

# Selvstabiliserende algoritmer for frekvensallokering

Tom Martens Meyer Solem  
Institutt for Informatikk  
Universitetet i Bergen  
24. juni 2004



# Innhold

- ⇒ Innledning.
- ⇒ Definisjoner.
- ⇒ Selvstabiliserende algoritmer.
- ⇒ Resultater, oppsummering og konklusjon.

# Innledning

Trådløs kommunikasjon blir stadig mer vanlig.

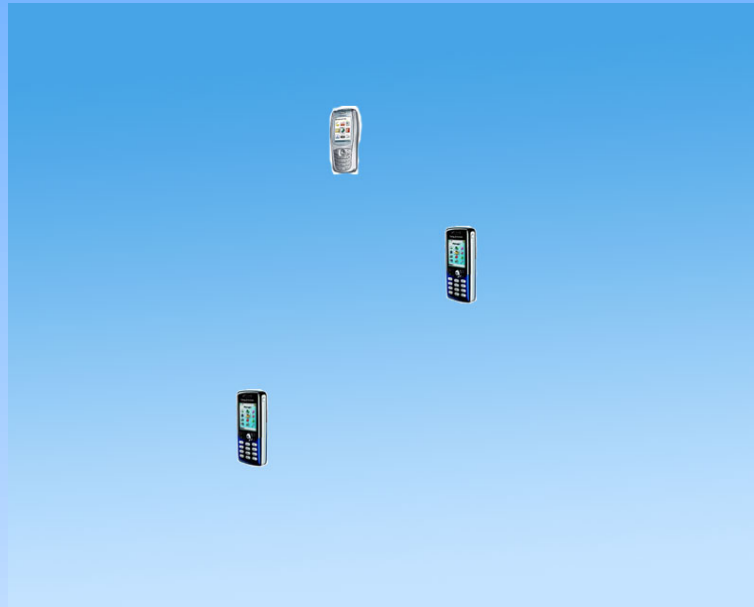
⇒ 1905, første trådløse telegram over Atlanterhavet.

⇒ 1973, første trådløse telefonsamtale.

⇒ 1997, første produkter basert på *IEEE* 802.11b lansert.

⇒ Hot spot, *Wi-Fi*, *ad hoc* nettverk.

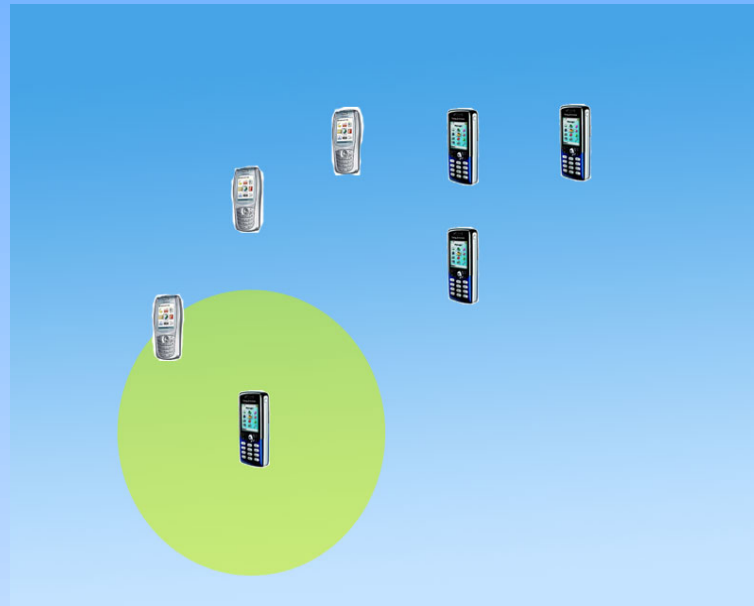
# *Ad hoc* nettverk



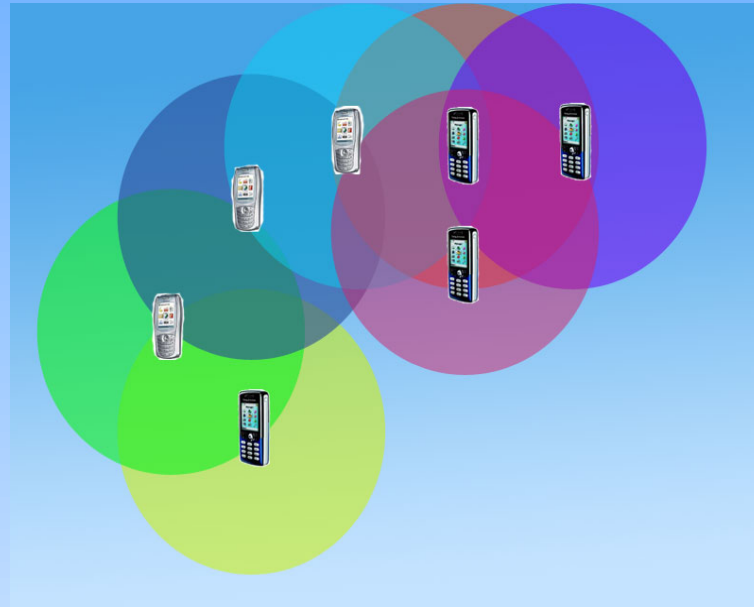
# *Ad hoc* nettverk



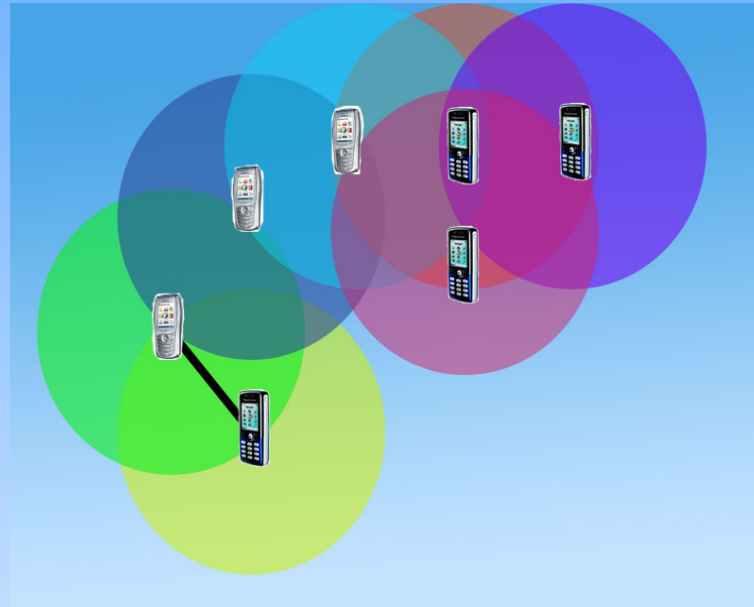
# *Ad hoc* nettverk



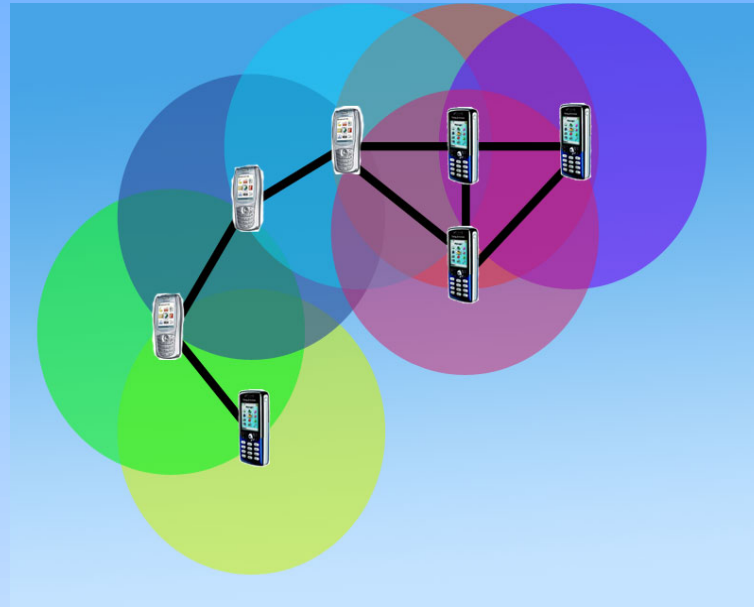
# *Ad hoc* nettverk



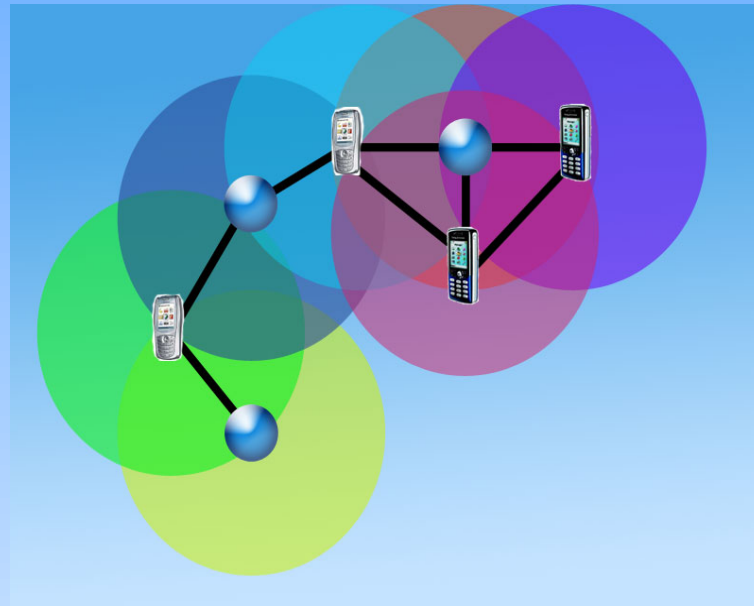
# *Ad hoc* nettverk



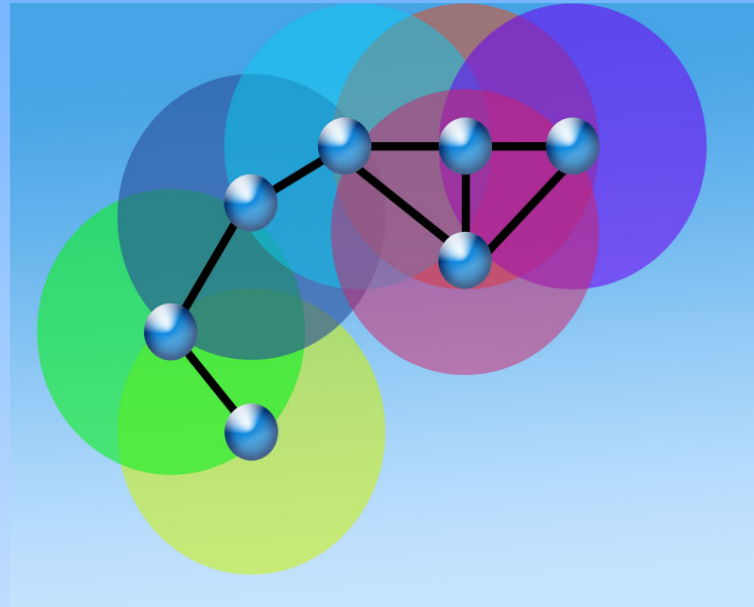
# *Ad hoc* nettverk



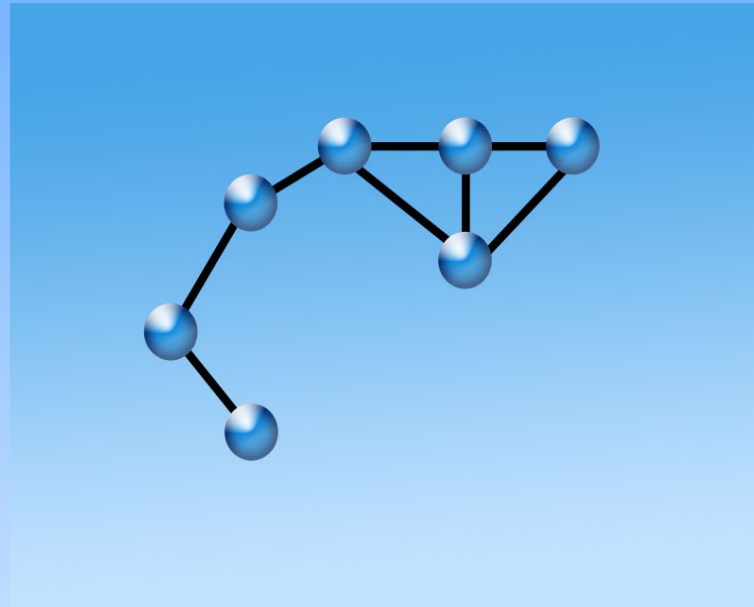
# *Ad hoc* nettverk



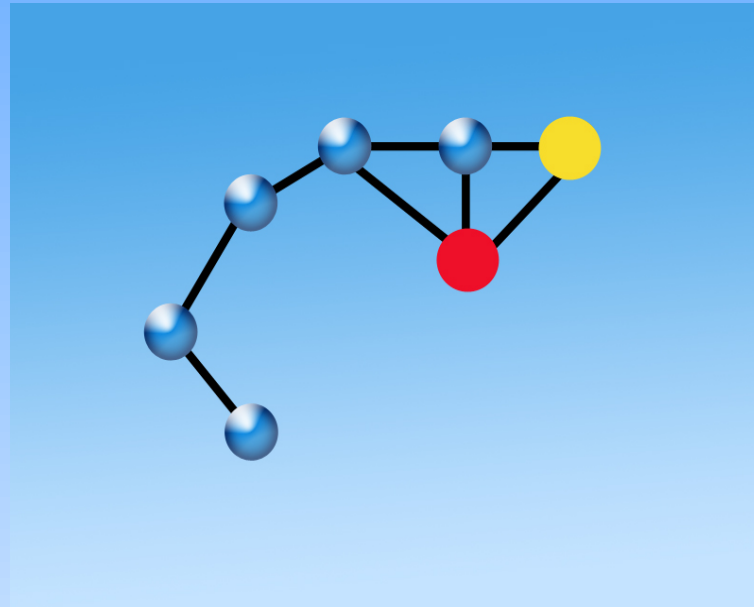
# *Ad hoc* nettverk



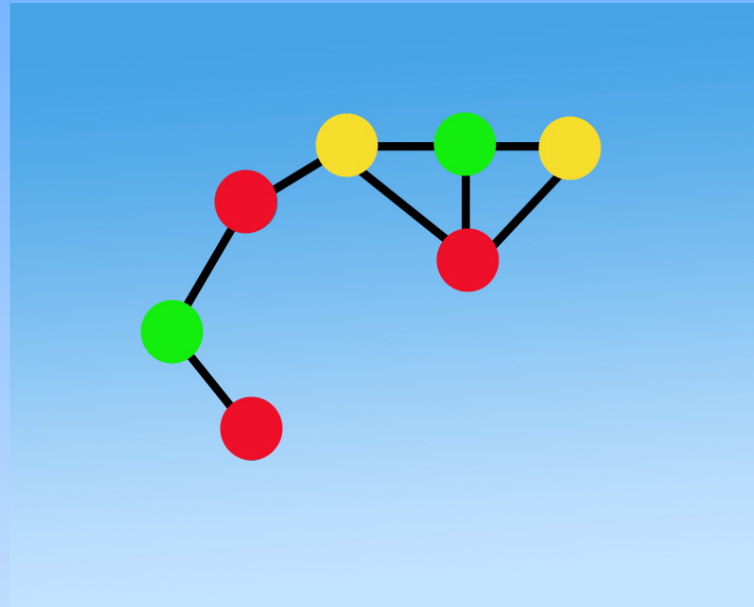
# *Ad hoc* nettverk



# *Ad hoc* nettverk



# *Ad hoc* nettverk



# Problemstilling

# Problemstilling

- ➡ Frekvenstildeling er det samme som graffargelegging.
- ➡ Alle grafer kan fargelegges med  $k$  farger, for  $k > \Delta$ .
- ➡ Hva om  $k < \Delta$ ?

# Problemstilling

- ➡ Frekvenstildeling er det samme som graffargelegging.
- ➡ Alle grafer kan fargelegges med  $k$  farger, for  $k > \Delta$ .
- ➡ Hva om  $k < \Delta$ ?  $\Rightarrow$  *klikk* av størrelse  $k$ . Antall uavhengige mengder . . .

# SelvStabiliserende Algoritmer (SSA)

⇒ SSA består av en mengde med regler

$$\mathcal{R} = \{r_1, r_2, \dots, r_k\}$$

⇒ Hver node får den samme mengden med regler.

⇒ Alle nodene har en mengde med variabler: ulike egenskaper, f.eks. farger, boolske operatorer.

⇒ Reglene består av et predikat  $P$  og en handling  $h$ .

# SelvStabiliserende Algoritmer (SSA)

- ⇒ En node er **privilegert** hvis den har minst et predikat som er *sant*.
- ⇒ De nodene som er privilegerte, kan utføre handlingen til de reglene med et *sant* predikat
- ⇒ Seriell modell av SSA. To naboer utfører ikke handlinger samtidig.

# Hvorfor SSA passer med *ad hoc* nettverk:

- ⇒ SSA er distribuerte akkurat som *ad hoc* nettverk.
- ⇒ SSA stabiliserer seg mot en ønsket tilstand.
- ⇒ SSA er robust ovenfor feil: noder kommer/går, faller ut . . .

# SSA Fargeleggingsalgoritme

# SSA Fargeleggingsalgoritme

Modifisert Rask Fargelegging *MRF* (Fast Coloring)  
(Hedetniemi, Jacobs, Sirmani)

En Regel: Predikat:

# SSA Fargeleggingsalgoritme

Modifisert Rask Fargelegging *MRF* (Fast Coloring)  
(Hedetniemi, Jacobs, Sirmani)

En Regel: Predikat:

Hvis{ (samme farge som en nabo)

eller

( -1 og ledige farger)

eller

(farge > tillatt) }

# SSA Fargeleggingsalgoritme

Modifisert Rask Fargelegging *MRF* (Fast Coloring)  
(Hedetniemi, Jacobs, Sirmani)

En Regel: Predikat:

Hvis{ (samme farge som en nabo)

eller

( -1 og ledige farger)

eller

(farge > tillatt) }

Handlingen:

Hvis(det er ledige farger)

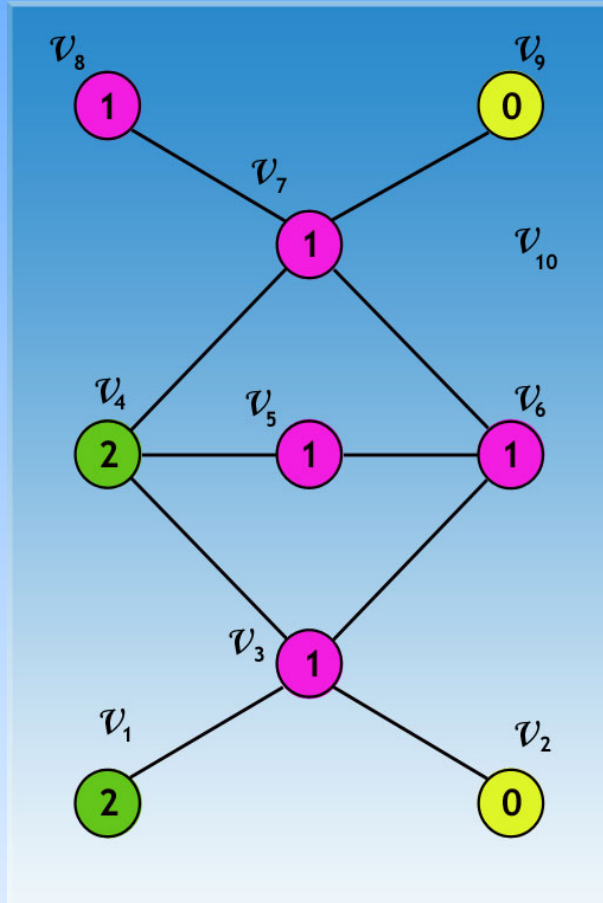
velg ledig farge.

ellers

velg -1 fargen.

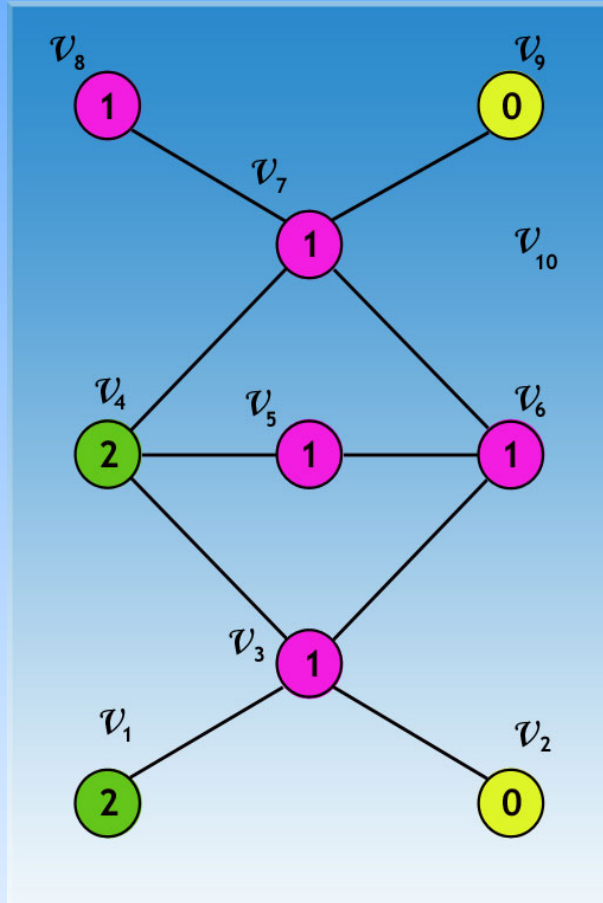
# Eksempel SSA Fargelegging

# Eksempel SSA Fargelegging



Det er 3 tillatte farger.  
{0, 1, 2}

# Eksempel SSA Fargelegging

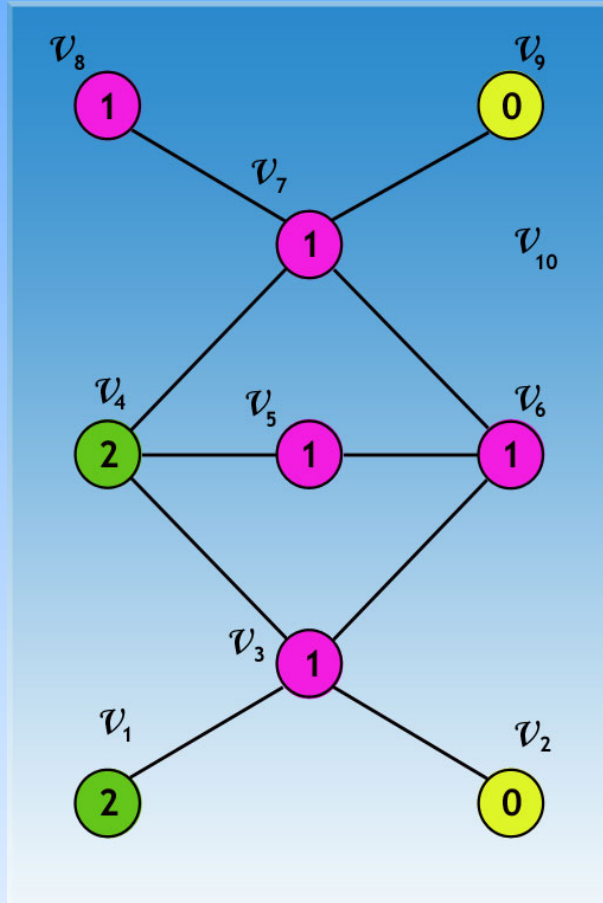


Det er 3 tillatte farger.

$\{0, 1, 2\}$

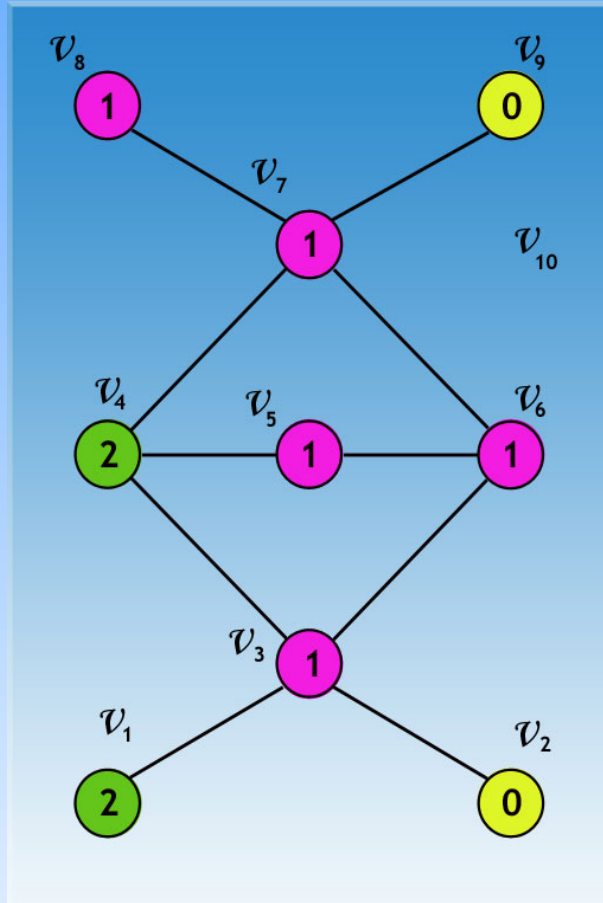
Nodene  $v_3$ ,  $v_5$ ,  $v_6$ ,  $v_7$  og  $v_8$  er privilegerte.

# Eksempel SSA Fargelegging



⇒ Noden  $v_5$  utfører handlingen.

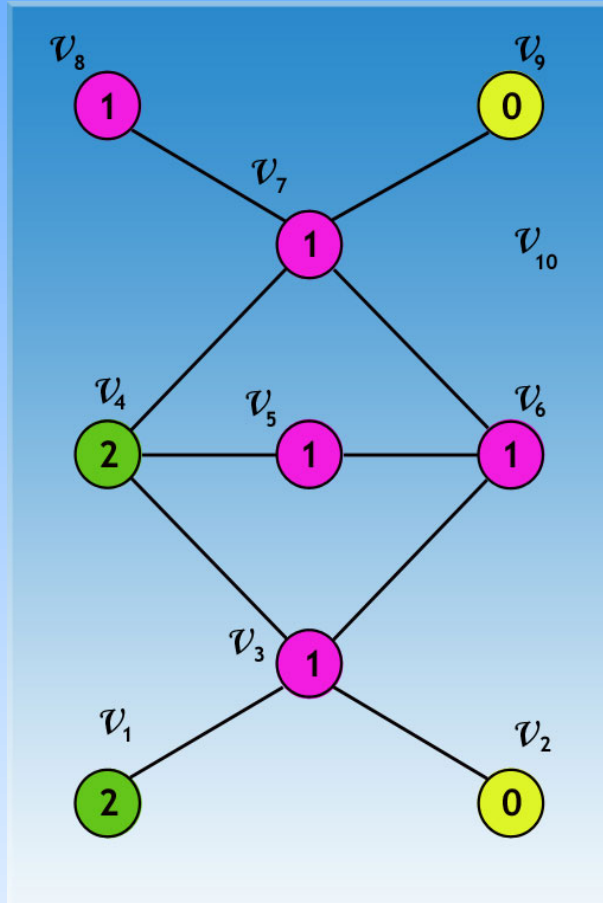
# Eksempel SSA Fargelegging



⇒ Noden  $v_5$  utfører handlingen.

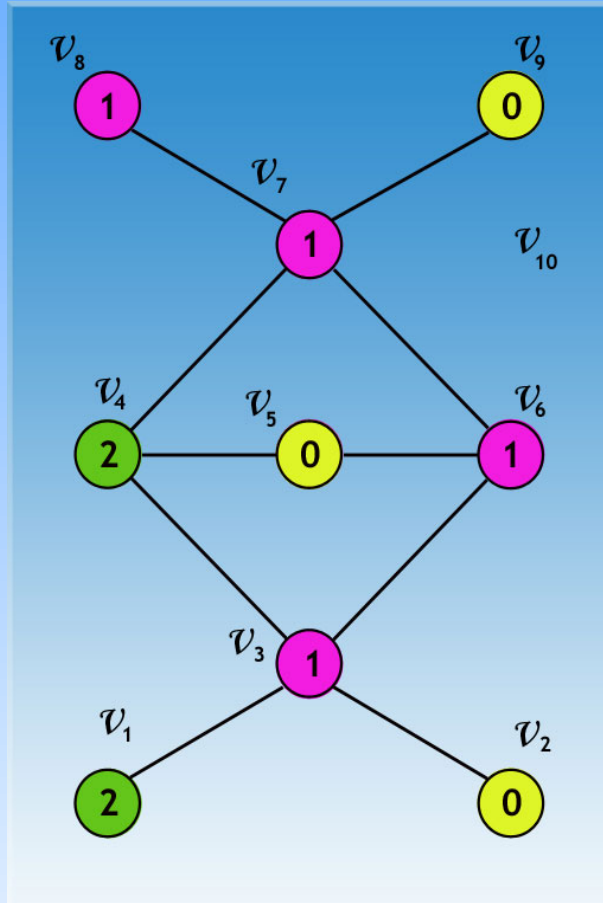
⇒ Hvis(det er ledige farger)

# Eksempel SSA Fargelegging



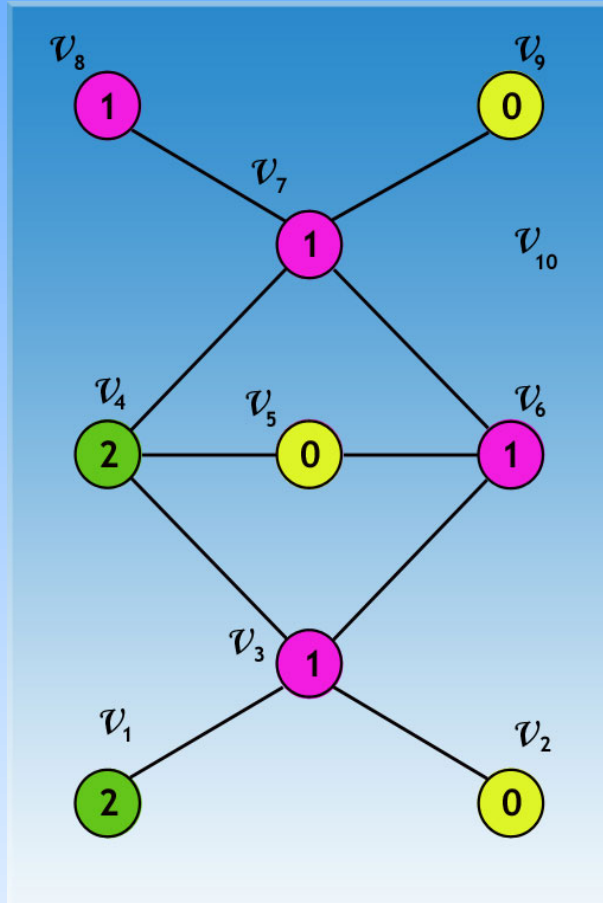
- ⇒ Noden  $v_5$  utfører handlingen.
- ⇒ Hvis(det er ledige farger)
- ⇒ velg ledig farge.

# Eksempel SSA Fargelegging



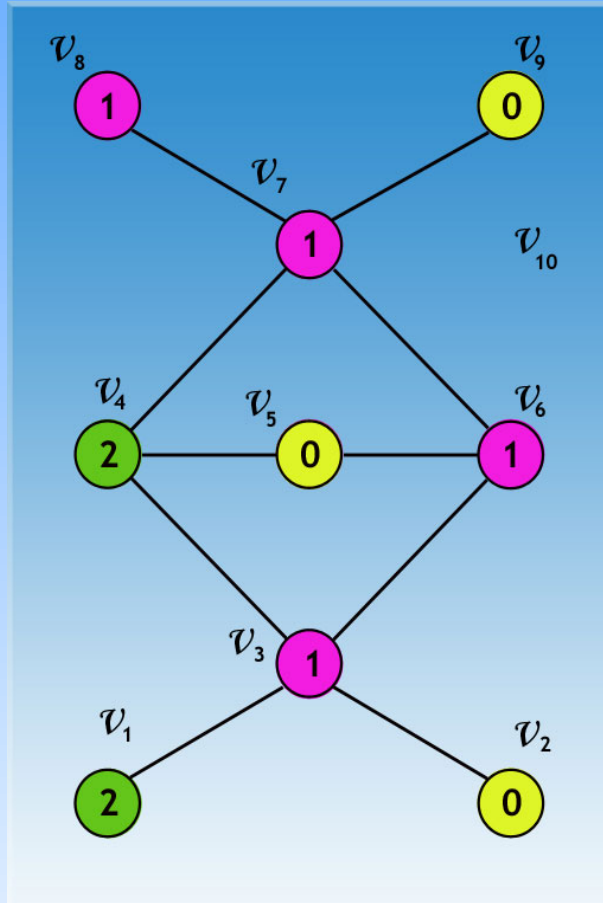
- ⇒ Noden  $v_5$  utfører handlingen.
- ⇒ Hvis(det er ledige farger)
- ⇒ velg ledig farge.

# Eksempel SSA Fargelegging



⇒ Noden  $v_3$  utfører handlingen.

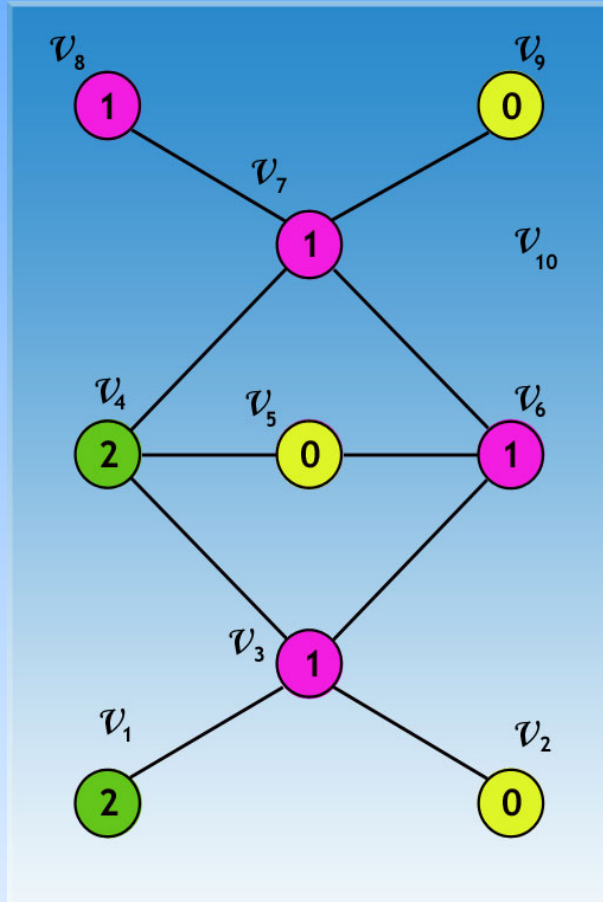
# Eksempel SSA Fargelegging



⇒ Noden  $v_3$  utfører handlingen.

⇒ Hvis(det er ledige farger)

# Eksempel SSA Fargelegging

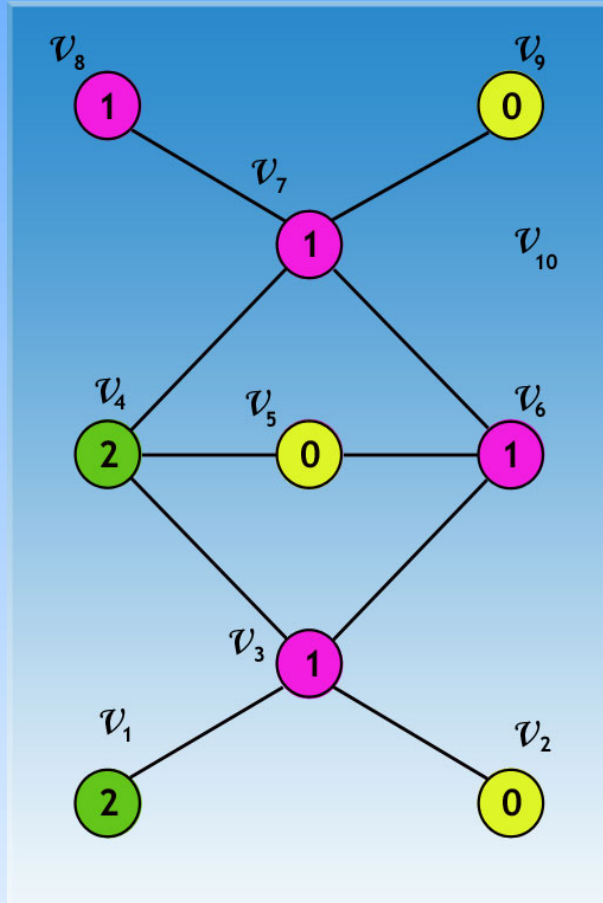


⇒ Noden  $v_3$  utfører handlingen.

⇒ Hvis(det er ledige farger)

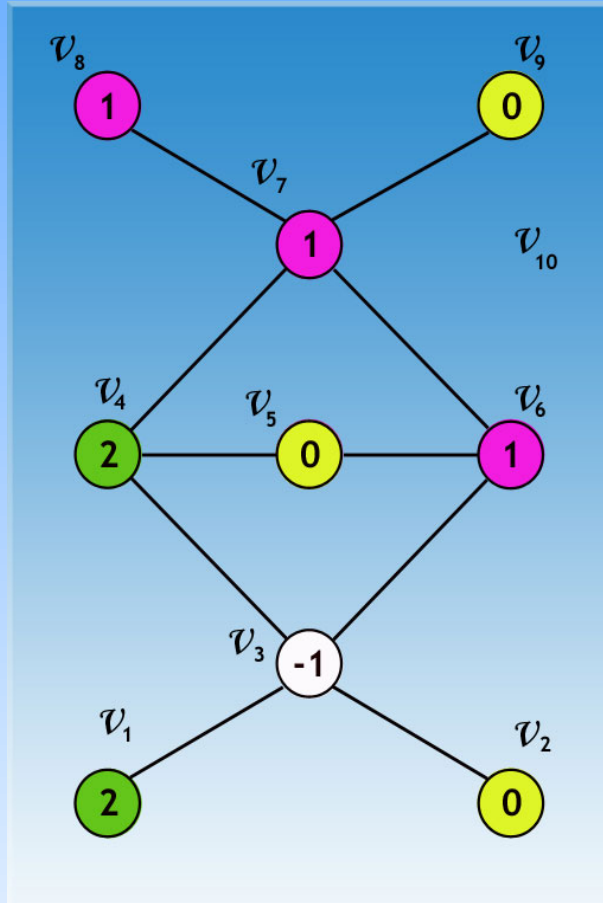
⇒ nei

# Eksempel SSA Fargelegging



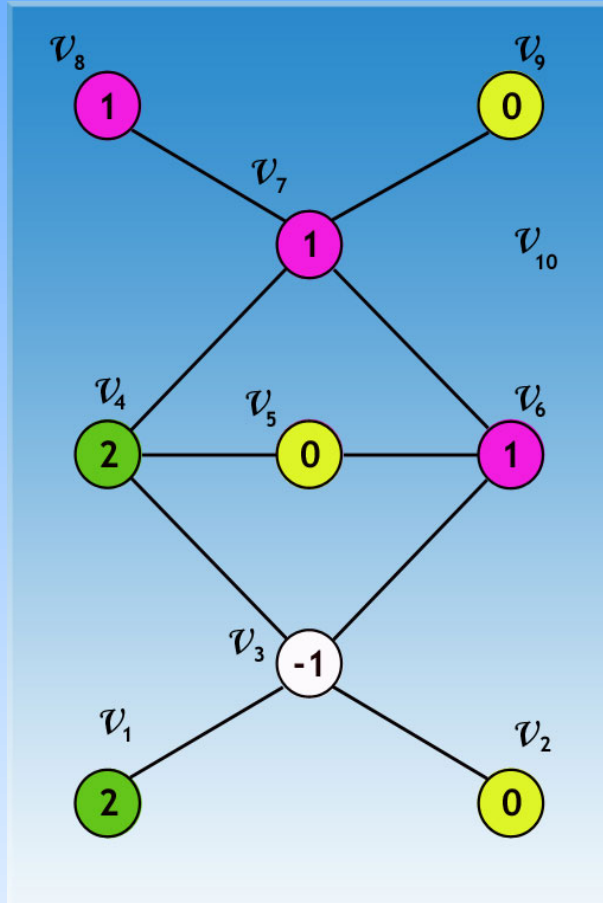
- ⇒ Noden  $v_3$  utfører handlingen.
- ⇒ Hvis(det er ledige farger)
- ⇒ nei ⇒ velg -1 (ingen farge).

# Eksempel SSA Fargelegging



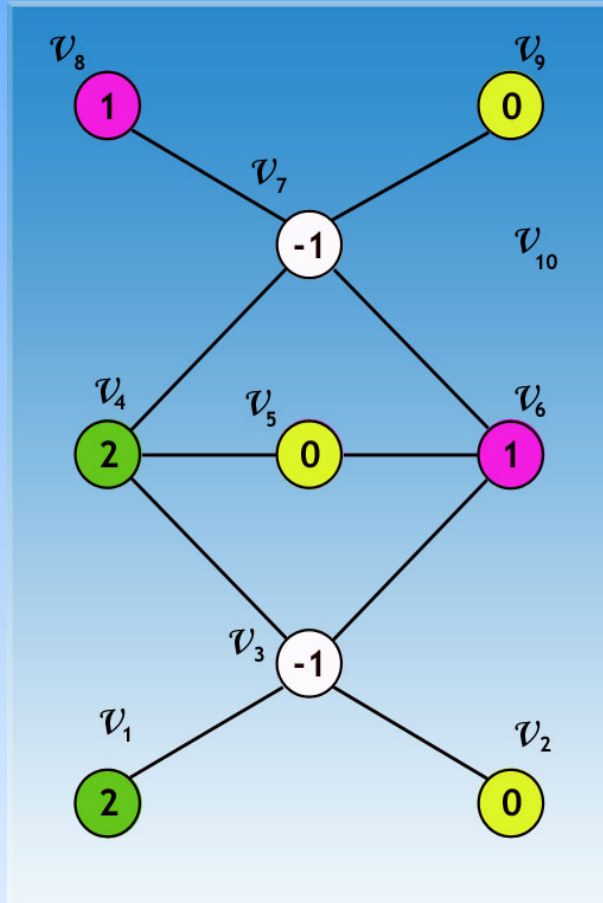
- ⇒ Noden  $v_3$  utfører handlingen.
- ⇒ Hvis(det er ledige farger)
- ⇒ nei ⇒ velg -1 (ingen farge).

# Eksempel SSA Fargelegging



- ⇒ Noden  $v_7$  utfører handlingen.
- ⇒ Hvis(det er ledige farger)
- ⇒ nei ⇒ velg -1 (ingen farge).

# Eksempel SSA Fargelegging



- ⇒ Noden  $v_7$  utfører handlingen.
- ⇒ Hvis(det er ledige farger)
- ⇒ nei  $\Rightarrow$  velg -1 (ingen farge).

# Begrensninger til Algoritmen

- ⇒ Ikke alle får en farge/frekvens.
- ⇒ Algoritmen har lineær handlingstid (handlingstid = kjøretid, antall handlinger som utføres).

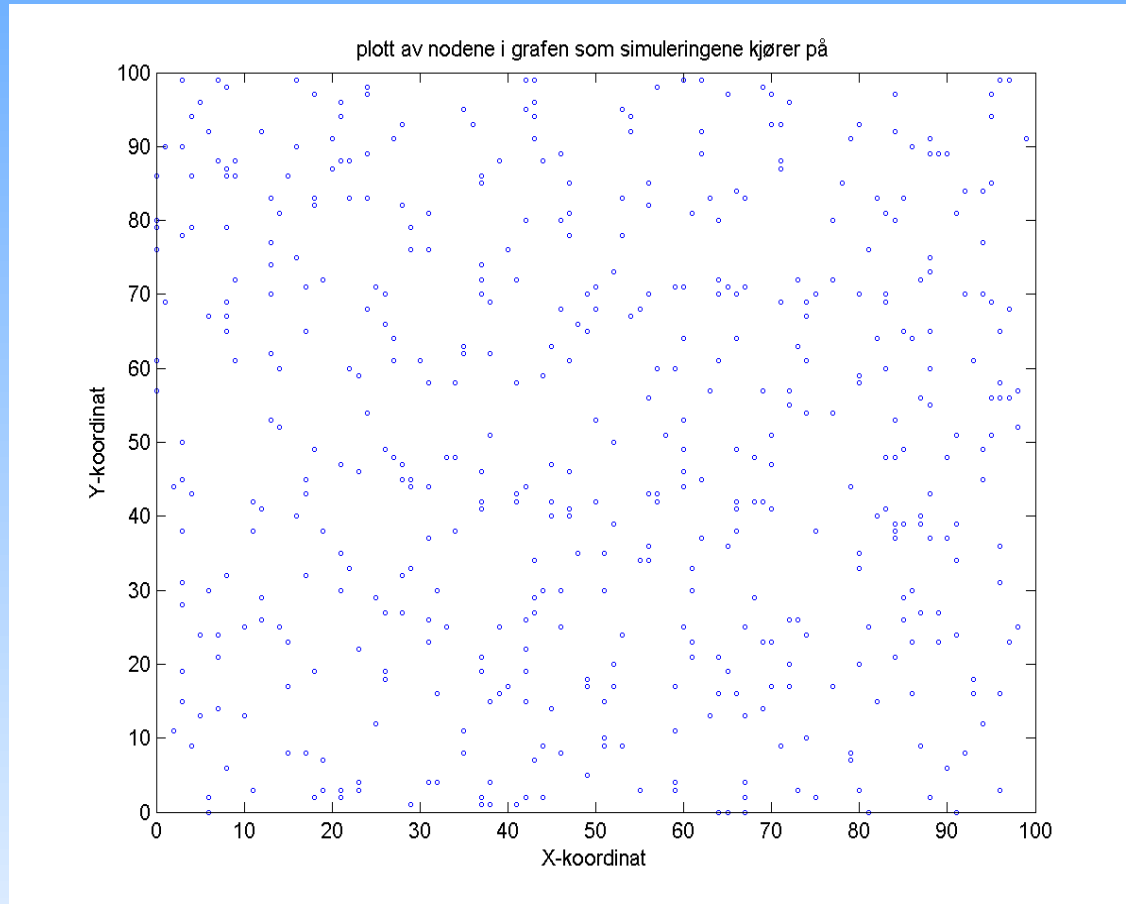
# Ulike initielle fordelinger

- ⇒ Uniform fordeling av de tillatte fargene.
- ⇒ Uniform fordeling av de tillatte fargene inkl. -1 (ingen farge).
- ⇒ Alle nodene har den samme fargen.
- ⇒ Alle nodene har -1 (ingen farge).

# Ulike initielle fordelinger

- ⇒ Ulike antall tillatte farger  
2, 3, 4, 5, 7, 9, 11 og 13.
- ⇒ Ulik radius (sendeeffekt) til enhetene/nodene  
2, 3, 4, 5, 6, 7, 8 og 9.

# Plott av nodene i grafen



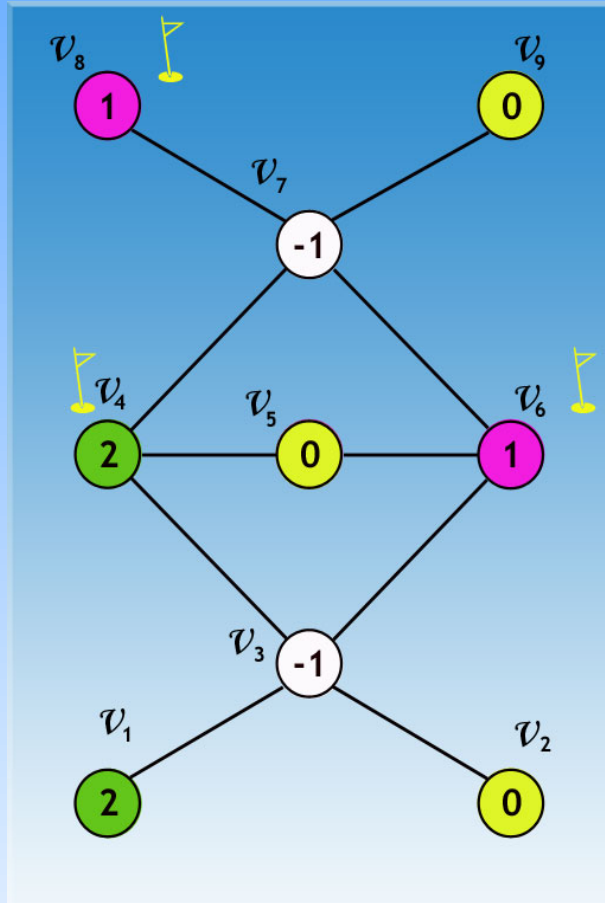
# Betydningen av initiell fordeling

- ⇒ Har ikke betydning for antall noder som blir ufarget (-1 fargen).
- ⇒ Ved liten radius blir det utført færrest regler ved en uniform fordeling.
- ⇒ Ved stor radius er det best å tildele alle nodene -1 (ingen farge) initielt.

# Ny Selvstabiliserende algoritme

- ⇒ Vi så i eks. til  $MRF$  at det var to noder som ikke fikk farge.
- ⇒ Kan vi gjøre dette bedre? Finnes det en måte å forespørre naboer om fargeendring?
- ⇒ Vi trenger da flere variabler.

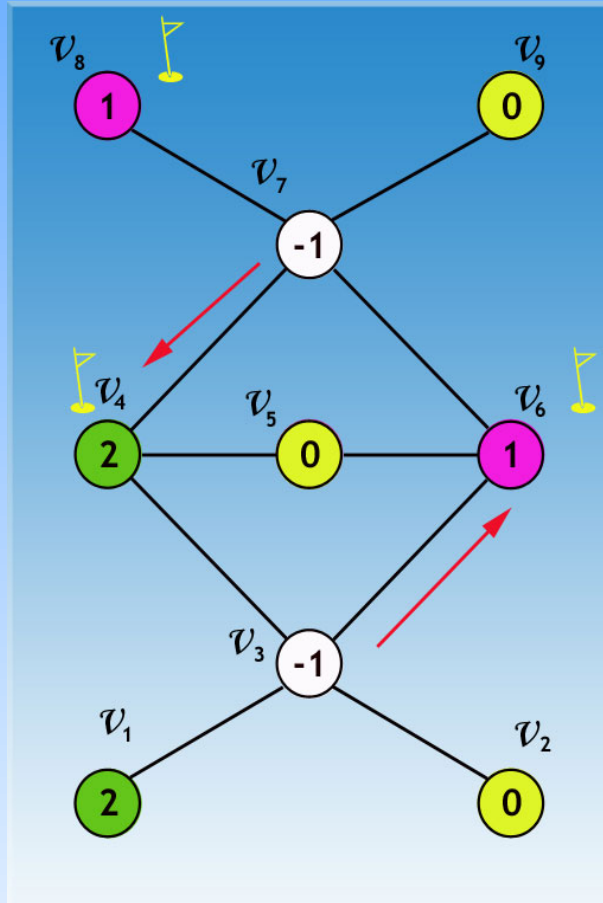
# Variabler som nodene må ha



Nodene må ha variabler som indikerer at de har ledige farger.

⇒ Flagg.

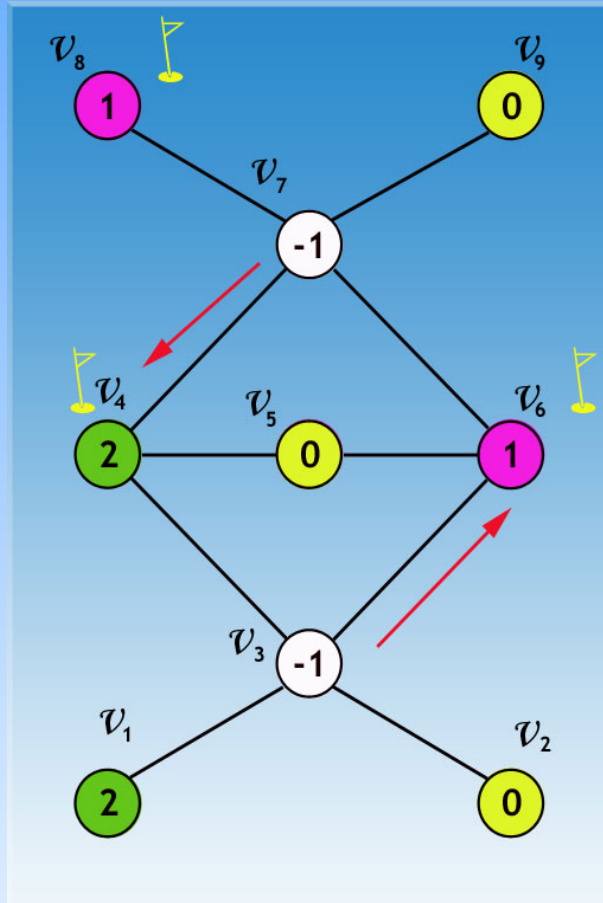
# Variabler som nodene må ha



Nodene må ha mulighet til å forespørre en nabo om fargeendring.

⇒ Peker.

# Variabler som nodene må ha

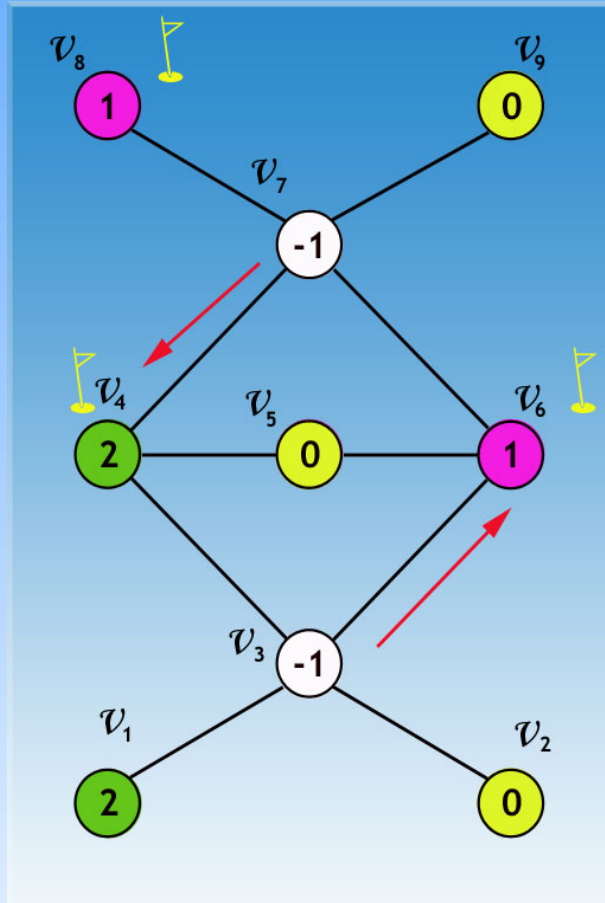


Nodene må ha mulighet til å forespørre en nabo om fargeendring.

⇒ Peker.

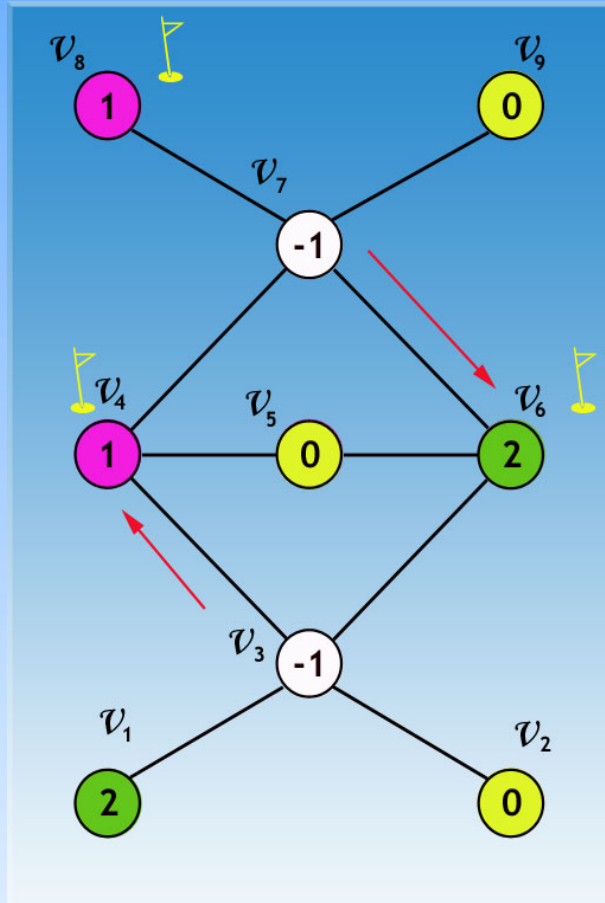
De som skal forespørres, må være **unike**

# Ser noen problemet?



⇒ Når de det pekes på bytter farge.

# Ser noen problemet?



- ⇒ Når de det pekes på bytter farge.
- ⇒ Vil pekerne flytte seg.

# Ser noen problemet?

- ⇒ Alle nodene tildeles en unik *id*. Dette er det samme som å nummerere nodene fra 1 til  $n$ .

# Ser noen problemet?

- ⇒ Alle nodene tildeles en unik *id*. Dette er det samme som å nummerere nodene fra 1 til  $n$ .
- ⇒ Når nodene endrer farge, blir flagget tatt ned.

# *Peker Flagg*–algoritmen (*PF*)

(en ny selvstabiliserende algoritme)

⇒ Har 7 ulike regler.

# *Peker Flagg*–algoritmen (*PF*)

(en ny selvstabiliserende algoritme)

- ⇒ Har 7 ulike regler.
- ⇒ De to første er en vanlig fargeleggingsalgoritme.

# *Peker Flagg*–algoritmen (*PF*)

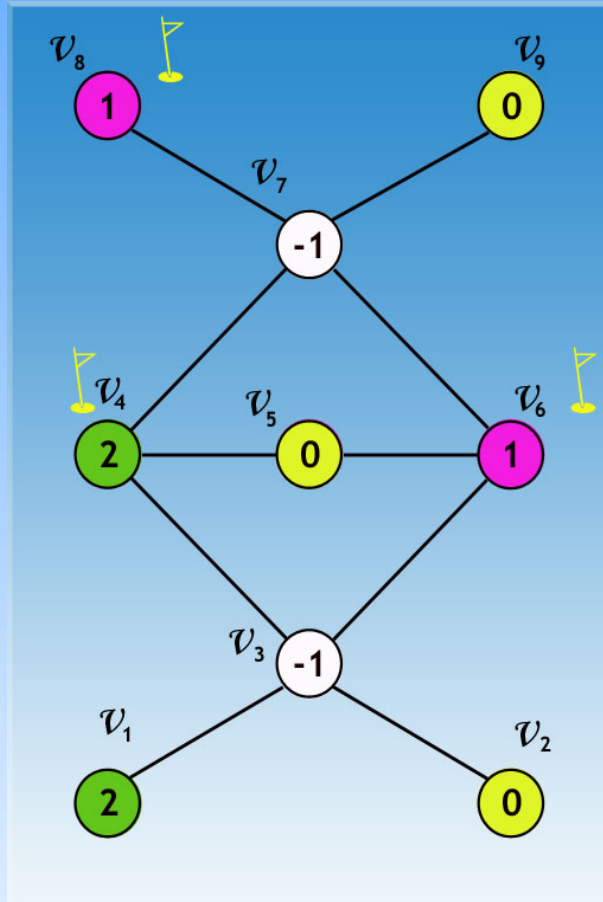
(en ny selvstabiliserende algoritme)

- ⇒ Har 7 ulike regler.
- ⇒ De to første er en vanlig fargeleggingsalgoritme.
- ⇒ De 5 andre er utelukkende med for å redusere antall ufargede noder

# *Peker Flagg*–algoritmen (*PF*)

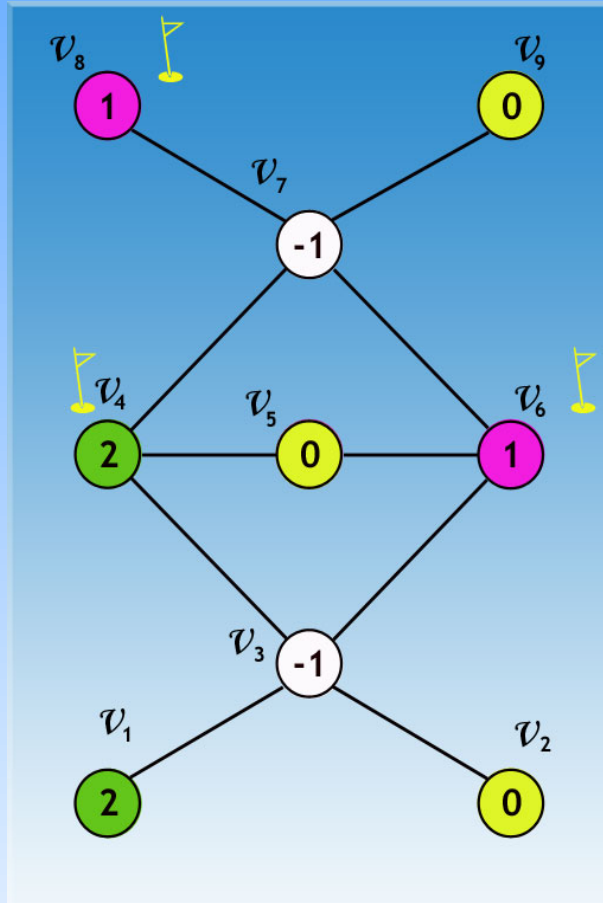
- ⇒ Kan bevise at handlingstiden er  $\mathcal{O}(m + (\delta_1 + \delta_2)n^2)$ 
  - ⇒  $m$  = antall kanter.
  - ⇒  $n$  = antall noder.
  - ⇒  $\delta_2$  = gj. antall naboer på avstand 2.
  - ⇒  $\delta_1$  = gj. antall naboer på avstand 1 (gj. gradtall).
- ⇒ Handlingstiden er verste tilfelle. Simuleringer viser at den er raskere i praksis.

# Eksempel med $PF$ -algoritmen



Nodene  $v_4$ ,  $v_6$ , og  $v_8$  har tatt opp flagget.

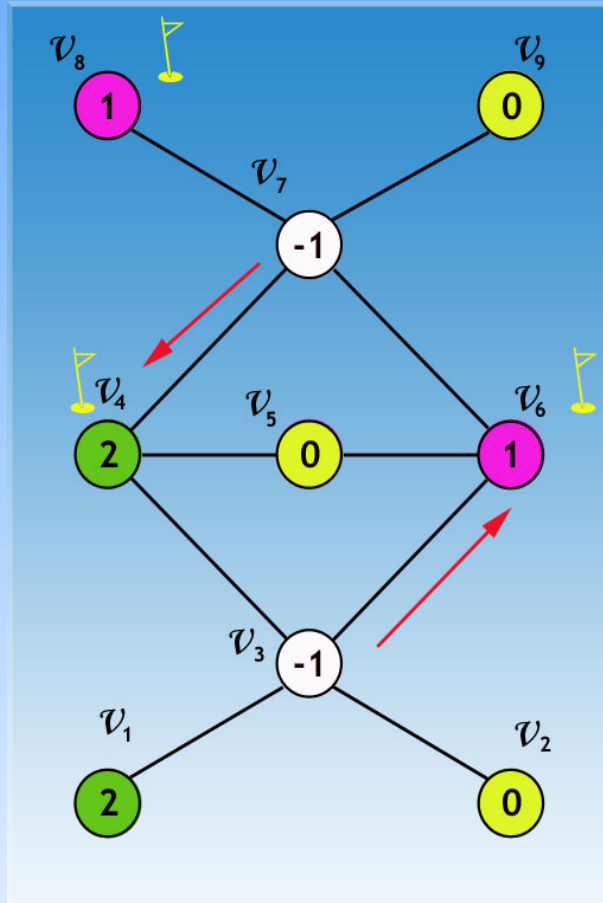
# Eksempel med $PF$ -algoritmen



Nodene  $v_4$ ,  $v_6$ , og  $v_8$  har tatt opp flagget.

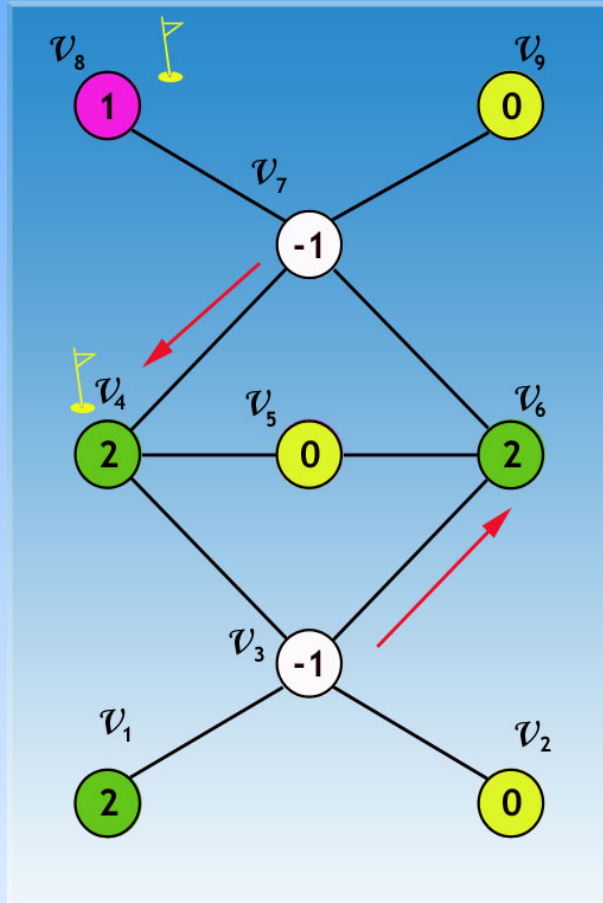
Anta at de andre nodene har naboer som blokkerer farger.

# Eksempel med $PF$ -algoritmen



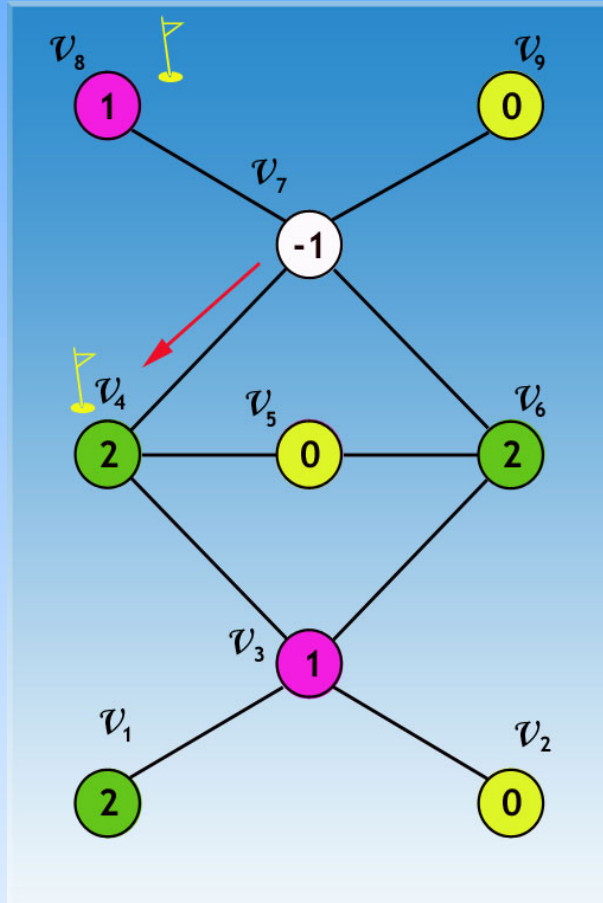
Nodene  $v_3$  og  $v_7$  peker på unike naboer.

# Eksempel med $PF$ -algoritmen



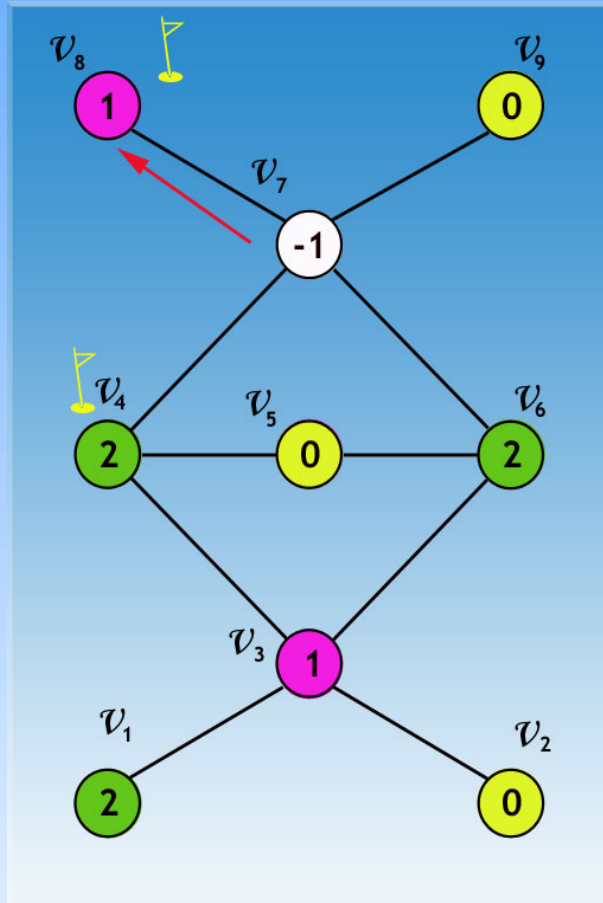
Siden  $v_6$  har den med lavest *id* pekende på seg, bytter den farge og tar ned flagget.

# Eksempel med $PF$ -algoritmen



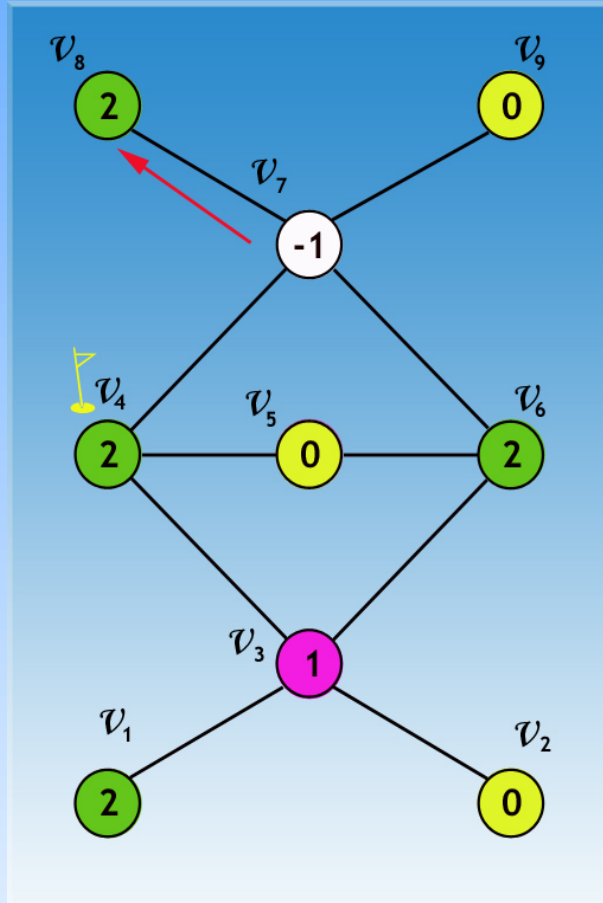
Node  $v_3$  har byttet til ledig farge og tatt vekk pekeren.

# Eksempel med $PF$ -algoritmen



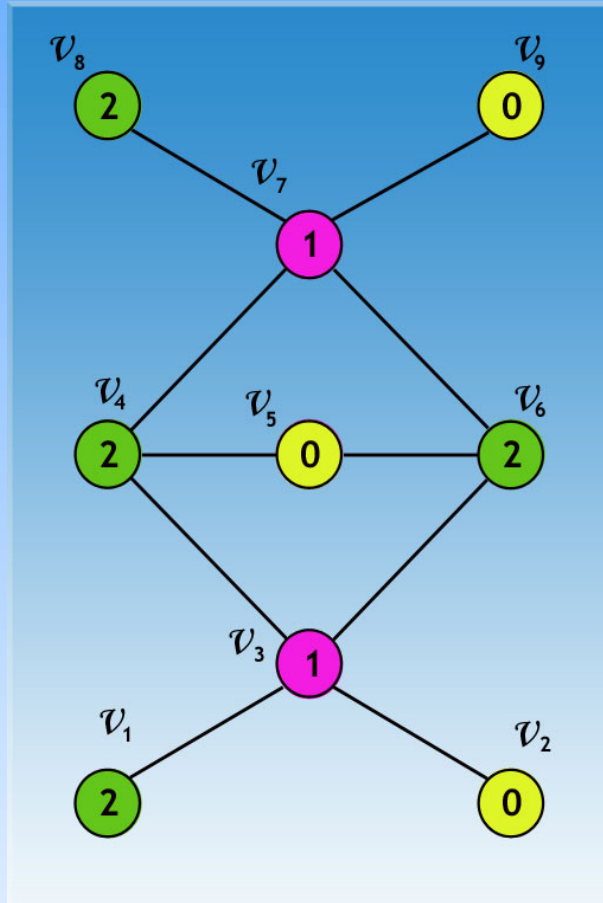
Node  $v_4$  er ikke unik for  $v_7$ , så  $v_7$  flytter pekeren til en annen unik nabo  $v_8$ .

# Eksempel med $PF$ -algoritmen



Node  $v_8$  bytter farge og tar ned flagget.

# Eksempel med $PF$ -algoritmen



Node  $v_7$  bytter farge og tar vekk pekeren.

Node  $v_4$  tar ned flagget.

# Ulike initielle fordelinger

Vi ser her det samme som ved *MRF* :

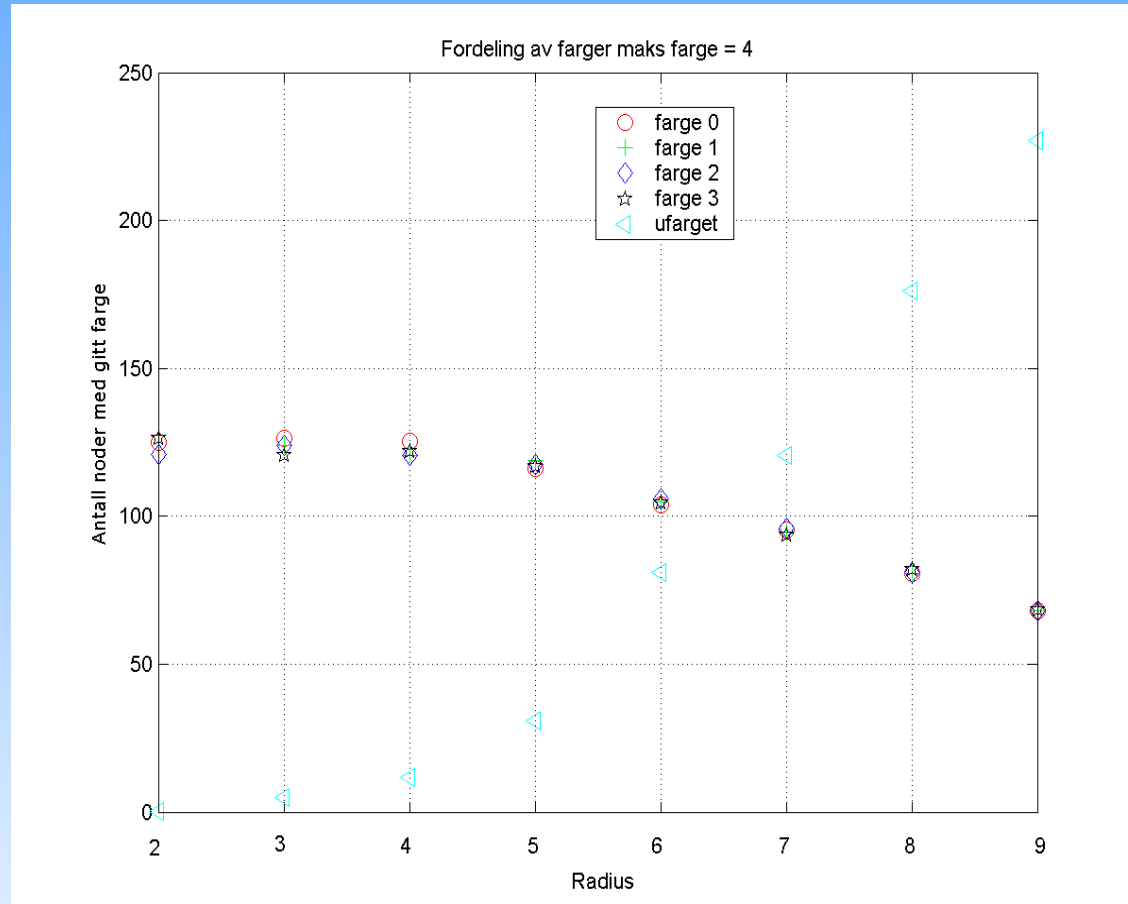
- ⇒ Har ikke betydning på antall ufargede noder.
- ⇒ Uniform fordeling bruker minst regler ved liten radius.
- ⇒ Alle noder har -1 (ingen farge) er best ved stor radius.

# Er *Peker Flagg* bedre enn *MRF*?

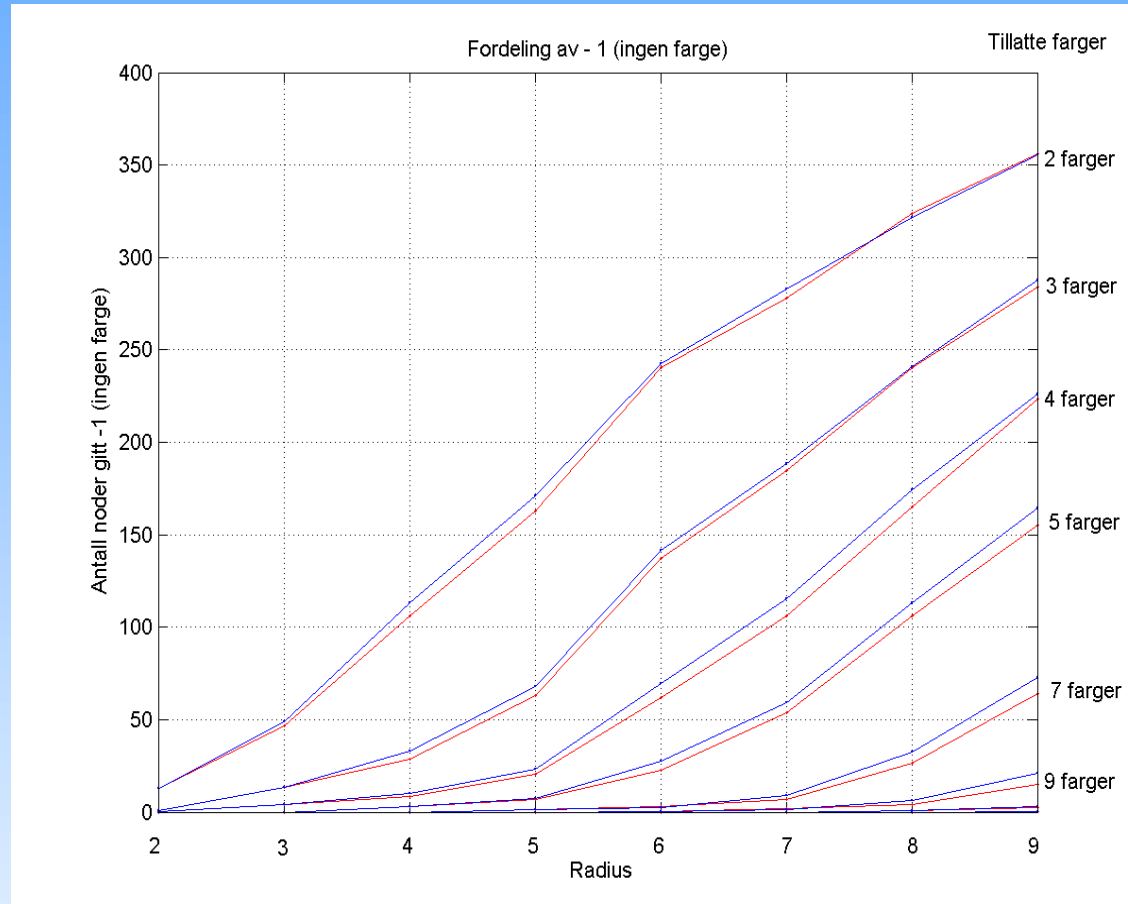
Simuleringer viser at:

- ⇒ Når radien ligger mellom 4 og 8, og antall tillatte farger er mellom 3 og 9, har *PF* ca 9,81 % færre ufargede noder enn *MRF*
- ⇒ Ved 9 tillatte farger og radius mellom 6 og 9, har *PF* i snitt 36,46 % færre ufargede noder. (begge velger laveste ledige farge).

# Plott av en simulering



# Sammenligning av $PF$ og $MRF$



# Konklusjon

- ⇒  $PF$ -algoritmen er like bra eller bedre enn  $MRF$ .
- ⇒ Den initielle fordelingen har ikke betydning for antall ufargede noder.
- ⇒ Antall enheter som kan kommunisere, er avhengig av antall tillatte farger, sendeeffekten og tettheten til enhetene.

## Videre arbeid. . .

- ⇒ Hvorfor ser det ut til å være bedre å velge den laveste ledige fargen og ikke en tilfeldig farge?
- ⇒ Hva er den forventede handlingstiden til  $PF$ -algoritmen?
- ⇒ Vil det bli bedre om en har mulighet til å forespørre flere av naboene samtidig?

# Slutt

Hovedfagsoppgaven og denne presentasjonen kan lastes ned her:

<http://www.nowires.org/FormerStudents.html>