8. Når d er liten, er det mange noder som beholder fargen de fikk ved start. Ved $d = 2$ er det i snitt 398,1 noder som har farge 1 (den fargen de fikk ved start). Denne tildelingen bruker flere regler enn de to foregående. Denne bruker 705,0 relger når $d = 2$ mot 631,8 og 568,2. Akkurat som de to tidligere algoritmene øker antall regler opp mot $d = 6$. Da bruker denne 1164,8 i snitt. Den går noe ned når $d = 9$ til 928,9.

Radius	2	3	4	5	6	7	8	9
maks. uten farge	0	4	10	22	64	108	176	224
Gj.snitt uten farge	0.0	4.0	8.8	19.7	62.6	105.4	169.1	219.4
min. uten farge	0	4	8	18	60	99	162	212
Gj.snitt regler brukt	705.0	919.6	1060.6	1150.6	1164.8	1120.0	999.9	928.9

Tabell 6.9: Alle nodene har den samme fargen.



Figur 6.9: Alle nodene har -1 fargen ved start.

IV) Alle nodene har -1 fargen. I figur 6.9 er alle nodene ufarget ved start. Når $d = 2$, bruker algoritmen gjennomsnittlig 1001,0 regler. Antall noder er 500. Når algoritmen bruker 1000 regler, tilsvarer dette 2 regler pr. node. Alle reglene velger en ledig farge (R_2), og alle reglene har mulighet til å bytte farge. De tar dermed opp flagget (R_5). Når d er stor, er gjennomsnittlig antall regler halvert til 507,2.

Antall ufargede noder går fra 0,0 til 217,2 i gjennomsnitt. Ved liten d ligger de tillatte fargene i mellom 131,9 - 120,1, mens når d er stor ligger alle de 4 tillatte fargene rundt 70.

Radius	2	3	4	5	6	7	8	9
maks. uten farge	0	4	10	22	66	109	172	224
Gj.snitt uten farge	0.0	4.0	8.9	20.2	62.8	105.6	164.8	217.2
min. uten farge	0	4	8	19	56	100	155	212
Gj.snitt regler brukt	1001.0	991.1	980.1	960.8	866.1	769.0	609.6	507.2

Tabell 6.10: Alle nodene har -1 fargen.

6.2.4 Diskusjon av initielle fordelinger til TPF–algoritmen

Vi har sett 4 ulike initielle tildelinger av farger ved start til TPF–algoritmen. Alle tildelingene har gjennomsnittlig antall ufargede noder fra 0 til rundt 218 (218,2, 217,4, 219,4, 217,4). Den første simuleringen som ble presentert, var uniform fordeling av fargene ved start. Antall noder pr. tillatte farge ligger mellom 120,0 og 129,7 ved $d = 2$. Når d øker mot 9, blir differansen mellom antall noder pr. tillatte farge mindre.

Når d er lav, bruker en uniform fordeling færre regler enn de tre andre. Denne har et gjennomsnitt på 568,2 mot de tre andre som har henholdsvis 631,8, 705,0 og 1001,0. Gjennomsnittlig antall regler som blir brukt ved uniform fargefordeling, øker med økende $d = 6$ (808,3). Deretter avtar det opp mot $d = 9$ (608,2). Når d er liten, er denne tildelingen å foretrekke med hensyn på antall regler som blir brukt.

Når en tar med -1 fargen i den uniforme fordelingen, øker gjennomsnittlig antall regler ved liten d fra 568,2 til 631,8. Gjennomsnittlig antall regler øker også her opp mot $d = 6$, men ikke like mye som ved uniform fordeling. Uniform fordeling inkl. -1 har en topp på 770,5 gjennomsnittlig antall regler utført. Dette er lavere enn alle de tre andre ved $d = 6$ (808,3 1164,8 og 866,1). Denne har også et lavere gjennomsnitt enn uniform tildeling når $d = 9$. Dette er da 555,4 mot 608,2. Intervallet til antall noder pr. tillatte farge er mellom 120,9 og 134,7 når $d = 2$. Dette intervallet minker til mellom 70,0 og 71,4 ved $d = 9$.

I den tredje simuleringen hadde alle nodene den samme fargen ved start. Her øker også antall regler som blir brukt opp mot $d = 6$. Det går fra 705,5 opp til 1164,8 ($d = 6$). Deretter går det ned til 928,9 ($d = 9$). Tildelingen der alle nodene har den samme fargen ved start hadde det høyeste av alle gjennomsnitt når $d = 6$ (1164,8).

Ved $d = 2$ er det i gjennomsnitt 398,1 noder som har den samme fargen før og etter en simulering. Dette tallet minker opp mot $d = 7$. Da har alle fargene rundt 100 noder hver,

både de tillatte fargene og ufargede. De tillatte fargene går så mot ca. 70 når d går mot 9.

Den siste simuleringen satt alle nodene til ufarget (ingen farge) ved start. Denne tildelingen hadde det høyeste gjennomsnittet ved liten d. Gjennomsnittlig antall regler utført var ved $d = 2$ 1001,0 mot 568,2, 631,8 og 705,0. I motsetning til alle de andre simuleringene minker dette fra $d = 2$ til $d = 9$. De andre økte fra 2 til 6 for deretter å avta opp mot 9.

Fordelingen av noder pr. tillatte farge ligger her, som i de to første simuleringene, mellom 120,1 og 131,9. Når d øker mot 9, blir differansen mellom de ulike fargene mindre og det er noe færre noder pr. farge (ca. 70 noder pr. farge).

Vi kan her trekke den samme konklusjonen som i diskusjon 6.2.2. Fordelingen av antall noder uten farge blir ikke påvirket av den initielle fordelingen av farger.

Fordelen med uniform fordeling av fargene ved lav d, er at gjennomsnittlig antall regler brukt blir mindre enn når alle nodene er ufarget ved start. Når d er stor, er det en fordel at alle nodene er ufarget (ingen farge) ved start. I tillegg er det færrest regler som blir brukt når d ligger mellom 5 og 7 ved en uniform fordeling inkludert -1 (ingen farge).

Handlingstiden Alle disse simuleringene ble kjørt med $n = 500$. Vi har fra seksjon 5.2.5 at PF–algoritmen har $\mathcal{O}((m + (\delta_2 + \bar{\delta})\, n^2)$ handlingstid. Den største handlingstiden simuleringene hadde, var i gjennomsnitt 1164,8. Dette var ved $d = 6$ og alle nodene hadde den samme fargen ved start. Når $d = 6$, er $\bar{\delta} = 4,88$ (se tabell 6.1). Dette skulle tilsvare ca 1250 kanter, dvs. $m = 1250$. Vi har her en kjøretid som er lavere enn m selv om $(\delta_2 + \bar{\delta})n^2$ er større enn m. Handlingstiden til PF–algoritmen tar høyde for det verste tenkelige tilfellet. Ut i fra simuleringene som er kjørt, ser det ikke ut til at verste tilfelle forekommer ofte.

6.3 Sekvensiell fargelegging

Det ble implementert to sekvensielle fargeleggingsalgoritmer. Dette ble gjort for å sammenligne dem med de selvstabiliserende algoritmene. De to sekvensielle algoritmene er Første ledige farge og SDO (Saturation Degree Ordering). Første ledige farge er algoritme 2.4.1 på side 14. Dette er en grådig algoritme som velger den laveste ledige fargen. SDO er om-

talt i seksjon 2.4.2 på side 15, og denne velger å fargelegge den noden med flest fargede naboer, der hver farge telles en gang.

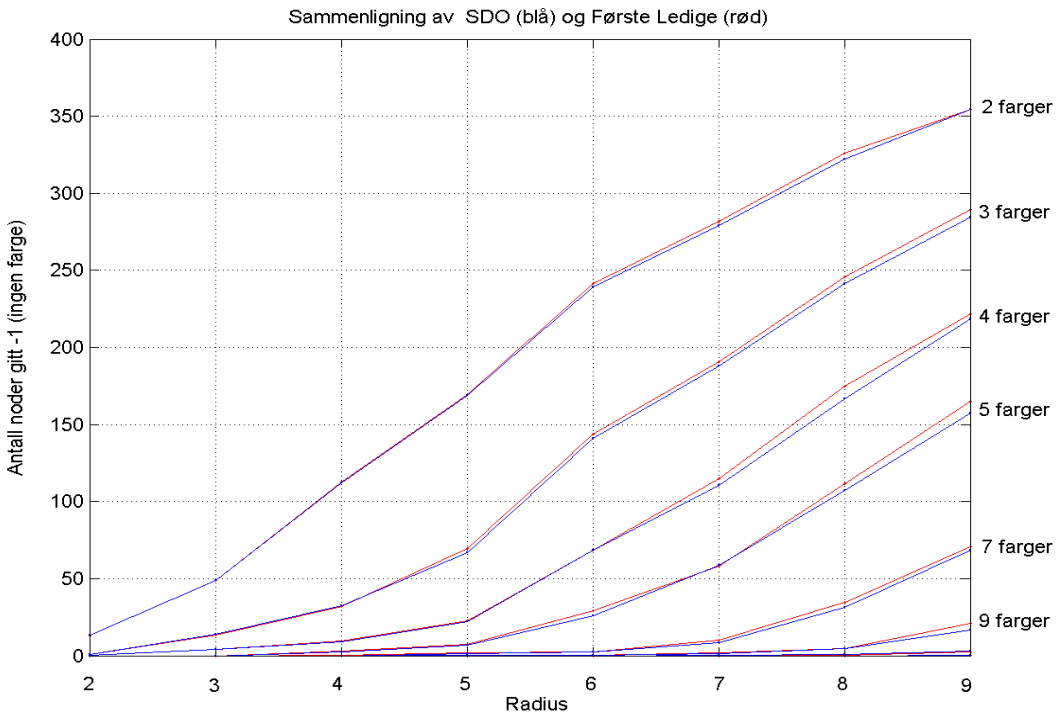
For å begrense antall sammenligninger som presenteres, vil SDO bli sammenlignet med Første ledige. Den beste av disse to blir sammenlignet med den beste av de nye selvstabiliserende algoritmene. De to selvstabiliserende algoritmene er like bortsett fra at de velger farge på ulike måter.

➪ PF–algoritmen (Peker Flagg–algoritmen).

I. Algoritmen velger tilfeldig ledig farge, TPF.

II. Algoritmen velger laveste ledige farge, $LLPF$.

6.3.1 Sammenligning av SDO og Første ledige



Figur 6.10: Sammenligning av SDO og Første ledige.

I figur 6.10 er de to sekvensielle algoritmene sammenlignet. Det er kjørt simuleringer med 8 ulike $maks$–verdier på de 8 grafene fra tabell 6.1. SDO–algoritmen er den blå linjen,

mens Første ledige er den røde linjen. I plottet kan en se at den blå linjen ligger likt eller like under den røde linjen. Dette tilsier at SDO–algoritmen er like bra eller bedre enn Første ledige. Ser vi nærmere på tallene, kan vi trekke frem følgende. Det var 31 tilfeller der SDO var best, mens det var 11 tilfeller der Første ledige var best. I de resterende 22 tilfellene var differansen mellom dem 0,0.

Den største absolutte differansen til SDO er 8,6 noder lavere enn Første ledige ($175,1 - 166,5$, ved $maks = 4$ og $d = 7$). Motsatt hadde Første ledige den største absolutte differansen på 0,5 noder bedre enn SDO ($13,3 - 13,8$, ved $maks = 3$ og $d = 3$). Disse verdiene er hentet fra tabellene B.5 og B.6.

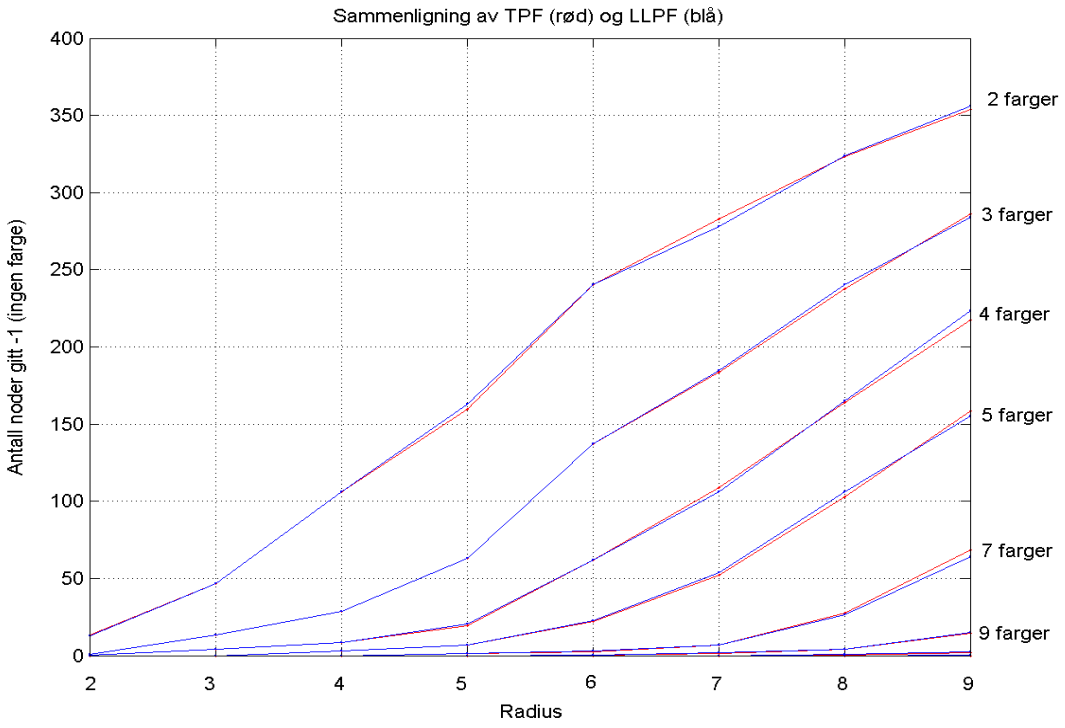
I snitt ligger SDO 1,13 noder bedre enn Første ledige. Ut i fra disse tallene og som vi ser i plottet, er det ikke stor forskjell på de to sekvensielle algoritmene. Siden SDO var litt bedre enn Første ledige, velger vi denne når vi skal sammenligne med de selvstabiliserende algoritmene.

6.3.2 Sammenligning av TPF og $LLPF$

I denne delen har vi sammenlignet to ulike måter PF–algoritmen velger farge på. I figur 6.11 ser vi sammenligningen av én initiell fordeling (uniform fordeling inkludert -1 fargen) på de 8 ulike $maks$–verdiene og grafene fra tabell 6.1. I figur 6.11 er det tilfeller der TPF er lavere enn $LLPF$, og det er tilfeller der det er motsatt. For å kunne skille mellom de to forskjellige måtene å velge farge på, må vi se nærmere på tallene til de ulike simuleringene.

Som det ble nevnt tidligere er det 8 ulike grafer. For hver graf ble det kjørt 10 simuleringer. Det ble simulert med 8 forskjellige antall tillatte farger. For hver farge ble det brukt 4 ulike initielle tildelinger til nodene. Dette blir til sammen $8 * 10 * 8 * 4 = 2560$ simuleringer for hver av de 4 ulike algoritmene. For de 10 simuleringene pr. graf lages det et gjennomsnitt.

Sammenligner vi de 256 gjennomsnittsverdiene til $LLPF$ med TPF får vi følgende resultat. $LLPF$ var best i 82 tilfeller (hadde laveste gjennomsnitt), mens TPF var best i 62. I de resterende tilfellene hadde de en differanse på 0,0. Dvs. det var 113 tilfeller der de var like. Grunnen til at det var mange tilfeller der de to var like, er at når en bruker mange farger på en graf med lav radius, blir det ingen noder som er ufarget (har -1 fargen).



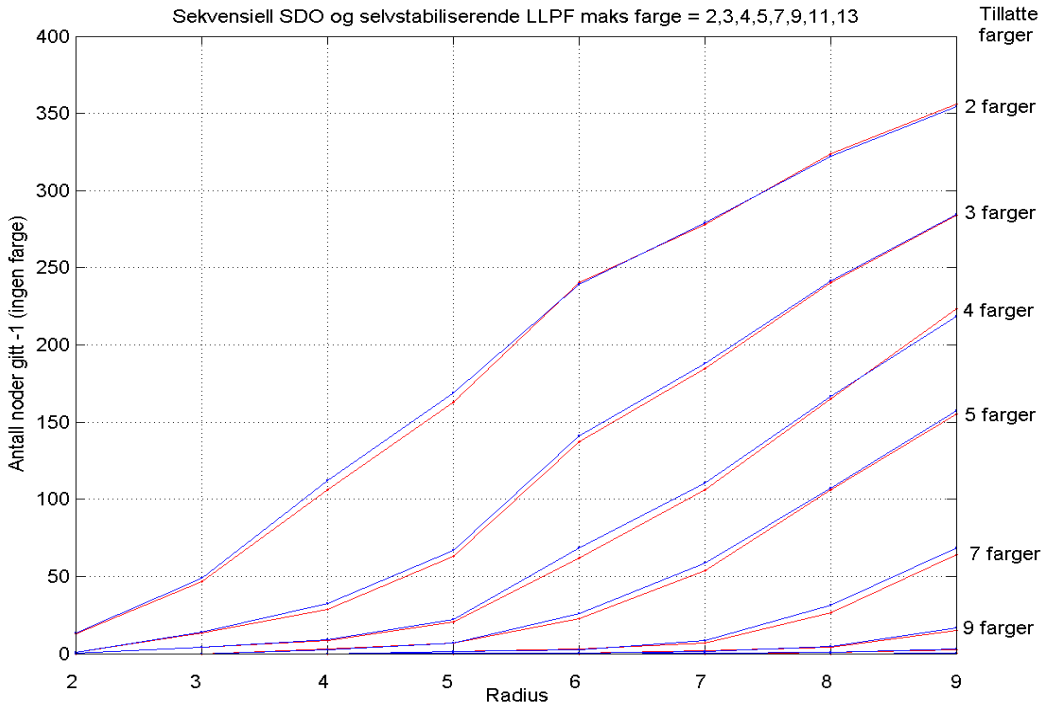
Figur 6.11: Sammenligning av TPF og $LLPF$.

Dermed får begge algoritmene gjennomsnittlig 0,0 ufargede noder. Siden det var flest tilfeller der $LLPF$–algoritmen var best, vil vi bruke denne videre når vi skal sammenligne med sekvensielle algoritmer.

6.3.3 Sammenligning av SDO og $LLPF$

Vi har nå vist at den sekvensielle SDO–algoritmen er litt bedre enn Første ledige, og den selvstabiliserende $LLPF$–algoritmen er noe bedre enn TPF. I denne delen skal vi sammenligne SDO med $LLPF$.

Vi så i seksjon 6.2 at de ulike initielle tildelingene ikke hadde stor betydning for antall ufargede noder. Det ble derfor valgt en tilfeldig initiell fordeling til $LLPF$ som var alle noder -1 fargen ved start. Dette ble gjort siden SDO ikke har noen initiell fordeling. Vi får derfor 64 ulike kombinasjoner som blir sammenlignet. I figur 6.12 kan vi se at SDO–linjen ligger over $LLPF$ når det er to tillatte farger og radien er 9. Det er her SDO er best. I absolutt verdi er den laveste verdien til SDO 3,1 færre ufargede noder enn $LLPF$



Figur 6.12: SDO sammenlignet med $LLPF$.

($322.2 - 325.3$). $LLPF$ har den største differansen med 10,3 færre ufargede noder enn SDO ($58.4 - 48.1$) ved 5 tillatte farger og $d = 7$.

Til sammen er det 5 tilfeller der SDO er bedre enn $LLPF$, mens det er 36 tilfeller der $LLPF$ er bedre enn SDO. Utenom dette er differansen mellom dem 0,0 i 36 tilfeller. Det er da enten ingen ufargede noder eller de har et gjennomsnitt som er det samme.

6.3.4 Diskusjon

Først ble de sekvensielle algoritmene SDO og Første ledige sammenlignet. Vi så at forskjellen mellom dem var liten, men SDO var litt bedre enn Første ledige. Deretter sammenlignet vi de selvstabiliserende algoritmene TPF og $LLPF$. Igjen var det ikke store differansen mellom de to vi sammenlignet, men $LLPF$ hadde flest tilfeller med lavere gjennomsnitt enn TPF.

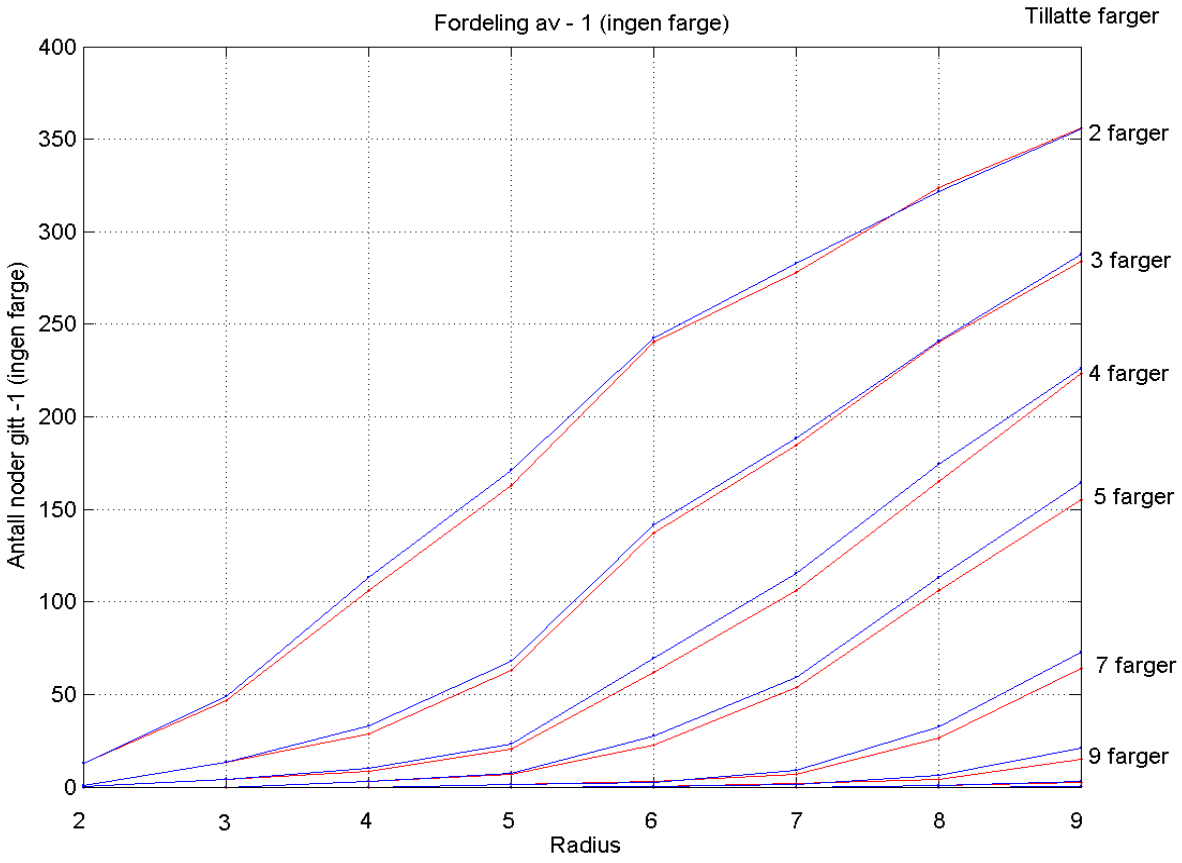
Vi valgte derfor å sammenligne $LLPF$ med SDO. Her hadde SDO 5 tilfeller der den var bedre enn $LLPF$. Gjennomsnittet til disse 5 tilfellene var 1,36. Dvs. at det i snitt

var 1,36 færre noder enn $LLPF$. I de 36 tilfellene der $LLPF$ var bedre enn SDO var gjennomsnittet på 2,84 noder.

6.4 Antall ufargede noder

I forrige seksjon ble PF–algoritmen sammenlignet med SDO. I denne delen vil $LLPF$ bli sammenlignet med den modifiserte raske fargeleggingsalgoritmen (MRF) og $MLLF$.

6.4.1 Sammenligning av $LLPF$–algoritmen med MRF og $MLLF$



Figur 6.13: Sammenligning av $LLPF$ og $MLLF$ algoritmene.

Vi har her sammenlignet $LLPF$ med de to modifiserte selvstabiliserende algoritmene MRF og $MLLF$. I figur 6.13 er $LLPF$ sammenlignet med $MLLF$. Ser vi på plottet er det størst forskjell når radien ligger mellom 4 og 8 og antall farger ligger i intervallet 3 til 9.

Forbedringer er her fra 3 % til 30 %. Den gjennomsnittlige forbedringen ligger på 9,81 %. Dvs. $LLPF$ ligger 9,81 % lavere enn $MLLF$ i dette intervallet. Når det er 9 tillatte farger og d ligger i mellom 6 og 9, er MRF i snitt 280,78 % over $LLPF$ i gjennomsnittlig antall ufargede noder. I samme intervall er $MLLF$ i snitt 36,46 % over $LLPF$.

	$LLPF$	MRF	$MLLF$	Alle 3 lik 0	$LLPF$ = $MLLF$ og > 0	Totalt
færrest ufargede	144	0	13	67	32	256
Prosent	56,25 %	0,0	5,07 %	26,2 %	12,5 %	100 %
færrest ufargede	144	0	13		32	189 (256 - 67)
Prosent	76,19 %	0,0	6,88 %		16,93 %	100 %

Tabell 6.11: Sammenligning av lavest ledige peker flagg og de to andre.

Tabellen 6.11 sammenligner $LLPF$ med MRF og $MLLF$. $LLPF$ hadde gjennomsnittlig færrest fargede noder i 144 tilfeller av totalt 256, og det er 56,25 %. Alle algoritmene hadde også her gjennomsnitt lik null 67 ganger (26,2 %). Dersom en ser bort i fra disse, var $LLPF$ best i 76,19 % av de totalt 189 gjennomsnittene. Av de 189 tilfellene var det 32 tilfeller der $LLPF$ hadde den samme gjennomsnittsverdien som $MLLF$ og denne var større enn null. Dette er 16,93 %. $MLLF$–algoritmen var best i 6,88 %.

Det var 13 tilfeller der $MLLF$ var bedre enn $LLPF$. Av disse 13, var det 2 tilfeller der differansen var mer enn 2 % lavere. I disse 2 tilfellene var differansen på gjennomsnittene 0,1 og 0,2 noder i favør av $MLLF$. Det beste gjennomsnittet til $MLLF$ hadde 2,7 færre noder enn $LLPF$ $(325,3 - 322,6)$, Dette var 0,83 % bedre enn $LLPF$.

6.4.2 Diskusjon

Det har nå blitt presentert sammenligninger av $MLLF$, MRF og $LLPF$. $LLPF$ hadde flest tilfeller der gjennomsnittlig antall ufargede noder var lavere enn MRF og $MLLF$. De gangene $LLPF$ ikke var best var differansen ned til den beste liten.

Når tillatte farger var lik 9 og d lå mellom 6 og 9, hadde $LLPF$ i snitt 36,46 % lavere verdier enn $MLLF$. Det er i dette intervallet vi har den beste forbedringen til $LLPF$ i forhold til $MLLF$.

Vi kan som en konklusjon si at $LLPF$ er like god eller bedre enn MRF og $MLLF$. I tillegg er $LLPF$ noe bedre enn TPF, da den hadde flere gjennomsnittsverdier som var

lavere enn TPF.

Kapittel 7

Oppsummering

Dette kapitlet er en oppsummering av oppgaven. Kapitlet inneholder konklusjoner som svarer på spørsmålene gitt i innledningen og forklarer oppnådde resultater. Til slutt er det listet opp noen punkter en kan arbeide videre med.

7.1 Konklusjon

Det var 3 spørsmål som ble nevnt i introduksjonen til denne oppgaven. Her kommer det en oppsummering av svarene til de 3 spørsmålene.

I. Antall enheter som kan kommunisere gitt k frekvenser. I seksjon 6.4 ble de ulike algoritmene tildelt k tillatte frekvenser. Det ble simulert tildeling av frekvenser på ulike grafer med k satt til forskjellige verdier. Simuleringsresultatene viste at når radien til grafene økte, økte antall enheter uten frekvenser. Antall enheter som hadde frekvenser økte også når k økte. Det som er hensiktsmessig å velge kommer ann på hva en ønsker å oppnå. Hvis alle enhetene skal kunne kommunisere samtidig er det en fordel med lav sendeeffekt (liten radius) hvis det er få frekvenser tilgjengelig.

II. Betydningen av den initielle tildelingen. Det ble kjørt simuleringer med 4 ulike initielle tildelinger av farger/frekvenser. To ulike algoritmer ble benyttet. Begge algoritmene velger farger tilfeldig. Resultatet ble omtrent det samme for de to når en sammenlingner gjennomsnittlig antall regler brukt. Det var kun forskjell i antall handlinger som ble utført,

og ikke i gjennomsnittlig antall ufargede noder. Ved en liten radius var det en fordel å velge uniform fordeling av fargene, mens ved stor radius ble det brukt minst handlinger ved at alle nodene var ufarget initielt.

III. Forespørsel om fargeendring. Det ble presentert en ny selvstabiliserende fargeleggingsalgoritme som har mulighet til å forhandle om fargeendring. Simuleringer viste at denne algoritmen var like bra eller bedre enn de enkle fargeleggingsalgoritmene. Det beste resultatet ble oppnådd når den nye algoritmen valgte laveste, ledige farge. Når det var 9 tillatte farger og radien til nodene lå i mellom 6 og 9, hadde $LLPF$–algoritmen (Laveste Ledige Peker Flagg) i snitt 36,46 % færre antall ufargede noder enn $MLLF$ (Modifisert Laveste Ledige Farge). Dette resultatet var noe bedre enn når PF–algoritmen valgte tilfeldig ledig farge.

7.2 Videre arbeid

Det ble presentert en algoritme som har muligheter til å forespørre en nabo om fargeendring. En naturlig utvidelse av denne ville være å se om den kan forespørre flere naboer samtidig. Det er kanskje mulig med én peker pr. nabo. En kan da peke på alle som har den samme fargen i nabolaget.

Det er mulig å få tak i grafer som en vet det kromatiske tallet til. Det hadde vært interessant å se hvor bra de selvstabiliserende algoritmene ble opp mot den optimale fargeleggingen. Hvor mange ufargede noder blir det når en har tillatt å bruke like mange farger som det kromatiske tallet?

En annen ting som kan undersøkes er hvorfor det ser ut til å være bedre å velge den laveste ledige fargen og ikke tilfeldig. Vi så at det var en større forbedring når PF–algoritmen (Peker Flagg) valgte laveste ledige enn når den valgte tilfeldig.

Handlingstiden til PF–algoritmene tar høyde for at det skjer en kjede med R_7 etter hverandre. Dette er et verste tilfelle. Det kunne vært inetressant å finne ut sannsynligheten for at denne kjeden med R_7 oppstår. Dermed kunne en funnet den forventede handlingstiden til PF.

Tillegg A

Implementasjon av nye regler til simulatoren

For å kunne implementere nye selvstabiliserende algoritmer til simulatorprogrammet, må en gjøre følgende:

- Alle reglene som skal simuleres må implementere grensesnittet *rule.rules*. Dvs. i pakken *rule* er det et grensesnitt (interface) som heter *rules*. Dette må enhver regel implementere.

- I grensesnittet *system.demon* er det en metode som heter *setRulesToNode(system.graph graph, rule.rules rule)*. Denne metoden lager nye regelobjekter og legger et objekt til hver node. Her må man legge til den nye reglen. Dette gjøres med en if sats *if(rules.rule instanceof new_Rule)*. Det er implementert to ulike "demoner" en for MTR og $MLLF$ (*systemDemon.java*) og en for PF–algoritmen (*smartSystemDemon.java*). Der kan en se hvordan det blir gjort.

- Det må lages en klasse som kjører simuleringene. Den må vite hvor grafene som skal leses inn ligger, og hvor den skal skrive ut objektene som holder på data fra hver simulering. Klassen som leser inn grafer heter *graphs.makeGraphOne*. Denne klassen har ulike metoder for å lese inn grafer.

  Det er også laget to ulike kjøreklasser en kan bruke som eksempler: *ModifiedGrundyColoring.runModifiedGrundy* og *ModifiedGrundyColoring.runSmartAlg7rules*. Disse to bruker følgende metode for å lese inn en graf:
  readGraphFromFileBinary(String filename, system.demon dem,int[] colorDist,int maxcolor, boolean includeNoColor, boolean sameColorToAll)

- Det er så bare å implementere den nye selvstabiliserende algoritmen. Det anbefales på det sterkeste å laste ned Jbulider fra Borland. En kan gjøre det gratis ved å registrere seg som student. Dette er et utviklingsmiljø for Java, og den kommer både for Windows og Linux. Simulatorprogrammet skal takle både Windowskataloger og linux. Den siste versjonen er bare testet på Windows. Typisk sti i Windows *C:\Hovedfag\dataObjekt* i Linux er tilsvarende */Home/stud3/tomso/Hovedfag/dataObjekt/*.

Tillegg B

Data fra simuleringene

B.1 Antall noder ufarget ved de ulike algoritmene

Radius	2	3	4	5	6	7	8	9
Gj.snitt -1, 2 farger	15.5	55.2	118.3	173.2	244.2	281.0	324.7	359.8
Gj.snitt -1, 3 farger	3.1	15.8	40.9	84.3	154.7	199.3	250.8	293.1
Gj.snitt -1, 4 farger	0.0	4.9	15.0	34.7	84.8	123.4	181.2	228.1
Gj.snitt -1, 5 farger	0.0	0.5	4.2	13.0	40.6	72.0	123.9	177.1
Gj.snitt -1, 7 farger	0.0	0.0	0.2	2.6	6.3	23.5	50.0	89.4
Gj.snitt -1, 9 farger	0.0	0.0	0.0	0.3	0.7	4.1	14.2	36.6
Gj.snitt -1, 11 farger	0.0	0.0	0.0	0.0	0.2	1.2	1.7	10.0
Gj.snitt -1, 13 farger	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.7

Tabell B.1: Antall noder ufarget ved MRF–algoritmen og uniform fordeling inkl. -1.

Radius	2	3	4	5	6	7	8	9
Gj.snitt -1, 2 farger	13.2	46.6	106.2	159.5	240.3	282.8	323.5	354.1
Gj.snitt -1, 3 farger	1.0	13.4	28.6	62.8	137.0	183.6	237.5	286.0
Gj.snitt -1, 4 farger	0.0	4.0	8.4	19.3	62.2	108.9	164.0	217.4
Gj.snitt -1, 5 farger	0.0	0.0	2.9	6.9	22.0	52.3	102.7	158.5
Gj.snitt -1, 7 farger	0.0	0.0	0.0	1.6	2.6	6.8	27.4	68.5
Gj.snitt -1, 9 farger	0.0	0.0	0.0	0.0	0.3	1.5	3.9	14.2
Gj.snitt -1, 11 farger	0.0	0.0	0.0	0.0	0.0	0.0	0.2	2.0
Gj.snitt -1, 13 farger	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tabell B.2: Antall noder ufarget ved TPF–algoritmen og uniform fordeling inkl. -1.

Radius	2	3	4	5	6	7	8	9
Gj.snitt -1, 2 farger	12.7	49.0	113.1	171.0	242.5	282.7	321.7	355.5
Gj.snitt -1, 3 farger	1.0	13.6	32.9	67.8	141.8	188.7	240.9	288.0
Gj.snitt -1, 4 farger	0.0	4.0	10.1	23.1	69.7	115.2	174.1	226.1
Gj.snitt -1, 5 farger	0.0	0.0	2.8	7.5	27.4	59.3	113.1	164.3
Gj.snitt -1, 7 farger	0.0	0.0	0.0	1.4	2.7	9.0	32.2	72.9
Gj.snitt -1, 9 farger	0.0	0.0	0.0	0.0	0.2	1.5	6.2	21.0
Gj.snitt -1, 11 farger	0.0	0.0	0.0	0.0	0.0	0.0	0.6	3.2
Gj.snitt -1, 13 farger	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1

Tabell B.3: Antall noder ufarget ved $MLLF$–algoritmen og uniform fordeling inkl. -1.

Radius	2	3	4	5	6	7	8	9
Gj.snitt -1, 2 farger	12.6	46.4	106.0	163.0	240.2	277.8	323.9	355.9
Gj.snitt -1, 3 farger	1.0	13.2	28.8	62.8	137.3	184.6	240.6	284.1
Gj.snitt -1, 4 farger	0.0	4.0	8.3	20.6	62.0	106.0	165.2	223.7
Gj.snitt -1, 5 farger	0.0	0.0	2.8	6.9	22.5	53.9	106.1	155.1
Gj.snitt -1, 7 farger	0.0	0.0	0.0	1.1	2.8	6.8	26.5	64.2
Gj.snitt -1, 9 farger	0.0	0.0	0.0	0.0	0.2	1.7	4.3	14.9
Gj.snitt -1, 11 farger	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.2
Gj.snitt -1, 13 farger	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2

Tabell B.4: Antall noder ufarget ved $LLPF$–algoritmen og uniform fordeling inkl. -1.

Radius	2	3	4	5	6	7	8	9
Gj.snitt ufarget 2 farger:	13.1	48.9	111.9	168.8	239.4	279.0	322.2	354.3
Gj.snitt ufarget 3 farger:	1.0	13.8	32.2	66.8	140.8	188.2	241.6	284.8
Gj.snitt ufarget 4 farger:	0.0	4.2	9.0	22.1	68.6	110.6	166.5	218.4
Gj.snitt ufarget 5 farger:	0.0	0.0	2.5	6.9	25.7	58.4	107.1	157.5
Gj.snitt ufarget 7 farger:	0.0	0.0	0.0	1.4	2.6	8.7	31.4	68.6
Gj.snitt ufarget 9 farger:	0.0	0.0	0.0	0.1	0.1	1.5	4.7	16.8
Gj.snitt ufarget 11 farger:	0.0	0.0	0.0	0.0	0.0	0.1	0.6	3.0
Gj.snitt ufarget 13 farger:	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1

Tabell B.5: Antall noder ufarget ved sekvensiell SDO–algoritmen.

Radius	2	3	4	5	6	7	8	9
Gj.snitt ufarget 2 farger:	13.0	48.6	112.9	169.3	241.3	282.1	326.3	354.5
Gj.snitt ufarget 3 farger:	1.0	13.3	31.8	69.7	144.0	190.8	245.9	289.5
Gj.snitt ufarget 4 farger:	0.0	4.0	9.7	22.8	68.3	114.9	175.1	222.1
Gj.snitt ufarget 5 farger:	0.0	0.0	2.9	7.2	29.4	58.3	111.6	165.1
Gj.snitt ufarget 7 farger:	0.0	0.0	0.1	1.7	2.7	10.0	34.5	70.9
Gj.snitt ufarget 9 farger:	0.0	0.0	0.0	0.0	0.3	1.8	4.9	21.0
Gj.snitt ufarget 11 farger:	0.0	0.0	0.0	0.0	0.0	0.0	0.4	2.6
Gj.snitt ufarget 13 farger:	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1

Tabell B.6: Antall noder er ufarget ved sekvensiell Første ledige farge algoritmen.

B.2 Tabeller til sammenligning av initielle verdier MRF, kap 6.2.1, side 53

Radius	2	3	4	5	6	7	8	9
maks uten farge	1	7	19	39	87	131	189	238
Gj.snitt uten farge	0.1	5.5	15.9	34.6	83.4	127.9	184.7	232.5
min uten farge	0	4	13	31	78	124	180	225
gj.snitt antall noder gitt farge 0	141.2	124.7	118.7	114.1	102.6	94.7	78.6	66.5
gj.snitt antall noder gitt farge 1	108.8	110.7	116.0	113.5	95.6	93.2	80.4	66.9
gj.snitt antall noder gitt farge 2	134.3	136.7	120.8	118.7	110.9	88.6	77.2	67.1
gj.snitt antall noder gitt farge 3	115.6	122.4	128.6	119.1	107.5	95.6	79.1	67.0
gj.snitt regler brukt	36.4	58.7	108.8	159.8	205.2	274.0	324.0	379.6

Tabell B.7: Fordeling av farger til MRF–algoritmen ved uniform initiell fordeling.

Tabell B.7 tilhører plottet på side 53.

Radius	2	3	4	5	6	7	8	9
maks uten farge	1	7	15	39	89	134	185	236
Gj.snitt uten farge	0.2	5.3	13.1	34.0	83.7	123.2	180.2	230.0
min uten farge	0	4	10	29	79	117	176	222
gj.snitt antall noder gitt farge 0	33.9	68.2	94.1	106.8	101.5	95.3	81.1	67.6
gj.snitt antall noder gitt farge 1	395.5	287.3	207.9	154.4	111.4	94.3	75.6	64.9
gj.snitt antall noder gitt farge 2	34.7	70.5	91.6	101.7	100.7	93.9	80.9	68.0
gj.snitt antall noder gitt farge 3	35.7	68.7	93.3	103.1	102.7	93.3	82.2	69.5
gj.snitt regler brukt	104.5	213.5	294.7	351.6	412.2	435.1	461.6	475.7

Tabell B.8: Fordeling av farger til MRF–algoritmen ved den samme fargen til alle nodene initielt.

Tabell B.8 tilhører plottet på side 55.

Radius	2	3	4	5	6	7	8	9
maks uten farge	0	6	18	39	91	128	188	235
Gj.snitt uten farge	0.0	4.9	15.0	34.7	84.8	123.4	181.2	228.1
min uten farge	0	4	13	28	80	119	177	222
gj.snitt antall noder gitt farge 0	123.4	121.0	112.5	118.5	103.5	91.1	80.3	70.3
gj.snitt antall noder gitt farge 1	120.2	125.2	129.9	117.0	97.8	96.2	81.3	66.6
gj.snitt antall noder gitt farge 2	134.1	129.9	118.8	115.7	104.9	95.1	79.1	64.7
gj.snitt antall noder gitt farge 3	122.3	119.0	123.8	114.1	109.0	94.2	78.1	70.3
gj.snitt regler brukt	108.9	140.3	165.4	199.1	226.1	224.7	278.4	291.1

Tabell B.9: Fordeling av farger til MRF–algoritmen ved uniform fordeling inkl. -1 initielt.

Tabell B.9 tilhører plottet på side 54.

Radius	2	3	4	5	6	7	8	9
maks uten farge	1	6	15	37	86	130	184	232
Gj.snitt uten farge	0.1	5.0	11.7	30.7	81.0	120.6	176.0	227.1
min uten farge	0	4	9	23	77	109	168	221
gj.snitt antall noder gitt farge 0	124.9	126.5	125.2	116.0	103.8	95.3	80.5	68.1
gj.snitt antall noder gitt farge 1	127.5	123.9	120.6	118.9	104.9	94.5	80.6	68.2
gj.snitt antall noder gitt farge 2	121.0	124.1	120.6	117.6	105.8	96.1	81.0	68.2
gj.snitt antall noder gitt farge 3	126.5	120.5	121.9	116.8	104.5	93.5	81.9	68.4
gj.snitt regler brukt	499.9	495.0	488.3	469.3	419.0	379.4	324.0	272.9

Tabell B.10: Fordeling av farger til MRF–algoritmen ved alle noder ufarget initielt.

Tabell B.10 tilhører plottet på side 56.

B.3 Tabeller til sammenligning av initielle verdier TPF, kap. 6.2.3, side 58

Radius	2	3	4	5	6	7	8	9
maks uten farge	0	4	9	22	66	107	175	222
Gj.snitt uten farge	0.0	4.0	8.6	20.2	62.5	103.2	166.9	218.2
min uten farge	0	4	8	19	58	99	159	213
gj.snitt antall noder gitt farge 0	120.0	143.3	125.4	128.3	107.8	103.8	85.2	71.0
gj.snitt antall noder gitt farge 1	129.7	113.8	131.3	114.5	109.0	93.2	80.9	70.1
gj.snitt antall noder gitt farge 2	123.1	123.3	113.7	123.6	107.6	100.8	83.4	71.5
gj.snitt antall noder gitt farge 3	127.2	115.6	121.0	113.4	113.1	99.0	83.6	69.2
gj.snitt regler brukt	568.2	630.3	699.4	750.0	808.3	732.2	651.6	608.2

Tabell B.11: Fordeling av farger til TPF–algoritmen ved uniform fordeling initielt.

Tabell B.11 tilhører plottet på side 58.

Radius	2	3	4	5	6	7	8	9
maks uten farge	0	4	10	22	64	108	176	224
Gj.snitt uten farge	0.0	4.0	8.8	19.7	62.6	105.4	169.1	219.4
min uten farge	0	4	8	18	60	99	162	212
gj.snitt antall noder gitt farge 0	35.7	68.1	92.0	103.8	107.5	97.2	81.1	70.5
gj.snitt antall noder gitt farge 1	398.1	289.5	213.5	165.7	119.3	103.8	82.9	70.1
gj.snitt antall noder gitt farge 2	33.2	68.4	91.8	106.6	107.2	97.0	82.0	69.6
gj.snitt antall noder gitt farge 3	33.0	70.0	93.9	104.2	103.4	96.6	84.9	70.4
gj.snitt regler brukt	705.0	919.6	1060.6	1150.6	1164.8	1120.0	999.9	928.9

Tabell B.12: Fordeling av farger til TPF–algoritmen ved den samme fargen til alle nodene initielt.

Tabell B.12 tilhører plottet på side 60.

Radius	2	3	4	5	6	7	8	9
maks uten farge	0	4	10	21	65	115	167	222
Gj.snitt uten farge	0.0	4.0	8.4	19.3	62.2	108.9	164.0	217.4
min uten farge	0	4	8	18	59	105	161	213
gj.snitt antall noder gitt farge 0	120.9	122.7	122.2	122.7	112.6	99.2	84.1	70.5
gj.snitt antall noder gitt farge 1	134.7	110.8	119.2	122.1	108.9	100.5	82.6	70.7
gj.snitt antall noder gitt farge 2	123.5	131.5	129.9	118.8	107.0	94.1	85.5	71.4
gj.snitt antall noder gitt farge 3	120.9	131.0	120.3	117.1	109.3	97.3	83.8	70.0
gj.snitt regler brukt	631.8	655.2	717.3	727.7	770.5	652.6	619.3	555.4

Tabell B.13: Fordeling av farger til TPF–algoritmen ved uniform fordeling inkl. -1 initielt.

Tabell B.13 tilhører plottet på side 59.

Radius	2	3	4	5	6	7	8	9
maks uten farge	0	4	10	22	66	109	172	224
Gj.snitt uten farge	0.0	4.0	8.9	20.2	62.8	105.6	164.8	217.2
min uten farge	0	4	8	19	56	100	155	212
gj.snitt antall noder gitt farge 0	131.9	122.4	122.2	119.5	109.1	98.3	83.2	70.8
gj.snitt antall noder gitt farge 1	120.1	122.4	122.3	119.5	110.9	96.9	84.8	70.4
gj.snitt antall noder gitt farge 2	123.7	124.5	121.3	122.3	109.3	99.7	82.5	71.3
gj.snitt antall noder gitt farge 3	124.3	126.7	125.3	118.5	107.9	99.5	84.7	70.3
gj.snitt regler brukt	1001.0	991.1	980.1	960.8	866.1	769.0	609.6	507.2

Tabell B.14: Fordeling av farger til TPF–algoritmen ved alle noder ufarget initielt.

Tabell B.14 tilhører plottet på side 61.

B.4 Simuleringer med TPF–algoritmen.

Forklaring til tabellen:

- **Radius** Den gjeldende radiusen på grafen det er simulert på.
- **tpf** Tilfeldig peker flagg–algoritmen.
- **gj.snitt** Gjennomsnitt for gjeldene algoritme.
- **%** Antall noder i % i forhold til **tpf** algoritmen. Dvs. tpf–algoritmen = 100 %.
- **mrf** Modifisert tilfeldig rask fargelegging.
- **mllf** Modifisert lavest ledige fargelegging.
- **tpsf** Standardavviket til gjennomsnittet for tpf–algoritmen.
- **mtrs** Standardavviket til gjennomsnittet for mtr–algoritmen.
- **mllfs** Standardavviket til gjennomsnittet for mllf–algoritmen.
- s Standardavvik.
- gj.snitt Gjennomsnittsverdi.
- prosent Antall i prosent der tpf = 100 %.
- Initiell fordeling
  - tt: Ingen noder har blitt tildelt farge ved start.
  - tf: Uniform fordeling av fargene ved start inkl. -1 (ingen farge).
  - ft: Alle noder har samme farge ved start.
  - ff: Uniform fordeling av fargene ved start.

Tabell B.15: Tabell der TPF–algortimen velger tilfeldig, ledig farge.

Radius	**tpf**	**gj.snitt**	**%**	**mrf**	**gj.snitt**	**%**	**mllf**	**gj.snitt**	**%**
Tillatte farger:	2	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.51	12.4		1.19	14.1	113.7	0.69	12.6	101.61
3	1.56	46.3		1.44	53.1	114.68	2.4	49.7	107.34
4	3.1	107.5		2.49	117.3	109.11	2.05	114.0	106.04
5	3.98	162.9		2.98	171.7	105.4	4.07	168.8	103.62
6	3.24	239.1		5.27	242.7	101.5	2.45	237.3	99.24
7	2.79	279.6		3.46	281.7	100.75	5.11	283.8	101.5
8	2.44	322.8		3.83	327.4	101.42	3.47	322.6	99.93
9	3.89	356.5		3.59	355.7	99.77	4.0	355.3	99.66
Tillatte farger:	3	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	1.0		0.84	1.6	160.0	0.0	1.0	100.0
3	0.42	13.2		1.5	15.4	116.66	0.51	13.6	103.03
4	1.19	29.1		3.3	39.3	135.05	1.63	32.0	109.96
5	2.79	63.6		3.09	76.4	120.12	4.44	68.2	107.23
6	4.14	134.6		4.56	146.8	109.06	5.45	143.8	106.83
7	4.39	184.8		2.27	193.6	104.76	3.42	189.8	102.7
8	4.34	241.3		5.71	248.6	103.02	5.01	246.5	102.15
9	4.39	285.7		5.46	290.5	101.68	3.55	287.8	100.73
Tillatte farger:	4	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.31	0.1	-1.0	0.0	0.0	-1.0
3	0.0	4.0		0.81	5.0	125.0	0.31	4.1	102.5
4	0.73	8.9		1.88	11.7	131.46	0.78	8.8	98.87
5	0.91	20.2		4.37	30.7	151.98	1.68	22.8	112.87
6	2.78	62.8		2.86	81.0	128.98	2.52	72.2	114.96
7	3.09	105.6		6.56	120.6	114.2	3.87	113.1	107.1
8	5.8	164.8		5.07	176.0	106.79	5.32	175.1	106.25
9	3.99	217.2		3.6	227.1	104.55	4.29	221.4	101.93
Tillatte farger:	5	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.78	0.8	-1.0	0.0	0.0	-1.0
4	0.67	2.7		1.15	4.3	159.25	0.63	2.8	103.7
5	0.91	6.8		1.47	10.8	158.82	0.51	6.6	97.05
6	1.77	22.5		3.65	35.3	156.88	2.48	27.2	120.88
Fortsetter på neste side									

Tabell B.15 – fortsetter fra forrige side

Radius	**tpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
7	3.16	49.7		4.49	71.2	143.25	3.66	59.1	118.91
8	2.84	101.1		5.02	123.2	121.85	3.65	113.4	112.16
9	5.48	157.9		4.96	171.3	108.48	3.19	167.8	106.26
Tillatte farger:	7	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.52	0.5	-1.0	0.0	0.0	-1.0
5	0.31	1.1		0.73	2.1	190.9	0.51	1.4	127.27
6	0.51	2.4		1.5	5.6	233.33	0.51	2.4	100.0
7	1.56	8.0		3.28	17.1	213.75	1.49	10.3	128.75
8	2.28	27.1		2.44	44.8	165.31	3.33	32.6	120.29
9	3.94	62.6		5.42	84.5	134.98	3.09	71.6	114.37
Tillatte farger:	9	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
6	0.31	0.1		0.48	0.7	700.0	0.42	0.2	200.0
7	0.51	1.4		1.25	3.7	264.28	0.84	1.6	114.28
8	1.69	4.0		1.57	11.4	285.0	0.96	4.6	115.0
9	2.94	15.7		2.78	32.2	205.09	2.45	19.4	123.56
Tillatte farger:	11	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.67	0.7	-1.0	0.0	0.0	-1.0
8	0.82	0.7		0.94	2.3	328.57	0.48	0.3	42.85
9	0.94	2.0		1.56	8.7	435.0	0.96	2.6	130.0
Tillatte farger:	13	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0

Fortsetter på neste side

Tabell B.15 – fortsetter fra forrige side

Radius	**tpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
8	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
9	0.0	0.0		0.69	1.6	-1.0	0.0	0.0	-1.0
Tillatte farger:	2	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.91	13.2		0.84	15.5	117.42	0.48	12.7	96.21
3	1.57	46.6		2.25	55.2	118.45	0.81	49.0	105.15
4	2.09	106.2		2.83	118.3	111.39	3.63	113.1	106.49
5	4.03	159.5		3.55	173.2	108.58	3.52	171.0	107.21
6	4.11	240.3		4.15	244.2	101.62	4.14	242.5	100.91
7	2.48	282.8		2.86	281.0	99.36	3.09	282.7	99.96
8	2.22	323.5		3.56	324.7	100.37	3.3	321.7	99.44
9	2.13	354.1		4.1	359.8	101.6	2.87	355.5	100.39
Tillatte farger:	3	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	1.0		1.19	3.1	310.0	0.0	1.0	100.0
3	0.51	13.4		1.47	15.8	117.91	0.69	13.6	101.49
4	0.84	28.6		3.47	40.9	143.0	1.85	32.9	115.03
5	2.2	62.8		3.91	84.3	134.23	2.39	67.8	107.96
6	2.44	137.0		2.75	154.7	112.91	3.19	141.8	103.5
7	3.16	183.6		4.98	199.3	108.55	3.12	188.7	102.77
8	3.97	237.5		2.52	250.8	105.6	3.51	240.9	101.43
9	4.05	286.0		2.8	293.1	102.48	3.29	288.0	100.69
Tillatte farger:	4	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	4.0		0.56	4.9	122.5	0.0	4.0	100.0
4	0.69	8.4		1.56	15.0	178.57	1.37	10.1	120.23
5	1.15	19.3		3.52	34.7	179.79	1.85	23.1	119.68
6	1.93	62.2		4.04	84.8	136.33	1.88	69.7	112.05
7	3.54	108.9		3.13	123.4	113.31	3.55	115.2	105.78
8	2.05	164.0		3.91	181.2	110.48	4.86	174.1	106.15
9	3.27	217.4		3.98	228.1	104.92	4.84	226.1	104.0
Tillatte farger:	5	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.7	0.5	-1.0	0.0	0.0	-1.0
4	0.73	2.9		1.31	4.2	144.82	0.42	2.8	96.55
Fortsetter på neste side									

Tabell B.15 – fortsetter fra forrige side

Radius	**tpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
5	0.31	6.9		2.35	13.0	188.4	1.17	7.5	108.69
6	2.21	22.0		2.95	40.6	184.54	2.31	27.4	124.54
7	3.3	52.3		2.98	72.0	137.66	4.05	59.3	113.38
8	3.71	102.7		3.72	123.9	120.64	5.13	113.1	110.12
9	3.4	158.5		4.43	177.1	111.73	5.16	164.3	103.65
Tillatte farger:	7	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
5	0.51	1.6		0.84	2.6	162.49	0.51	1.4	87.5
6	0.51	2.6		1.15	6.3	242.3	0.82	2.7	103.84
7	1.03	6.8		2.91	23.5	345.58	1.05	9.0	132.35
8	2.01	27.4		2.58	50.0	182.48	2.44	32.2	117.51
9	5.91	68.5		3.02	89.4	130.51	2.88	72.9	106.42
Tillatte farger:	9	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.48	0.3	-1.0	0.0	0.0	-1.0
6	0.48	0.3		0.67	0.7	233.33	0.42	0.2	66.66
7	0.52	1.5		1.79	4.1	273.33	0.52	1.5	100.0
8	1.28	3.9		2.89	14.2	364.1	1.75	6.2	158.97
9	2.52	14.2		3.77	36.6	257.74	2.98	21.0	147.88
Tillatte farger:	11	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.78	1.2	-1.0	0.0	0.0	-1.0
8	0.42	0.2		1.41	1.7	850.0	0.51	0.6	300.0
9	0.66	2.0		2.26	10.0	500.0	1.47	3.2	160.0
Tillatte farger:	13	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0

Fortsetter på neste side

Tabell B.15 – fortsetter fra forrige side

Radius	tpfs	gj.snitt	%	mrfs	gj.snitt	%	mllfs	gj.snitt	%
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
8	0.0	0.0		0.52	0.5	-1.0	0.0	0.0	-1.0
9	0.0	0.0		0.82	1.7	-1.0	0.31	0.1	-1.0
Tillatte farger:	2	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.51	12.4		1.77	15.5	125.0	0.48	12.7	102.41
3	0.94	46.7		2.48	58.8	125.91	1.89	48.4	103.64
4	2.91	108.6		3.09	123.0	113.25	2.29	111.8	102.94
5	2.17	164.6		1.87	175.8	106.8	4.0	167.3	101.64
6	2.21	237.0		5.33	245.3	103.5	3.7	239.2	100.92
7	2.79	278.5		3.97	283.4	101.75	2.74	281.0	100.89
8	3.86	321.6		5.1	323.9	100.71	4.76	324.4	100.87
9	2.66	355.7		2.82	355.8	100.02	2.64	355.1	99.83
Tillatte farger:	3	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	1.0		1.43	2.5	250.0	0.0	1.0	100.0
3	0.69	13.4		1.56	16.7	124.62	0.84	13.6	101.49
4	1.15	29.0		2.98	44.4	153.1	1.68	32.2	111.03
5	2.44	62.8		3.48	82.8	131.84	2.89	70.2	111.78
6	1.95	135.4		3.27	152.6	112.7	4.6	143.9	106.27
7	3.3	182.3		5.04	197.9	108.55	4.17	191.1	104.82
8	3.74	242.5		3.02	247.4	102.02	4.02	243.0	100.2
9	4.0	284.5		3.56	290.5	102.1	3.74	287.0	100.87
Tillatte farger:	4	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
3	0.0	4.0		0.94	5.3	132.5	0.31	4.1	102.5
4	0.91	8.8		1.79	13.1	148.86	0.84	9.6	109.09
5	1.56	19.7		3.88	34.0	172.58	3.14	23.1	117.25
6	1.34	62.6		3.05	83.7	133.7	3.74	68.7	109.74
7	2.91	105.4		4.98	123.2	116.88	3.35	115.8	109.86
8	3.95	169.1		2.52	180.2	106.56	3.65	174.0	102.89
9	3.3	219.4		3.94	230.0	104.83	4.72	226.1	103.05
Tillatte farger:	5	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
Fortsetter på neste side									

Tabell B.15 – fortsetter fra forrige side

Radius	**tpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
3	0.0	0.0		0.69	0.4	-1.0	0.0	0.0	-1.0
4	0.69	2.4		1.34	4.4	183.33	0.73	3.1	129.16
5	0.78	6.8		2.51	11.9	174.99	0.67	7.3	107.35
6	1.61	21.8		3.53	39.5	181.19	2.59	27.5	126.14
7	2.01	50.5		3.62	72.6	143.76	2.28	58.9	116.63
8	3.54	104.9		6.02	125.4	119.54	4.14	112.4	107.14
9	3.62	158.3		3.8	173.5	109.6	4.62	165.9	104.8
Tillatte farger:	7	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.63	0.8	-1.0	0.0	0.0	-1.0
5	0.51	1.4		0.63	2.2	157.14	0.42	1.2	85.71
6	0.48	2.3		1.54	5.8	252.17	0.69	2.6	113.04
7	1.17	7.5		2.27	19.6	261.33	1.82	10.3	137.33
8	3.04	27.8		6.15	48.5	174.46	2.93	32.2	115.82
9	4.21	64.8		2.66	85.3	131.63	4.17	74.1	114.35
Tillatte farger:	9	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.42	0.2	-1.0	0.48	0.3	-1.0
6	0.31	0.1		0.47	1.0	1000.0	0.31	0.1	100.0
7	0.82	1.7		1.71	4.6	270.58	0.67	1.7	100.0
8	0.67	3.3		1.91	13.1	396.96	1.39	4.8	145.45
9	3.12	16.0		4.19	32.4	202.5	1.34	18.6	116.25
Tillatte farger:	11	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.66	1.0	-1.0	0.31	0.1	-1.0
8	0.48	0.7		1.28	2.9	414.28	0.51	0.4	57.14
9	0.82	2.3		2.7	10.0	434.78	0.67	2.7	117.39
Tillatte farger:	13	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
Fortsetter på neste side									

Tabell B.15 – fortsetter fra forrige side

Radius	tpfs	gj.snitt	%	mrfs	gj.snitt	%	mllfs	gj.snitt	%
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
8	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
9	0.63	0.2		0.69	1.6	800.0	0.0	0.0	0.0
Tillatte farger:	2	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.52	12.5		1.47	14.8	118.4	0.87	12.9	103.2
3	1.41	46.7		3.63	59.1	126.55	1.33	49.0	104.92
4	2.91	109.4		2.44	121.8	111.33	3.43	113.6	103.83
5	2.94	164.7		3.3	178.3	108.25	3.92	171.1	103.88
6	2.54	236.4		4.3	245.5	103.84	2.0	239.7	101.39
7	4.28	278.8		3.22	284.2	101.93	3.73	280.8	100.71
8	2.09	323.2		4.14	326.5	101.02	4.59	324.0	100.24
9	3.23	355.0		2.62	356.7	100.47	2.48	355.2	100.05
Tillatte farger:	3	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	1.0		0.63	2.8	280.0	0.0	1.0	100.0
3	0.0	13.0		1.64	17.4	133.84	0.63	13.2	101.53
4	0.84	29.5		3.13	44.6	151.18	2.05	32.7	110.84
5	1.17	63.5		4.21	82.3	129.6	2.34	68.8	108.34
6	4.76	136.3		3.49	156.7	114.96	1.26	141.5	103.81
7	3.65	182.4		3.01	196.8	107.89	1.96	190.1	104.22
8	5.96	239.7		5.01	249.3	104.0	4.04	241.9	100.91
9	3.74	285.5		5.67	292.7	102.52	3.56	287.4	100.66
Tillatte farger:	4	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.31	0.1	-1.0	0.0	0.0	-1.0
3	0.0	4.0		0.97	5.5	137.5	0.0	4.0	100.0
4	0.51	8.6		1.72	15.9	184.88	1.13	9.8	113.95
5	0.91	20.2		2.31	34.6	171.28	2.17	23.5	116.33
6	2.5	62.5		3.16	83.4	133.44	3.8	71.7	114.72
7	2.78	103.2		2.42	127.9	123.93	4.34	113.7	110.17
8	4.4	166.9		2.66	184.7	110.66	3.27	173.6	104.01
9	2.74	218.2		4.76	232.5	106.55	3.49	223.0	102.19
Tillatte farger:	5	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
Fortsetter på neste side									

Tabell B.15 – fortsetter fra forrige side

Radius	**tpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.63	0.8	-1.0	0.31	0.1	-1.0
4	0.63	2.8		1.19	4.9	175.0	0.47	3.0	107.14
5	0.69	6.6		2.37	12.1	183.33	0.51	6.4	96.96
6	2.16	22.7		2.82	43.2	190.3	1.59	27.1	119.38
7	2.36	53.5		3.9	72.9	136.26	2.83	60.5	113.08
8	3.84	101.9		4.92	128.7	126.3	2.88	112.9	110.79
9	6.05	158.5		3.06	176.4	111.29	5.32	166.9	105.29
Tillatte farger:	7	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.63	0.8	-1.0	0.0	0.0	-1.0
5	0.42	1.2		0.0	2.0	166.66	0.48	1.3	108.33
6	0.48	2.3		1.56	6.3	273.91	0.67	2.7	117.39
7	1.05	7.3		2.79	23.3	319.17	1.64	12.4	169.86
8	1.95	28.5		4.3	51.1	179.29	2.02	32.1	112.63
9	5.35	66.5		4.71	91.5	137.59	3.01	73.2	110.07
Tillatte farger:	9	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.48	0.3	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.63	0.8	-1.0	0.31	0.1	-1.0
7	0.31	1.1		0.96	4.4	400.0	0.66	2.0	181.81
8	1.05	4.0		1.7	14.3	357.5	1.41	5.3	132.5
9	2.95	16.6		2.39	36.8	221.68	4.85	21.5	129.51
Tillatte farger:	11	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.31	0.1	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.51	0.4	-1.0	0.31	0.1	-1.0
8	0.48	0.3		0.78	3.2	1066.66	0.31	1.1	366.66
9	0.97	2.5		2.21	12.3	492.0	0.67	2.3	91.99
Tillatte farger:	13	Initiell fordeling:	ff						
Fortsetter på neste side									

Tabell B.15 – fortsetter fra forrige side

Radius	**tpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
8	0.0	0.0		0.48	0.3	-1.0	0.0	0.0	-1.0
9	0.0	0.0		0.82	2.7	-1.0	0.48	0.3	-1.0

B.5 Simuleringer med $LLPF$–algoritmen.

Tabell B.16: Tabell der $LLPF$–algortimen velger laveste ledige farge.

Radius	**pf**	**gj.snitt**	**%**	**mrf**	**gj.snitt**	**%**	**mllf**	**gj.snitt**	**%**
Tillatte farger:	2	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.91	12.8		1.19	14.1	110.15	0.69	12.6	98.43
3	1.07	46.4		1.44	53.1	114.43	2.4	49.7	107.11
4	3.41	109.1		2.49	117.3	107.51	2.05	114.0	104.49
5	2.98	163.6		2.98	171.7	104.95	4.07	168.8	103.17
6	4.35	238.1		5.27	242.7	101.93	2.45	237.3	99.66
7	5.52	281.5		3.46	281.7	100.07	5.11	283.8	100.81
8	3.94	325.3		3.83	327.4	100.64	3.47	322.6	99.17
9	5.06	354.9		3.59	355.7	100.22	4.0	355.3	100.11
Tillatte farger:	3	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	1.0		0.84	1.6	160.0	0.0	1.0	100.0
3	0.42	13.2		1.5	15.4	116.66	0.51	13.6	103.03
4	0.84	28.5		3.3	39.3	137.89	1.63	32.0	112.28
5	1.56	62.0		3.09	76.4	123.22	4.44	68.2	109.99
6	3.13	135.4		4.56	146.8	108.41	5.45	143.8	106.2
7	4.39	183.0		2.27	193.6	105.79	3.42	189.8	103.71
8	3.91	240.2		5.71	248.6	103.49	5.01	246.5	102.62
9	3.9	285.1		5.46	290.5	101.89	3.55	287.8	100.94
Tillatte farger:	4	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.31	0.1	-1.0	0.0	0.0	-1.0
3	0.0	4.0		0.81	5.0	125.0	0.31	4.1	102.5
4	0.99	8.9		1.88	11.7	131.46	0.78	8.8	98.87
5	1.22	19.2		4.37	30.7	159.89	1.68	22.8	118.74
6	2.25	60.2		2.86	81.0	134.55	2.52	72.2	119.93
7	3.56	104.3		6.56	120.6	115.62	3.87	113.1	108.43
8	3.82	164.8		5.07	176.0	106.79	5.32	175.1	106.25
9	3.3	218.7		3.6	227.1	103.84	4.29	221.4	101.23
Tillatte farger:	5	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.78	0.8	-1.0	0.0	0.0	-1.0
4	0.42	2.2		1.15	4.3	195.45	0.63	2.8	127.27
Fortsetter på neste side									

Tabell B.16 – fortsetter fra forrige side

Radius	**llpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
5	0.51	6.4		1.47	10.8	168.75	0.51	6.6	103.12
6	2.23	21.1		3.65	35.3	167.29	2.48	27.2	128.9
7	2.51	48.1		4.49	71.2	148.02	3.66	59.1	122.86
8	3.12	101.0		5.02	123.2	121.98	3.65	113.4	112.27
9	4.92	155.3		4.96	171.3	110.3	3.19	167.8	108.04
Tillatte farger:	7	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.52	0.5	-1.0	0.0	0.0	-1.0
5	0.42	1.2		0.73	2.1	174.99	0.51	1.4	116.66
6	0.42	2.2		1.5	5.6	254.54	0.51	2.4	109.09
7	1.1	6.9		3.28	17.1	247.82	1.49	10.3	149.27
8	2.2	24.8		2.44	44.8	180.64	3.33	32.6	131.45
9	3.09	60.7		5.42	84.5	139.2	3.09	71.6	117.95
Tillatte farger:	9	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.48	0.7	-1.0	0.42	0.2	-1.0
7	0.31	1.1		1.25	3.7	336.36	0.84	1.6	145.45
8	0.73	3.1		1.57	11.4	367.74	0.96	4.6	148.38
9	1.15	12.3		2.78	32.2	261.78	2.45	19.4	157.72
Tillatte farger:	11	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.67	0.7	-1.0	0.0	0.0	-1.0
8	0.31	0.1		0.94	2.3	2300.0	0.48	0.3	300.0
9	0.69	1.6		1.56	8.7	543.75	0.96	2.6	162.49
Tillatte farger:	13	Initiell fordeling:	tt						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0

Fortsetter på neste side

Tabell B.16 – fortsetter fra forrige side

Radius	llpfs	gj.snitt	%	mrfs	gj.snitt	%	mllfs	gj.snitt	%
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
8	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
9	0.31	0.1		0.69	1.6	1600.0	0.0	0.0	0.0
Tillatte farger:	2	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.69	12.6		0.84	15.5	123.01	0.48	12.7	100.79
3	1.07	46.4		2.25	55.2	118.96	0.81	49.0	105.6
4	2.7	106.0		2.83	118.3	111.6	3.63	113.1	106.69
5	2.16	163.0		3.55	173.2	106.25	3.52	171.0	104.9
6	3.35	240.2		4.15	244.2	101.66	4.14	242.5	100.95
7	3.7	277.8		2.86	281.0	101.15	3.09	282.7	101.76
8	3.31	323.9		3.56	324.7	100.24	3.3	321.7	99.32
9	3.14	355.9		4.1	359.8	101.09	2.87	355.5	99.88
Tillatte farger:	3	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	1.0		1.19	3.1	310.0	0.0	1.0	100.0
3	0.42	13.2		1.47	15.8	119.69	0.69	13.6	103.03
4	1.87	28.8		3.47	40.9	142.01	1.85	32.9	114.23
5	2.04	62.8		3.91	84.3	134.23	2.39	67.8	107.96
6	2.35	137.3		2.75	154.7	112.67	3.19	141.8	103.27
7	3.02	184.6		4.98	199.3	107.96	3.12	188.7	102.22
8	3.06	240.6		2.52	250.8	104.23	3.51	240.9	100.12
9	4.38	284.1		2.8	293.1	103.16	3.29	288.0	101.37
Tillatte farger:	4	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	4.0		0.56	4.9	122.5	0.0	4.0	100.0
4	0.48	8.3		1.56	15.0	180.72	1.37	10.1	121.68
5	1.07	20.6		3.52	34.7	168.44	1.85	23.1	112.13
6	3.52	62.0		4.04	84.8	136.77	1.88	69.7	112.41
7	2.21	106.0		3.13	123.4	116.41	3.55	115.2	108.67
8	3.64	165.2		3.91	181.2	109.68	4.86	174.1	105.38
9	3.12	223.7		3.98	228.1	101.96	4.84	226.1	101.07
Tillatte farger:	5	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0

Fortsetter på neste side

Tabell B.16 – fortsetter fra forrige side

Radius	**llpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
3	0.0	0.0		0.7	0.5	-1.0	0.0	0.0	-1.0
4	0.78	2.8		1.31	4.2	150.0	0.42	2.8	100.0
5	0.31	6.9		2.35	13.0	188.4	1.17	7.5	108.69
6	1.71	22.5		2.95	40.6	180.44	2.31	27.4	121.77
7	2.28	53.9		2.98	72.0	133.58	4.05	59.3	110.01
8	4.84	106.1		3.72	123.9	116.77	5.13	113.1	106.59
9	4.04	155.1		4.43	177.1	114.18	5.16	164.3	105.93
Tillatte farger:	7	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
5	0.31	1.1		0.84	2.6	236.36	0.51	1.4	127.27
6	0.42	2.8		1.15	6.3	225.0	0.82	2.7	96.42
7	1.22	6.8		2.91	23.5	345.58	1.05	9.0	132.35
8	2.59	26.5		2.58	50.0	188.67	2.44	32.2	121.5
9	6.08	64.2		3.02	89.4	139.25	2.88	72.9	113.55
Tillatte farger:	9	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.48	0.3	-1.0	0.0	0.0	-1.0
6	0.42	0.2		0.67	0.7	350.0	0.42	0.2	100.0
7	0.48	1.7		1.79	4.1	241.17	0.52	1.5	88.23
8	0.94	4.3		2.89	14.2	330.23	1.75	6.2	144.18
9	2.72	14.9		3.77	36.6	245.63	2.98	21.0	140.93
Tillatte farger:	11	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.78	1.2	-1.0	0.0	0.0	-1.0
8	0.81	1.0		1.41	1.7	170.0	0.51	0.6	60.0
9	0.78	2.2		2.26	10.0	454.54	1.47	3.2	145.45
Tillatte farger:	13	Initiell fordeling:	tf						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
Fortsetter på neste side									

Tabell B.16 – fortsetter fra forrige side

Radius	llpfs	gj.snitt	%	mrfs	gj.snitt	%	mllfs	gj.snitt	%
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
8	0.0	0.0		0.52	0.5	-1.0	0.0	0.0	-1.0
9	0.63	0.2		0.82	1.7	850.0	0.31	0.1	50.0
Tillatte farger:	2	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.31	12.1		1.77	15.5	128.09	0.48	12.7	104.95
3	2.45	46.6		2.48	58.8	126.18	1.89	48.4	103.86
4	1.68	108.8		3.09	123.0	113.05	2.29	111.8	102.75
5	1.52	164.1		1.87	175.8	107.12	4.0	167.3	101.95
6	3.43	235.7		5.33	245.3	104.07	3.7	239.2	101.48
7	3.95	279.1		3.97	283.4	101.54	2.74	281.0	100.68
8	2.91	323.5		5.1	323.9	100.12	4.76	324.4	100.27
9	2.91	354.6		2.82	355.8	100.33	2.64	355.1	100.14
Tillatte farger:	3	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	1.0		1.43	2.5	250.0	0.0	1.0	100.0
3	0.7	13.5		1.56	16.7	123.7	0.84	13.6	100.74
4	2.07	29.9		2.98	44.4	148.49	1.68	32.2	107.69
5	1.93	62.2		3.48	82.8	133.11	2.89	70.2	112.86
6	2.66	133.7		3.27	152.6	114.13	4.6	143.9	107.62
7	2.25	184.2		5.04	197.9	107.43	4.17	191.1	103.74
8	3.81	238.9		3.02	247.4	103.55	4.02	243.0	101.71
9	3.36	285.0		3.56	290.5	101.92	3.74	287.0	100.7
Tillatte farger:	4	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
3	0.0	4.0		0.94	5.3	132.5	0.31	4.1	102.5
4	0.52	8.5		1.79	13.1	154.11	0.84	9.6	112.94
5	1.33	20.7		3.88	34.0	164.25	3.14	23.1	111.59
6	3.6	61.1		3.05	83.7	136.98	3.74	68.7	112.43
7	3.23	105.4		4.98	123.2	116.88	3.35	115.8	109.86
8	2.27	167.5		2.52	180.2	107.58	3.65	174.0	103.88
9	5.52	219.4		3.94	230.0	104.83	4.72	226.1	103.05
Tillatte farger:	5	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
Fortsetter på neste side									

Tabell B.16 – fortsetter fra forrige side

Radius	**llpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.69	0.4	-1.0	0.0	0.0	-1.0
4	0.67	2.7		1.34	4.4	162.96	0.73	3.1	114.81
5	0.51	6.4		2.51	11.9	185.93	0.67	7.3	114.06
6	1.9	22.5		3.53	39.5	175.55	2.59	27.5	122.22
7	4.27	50.6		3.62	72.6	143.47	2.28	58.9	116.4
8	4.66	102.2		6.02	125.4	122.7	4.14	112.4	109.98
9	4.8	156.0		3.8	173.5	111.21	4.62	165.9	106.34
Tillatte farger:	7	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.63	0.8	-1.0	0.0	0.0	-1.0
5	0.42	1.2		0.63	2.2	183.33	0.42	1.2	100.0
6	0.48	2.3		1.54	5.8	252.17	0.69	2.6	113.04
7	1.87	7.2		2.27	19.6	272.22	1.82	10.3	143.05
8	2.76	26.9		6.15	48.5	180.29	2.93	32.2	119.7
9	3.3	62.4		2.66	85.3	136.69	4.17	74.1	118.75
Tillatte farger:	9	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.42	0.2	-1.0	0.48	0.3	-1.0
6	0.31	0.1		0.47	1.0	1000.0	0.31	0.1	100.0
7	0.42	1.2		1.71	4.6	383.33	0.67	1.7	141.66
8	1.05	3.7		1.91	13.1	354.05	1.39	4.8	129.72
9	1.82	13.3		4.19	32.4	243.6	1.34	18.6	139.84
Tillatte farger:	11	Initiell fordeling:	ft						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.66	1.0	-1.0	0.31	0.1	-1.0
8	0.48	0.3		1.28	2.9	966.66	0.51	0.4	133.33
9	0.56	2.1		2.7	10.0	476.19	0.67	2.7	128.57
Tillatte farger:	13	Initiell fordeling:	ft						
Fortsetter på neste side									

Tabell B.16 – fortsetter fra forrige side

Radius	**llpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
8	0.0	0.0		0.42	0.2	-1.0	0.0	0.0	-1.0
9	0.31	0.1		0.69	1.6	1600.0	0.0	0.0	0.0
Tillatte farger:	2	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.67	12.7		1.47	14.8	116.53	0.87	12.9	101.57
3	1.28	44.9		3.63	59.1	131.62	1.33	49.0	109.13
4	2.14	107.8		2.44	121.8	112.98	3.43	113.6	105.38
5	3.21	161.9		3.3	178.3	110.12	3.92	171.1	105.68
6	3.08	236.8		4.3	245.5	103.67	2.0	239.7	101.22
7	5.12	280.6		3.22	284.2	101.28	3.73	280.8	100.07
8	4.1	322.0		4.14	326.5	101.39	4.59	324.0	100.62
9	2.91	353.5		2.62	356.7	100.9	2.48	355.2	100.48
Tillatte farger:	3	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	1.0		0.63	2.8	280.0	0.0	1.0	100.0
3	0.0	13.0		1.64	17.4	133.84	0.63	13.2	101.53
4	1.76	30.3		3.13	44.6	147.19	2.05	32.7	107.92
5	2.85	62.2		4.21	82.3	132.31	2.34	68.8	110.61
6	2.58	133.7		3.49	156.7	117.2	1.26	141.5	105.83
7	3.8	185.7		3.01	196.8	105.97	1.96	190.1	102.36
8	2.79	242.6		5.01	249.3	102.76	4.04	241.9	99.71
9	3.6	284.9		5.67	292.7	102.73	3.56	287.4	100.87
Tillatte farger:	4	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.31	0.1	-1.0	0.0	0.0	-1.0
3	0.0	4.0		0.97	5.5	137.5	0.0	4.0	100.0
4	0.81	9.0		1.72	15.9	176.66	1.13	9.8	108.88
5	1.75	20.2		2.31	34.6	171.28	2.17	23.5	116.33
6	3.9	61.9		3.16	83.4	134.73	3.8	71.7	115.83
7	3.33	105.4		2.42	127.9	121.34	4.34	113.7	107.87
8	3.09	164.6		2.66	184.7	112.21	3.27	173.6	105.46
9	4.41	221.2		4.76	232.5	105.1	3.49	223.0	100.81
Fortsetter på neste side									

Tabell B.16 – fortsetter fra forrige side

Radius	**llpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
Tillatte farger:	5	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.63	0.8	-1.0	0.31	0.1	-1.0
4	0.56	2.9		1.19	4.9	168.96	0.47	3.0	103.44
5	0.48	6.3		2.37	12.1	192.06	0.51	6.4	101.58
6	1.82	21.0		2.82	43.2	205.71	1.59	27.1	129.04
7	3.83	51.6		3.9	72.9	141.27	2.83	60.5	117.24
8	3.86	105.6		4.92	128.7	121.87	2.88	112.9	106.91
9	3.65	157.3		3.06	176.4	112.14	5.32	166.9	106.1
Tillatte farger:	7	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.63	0.8	-1.0	0.0	0.0	-1.0
5	0.48	1.3		0.0	2.0	153.84	0.48	1.3	100.0
6	0.69	2.6		1.56	6.3	242.3	0.67	2.7	103.84
7	1.15	7.7		2.79	23.3	302.59	1.64	12.4	161.03
8	2.17	29.4		4.3	51.1	173.8	2.02	32.1	109.18
9	4.42	66.3		4.71	91.5	138.0	3.01	73.2	110.4
Tillatte farger:	9	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.48	0.3	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.63	0.8	-1.0	0.31	0.1	-1.0
7	0.63	1.2		0.96	4.4	366.66	0.66	2.0	166.66
8	1.07	4.6		1.7	14.3	310.86	1.41	5.3	115.21
9	3.12	16.7		2.39	36.8	220.35	4.85	21.5	128.74
Tillatte farger:	11	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.31	0.1	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.51	0.4	-1.0	0.31	0.1	-1.0
8	0.31	0.1		0.78	3.2	3200.0	0.31	1.1	1100.0
Fortsetter på neste side									

Tabell B.16 – fortsetter fra forrige side

Radius	**llpfs**	**gj.snitt**	**%**	**mrfs**	**gj.snitt**	**%**	**mllfs**	**gj.snitt**	**%**
9	0.87	2.1		2.21	12.3	585.71	0.67	2.3	109.52
Tillatte farger:	13	Initiell fordeling:	ff						
Radius	pf			mrf			mllf		
	s	gj.snitt		s	gj.snitt	prosent	s	gj.snitt	prosent
2	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
3	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
4	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
5	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
6	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
7	0.0	0.0		0.0	0.0	-1.0	0.0	0.0	-1.0
8	0.0	0.0		0.48	0.3	-1.0	0.0	0.0	-1.0
9	0.63	0.2		0.82	2.7	1350.0	0.48	0.3	150.0

Bibliografi

[1] M. J. Atallah, editor. *Algorithms and theory of computation handbook*. CRC Press, 1998.

[2] M. Bellare, O. Goldreich, and M. Sudan. Free bits, pcps and non-approximability. *SIAMJ. on Computing, Vol. 27, No 3, 1998 pp. 804-915*, 1998.

[3] Self-Stabilization Bibliography. ftp://www.cs.uiowa.edu/pub/selfstab/bibliography/index.html. 15. februar 2004.

[4] Jbuilder by Borland. *http://www.borland.com/jbuilder/*. 10. januar 2004.

[5] T. F. Coleman and J. J. Moré. Estimation of sparse jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal., 20(1):187-209, Februar 1983*, 1983.

[6] E. W. Dijkstra. Self-stabilizing system in spite of distributed control. *Communications of the ACM*, 1974.

[7] M. S. Gast. *802.11 Wireless Network - The Definitive Guide*. O'Reilly, 2002.

[8] A. H. Gebremedhin. Parallel graph coloring. Master's thesis, Universitetet i Bergen, 1999.

[9] A. H. Gebremedhin and F. Manne. Scalable parallel graph coloring algorithms. *Concurrency: Practice and Experience 2000; 12:1131-1146*, 2000.

[10] M. M. Halldórsson. A still better preformance guarantee for approximate graph coloring. *Information Processing Letters, 45:19–23, 25 januar 1993*, 1993.

[11] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Linear time self-stabilizing colorings. *Computer Science Web*, 2002.

[12] S.M. Hedetniemi, S.T. Hedetniemi, D.P. Jacobs, and P.K. Srimani. Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. *Department of Computer Science Clemson University*, 2001.

[13] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal of Computing 15(4) 1036-1053*, 1986.

[14] J. W. Moon and L. Moser. On cliques in graphs. *Israel J. Math. 3:23-28*, 1965.

[15] Post og Teletilsynet. http://www.npt.no. 18. februar 2004.

[16] Alan Turing Home page. http://www.turing.org/turing.

[17] K. Pahlavan and P. Krishnamurthy. *Principles of wireless networks*. Prentice Hall PTR, 2002.

[18] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[19] S. Sur and P. K. Srimani. A self-stabilizing algorithm for coloring bipartite graphs. *Department of Computer Science, Colorado State University*.

[20] Textually.org. http://www.textually.org/textually/archives/cat_sms_a_little_history.htm. 10. mars 2004.

[21] R. J. Wilson. *Introduction to Graph Theory*. Person Education Limited, 1999.