

AI

Kunstmatige Intelligentie (AI)

Hoofdstuk 6 van Russell/Norvig = [RN]
Constrained Satisfaction Problemen (CSP's)

voorjaar 2011 — College 8, 29 maart 2011

www.liacs.nl/home/kosters/AI/

Bij **Constraint Satisfaction Problems** (CSP's) gaat het om problemen waarbij de mogelijke toestanden worden gedefinieerd door toekenningen aan variabelen X_i (met waarden uit een domein D_i ; $1 \leq i \leq n$); de doeltest is een verzameling **constraints** die aangeven welke combinaties van waarden voor deelverzamelingen van de variabelen zijn toegestaan. Dit zijn dus speciale zoekproblemen.

Standaard voorbeeld: kleur een landkaart met een beperkt aantal kleuren, zodat aangrenzende landen verschillende kleuren hebben.

Variabelen:

$X_1 = WA, X_2 = NT,$

$X_3 = Q, X_4 = NSW,$

$X_5 = V, X_6 = SA$

and $X_7 = T.$

Domeinen:

$D_i = \{\text{rood}, \text{groen}, \text{blauw}\}$

voor $i = 1, 2, \dots, 7.$

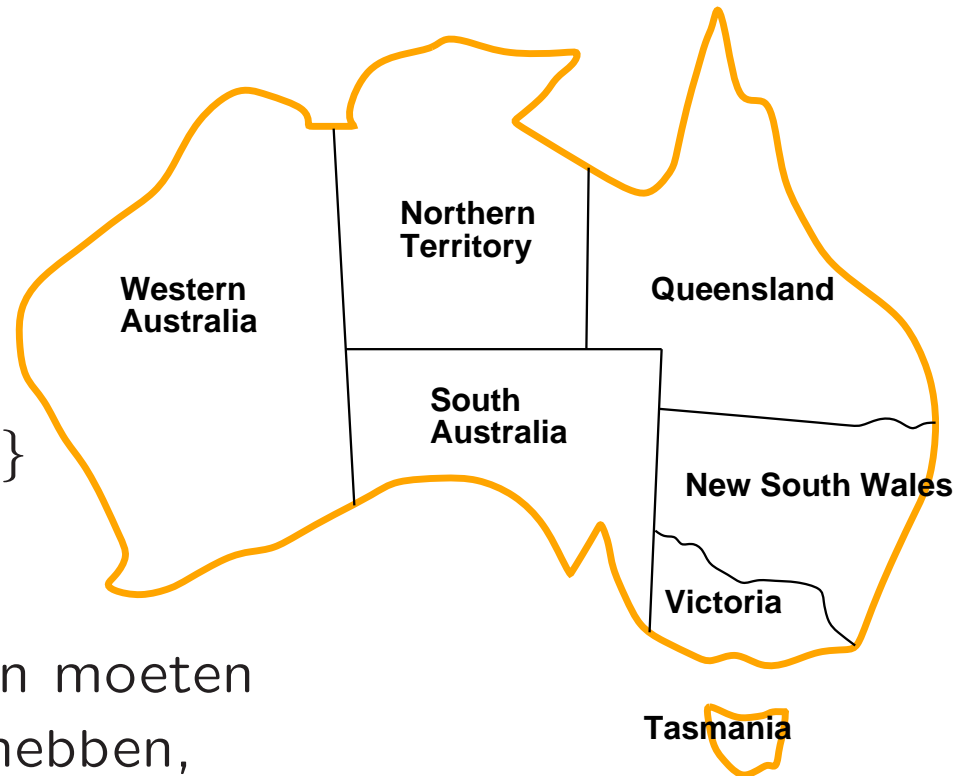
Constraints:

aangrenzende gebieden moeten

verschillende kleuren hebben,

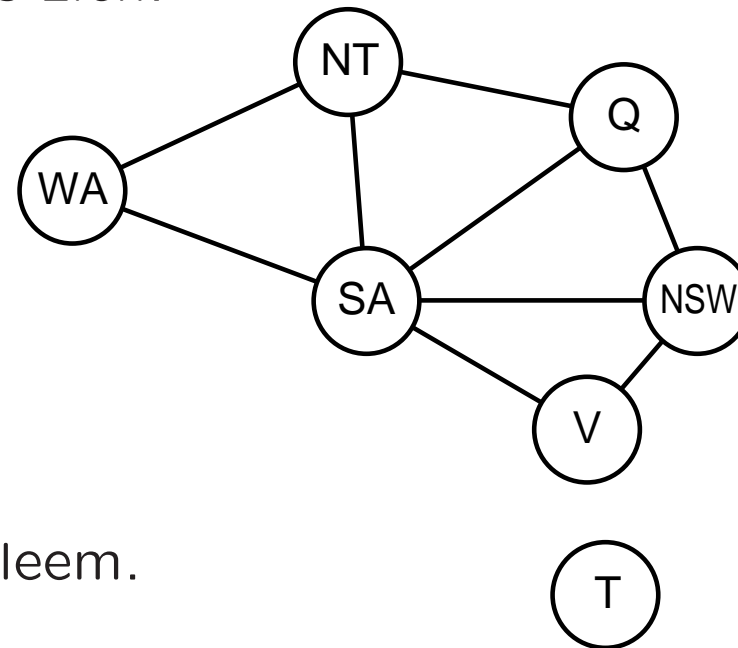
oftewel: $WA \neq NT, \dots$ (als de "taal" dat toelaat)

oftewel: $(WA, NT) \in \{(\text{rood}, \text{groen}), (\text{groen}, \text{blauw}), \dots\}, \dots$



In een **binair CSP** heeft elke constraint betrekking op maximaal twee variabelen.

De **constraint graaf** heeft de variabelen als knopen, en de takken laten de constraints zien.



Tasmanië vormt overigens een onafhankelijk deelprobleem.

Discrete variabelen:

Eindige domeinen, bijvoorbeeld Boolese CSP's, waaronder het NP-volledige SAT (Satisfiability); als alle domeinen $\leq d$ elementen hebben, zijn er $O(d^n)$ volledige toekenningen.

Oneindige domeinen, bijvoorbeeld job-scheduling; noodzaak voor een speciale “constraints-taal”: $StartJob_1 + 5 \leq StartJob_3$; lineaire constraints: oplosbaar, niet-lineair: onbeslisbaar.

Continue variabelen:

Bijvoorbeeld tijden voor waarnemingen met de Hubble-telescoop; lineaire constraints oplosbaar in polynomiale tijd met Lineair Programmeren.

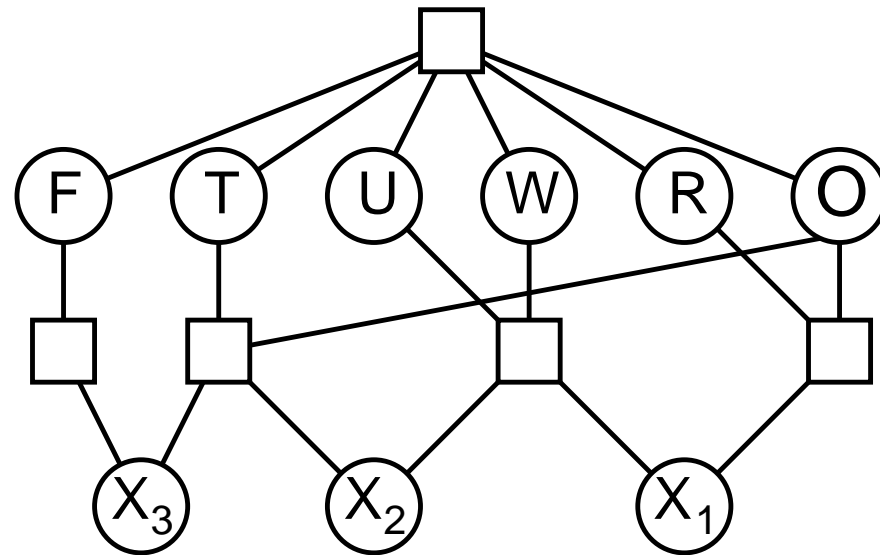
unaire constraints hebben betrekking op één variabele, bijvoorbeeld $SA \neq \textit{groen}$; je kunt ze wegwerken door de domeinen aan te passen

binaire constraints hebben betrekking op paren variabelen, bijvoorbeeld $SA \neq WA$

hogere orde constraints hebben betrekking op 3 of meer variabelen, bijvoorbeeld “rekenpuzzels” (zie straks)

voorkeuren — soft constraints, bijvoorbeeld *groen* is beter dan *rood* → “constrained optimization problems”

$$\begin{array}{r}
 T W O \\
 + T W O \\
 \hline
 F O U R
 \end{array}$$



hypergraaf

Variabelen: $F, T, U, W, R, O, X_1, X_2, X_3$

Domeinen: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints: $F \neq T, F \neq U, \dots$ ($Alldiff(F, T, U, W, R, O)$)

$O + O = R + 10 \cdot X_1, W + W + X_1 = U + 10 \cdot X_2,$

$T + T + X_2 = O + 10 \cdot X_3, X_3 = F$

- Toekenningsproblemen: wie geeft welke les?
- Roosterproblemen: welke les wordt wanneer en waar gegeven?
- Configuratie van hardware
- Spreadsheets
- Logistieke problemen

NB Vaak zijn de variabelen *reëelwaardig*.

De meest voor de hand liggende (“incrementele”) aanpak is de volgende. Toestanden worden gedefinieerd door de tot dan toe toegekende waarden. Begintoestand: de “lege” toekenning \emptyset . Doeltest: de huidige toekenning is compleet (alle n variabelen OK). Opvolger-functie: geef waarde aan “vrije” variabele, zó dat er geen conflicten optreden.

Elke oplossing zit op diepte n ; we kunnen dus DFS gebruiken. Het pad is irrelevant. Helaas: de vertakkingsgraad b op nivo ℓ is $(n - \ell)d$ (d is de (maximale) domeingrootte), en we krijgen een boom met $n! \cdot d^n$ bladeren ... terwijl er slechts d^n complete toekenningen zijn. Dit komt door **commutativiteit**: of je eerst aan X_1 toekent of eerst aan X_2 maakt niet uit!

We kunnen ons (dus) beperken tot toekenningen aan één variabele per knoop. In de wortel heb je dan d mogelijkheden in plaats van nd — als je tenminste een variabele hebt weten te kiezen. Dus $b = d$ en er zijn (zoals verwacht) d^n bladeren.

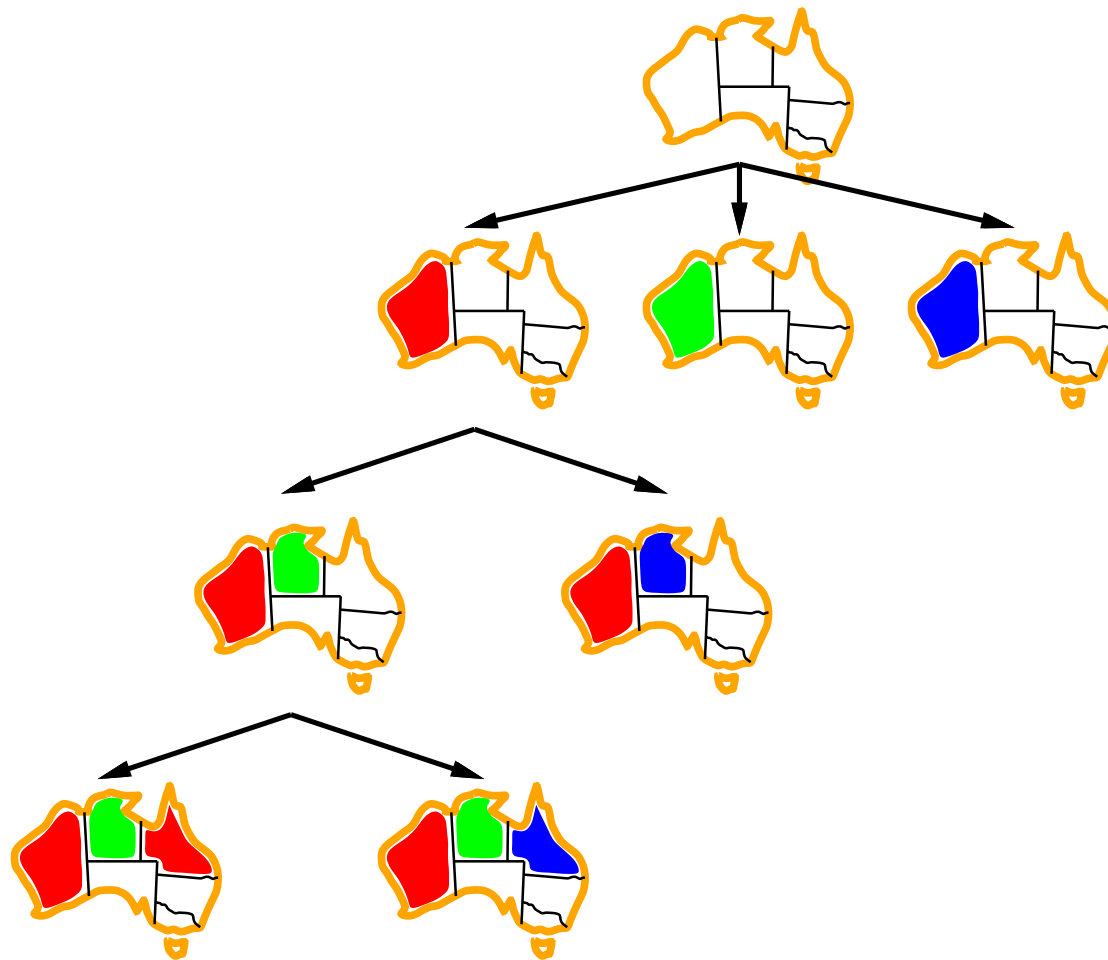
Backtracking is depth-first search voor CSP's, met toekenningen aan enkele variabelen.

Dit is *het* basisalgoritme voor CSP's. Er zijn allerlei verbeteringen mogelijk, zoals we zullen zien.

Backtracking levert de volgende recursieve functie op:

```
function RecBack (reeds, csp)
  if reeds is compleet then return reeds
  var ← KiesVrijeVar (vars, reeds, csp)
  for each waarde in Waardes (var, csp)
    if waarde is consistent met reeds
      volgens Constraints[csp] then
        res ← RecBack ( $\{var = waarde\} \cup reeds$ , csp)
        if res ≠ failure then return res
  return failure
```

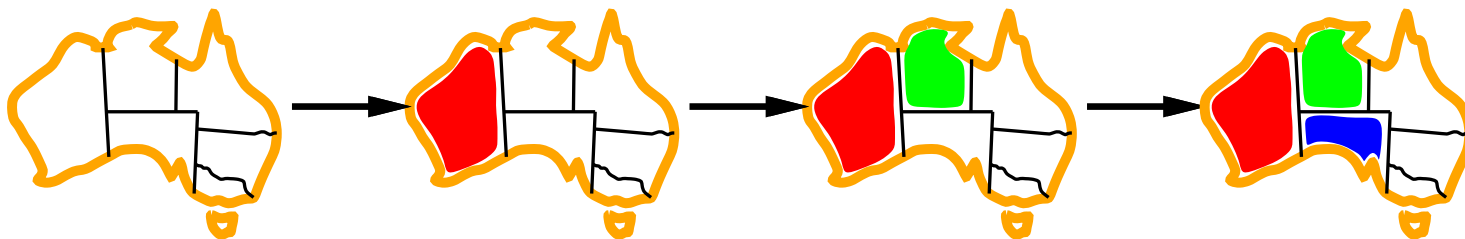
Hierbij is *reeds* de verzameling van de reeds toegekende variabelen en hun waarden ($\{WA = \textit{rood}, T = \textit{blauw}\}$).



Drie hoofdvragen zijn:

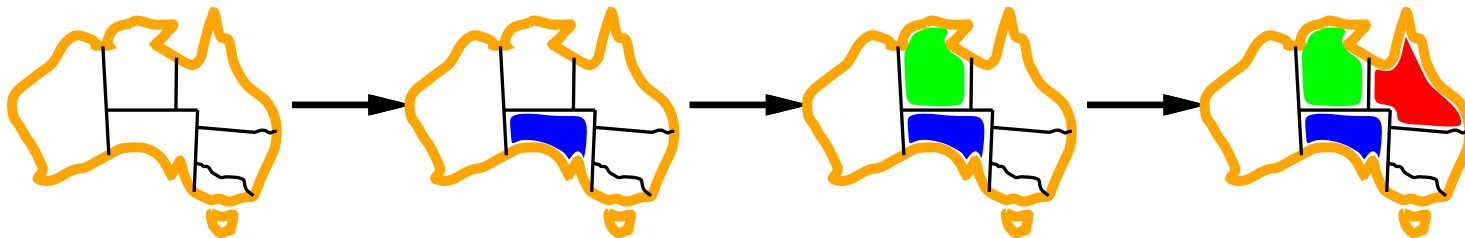
- Welke variabele moet ik eerst doen, en welke waarde kan ik hem geven?
- Wat zijn de implicaties van de huidige toekenningen voor de nog niet toegekende variabelen?
- Als een pad faalt, hoe kun je dan dit zelfde probleem in de toekomst vermijden?

De **Minimum Remaining Values (MRV)** heuristiek (= **Most Constrained Variable**) kiest de variabele met de minste toegestane waarden, en kleurt in de derde stap SA (en wel **blauw**), want SA heeft nog slechts één mogelijke kleur:



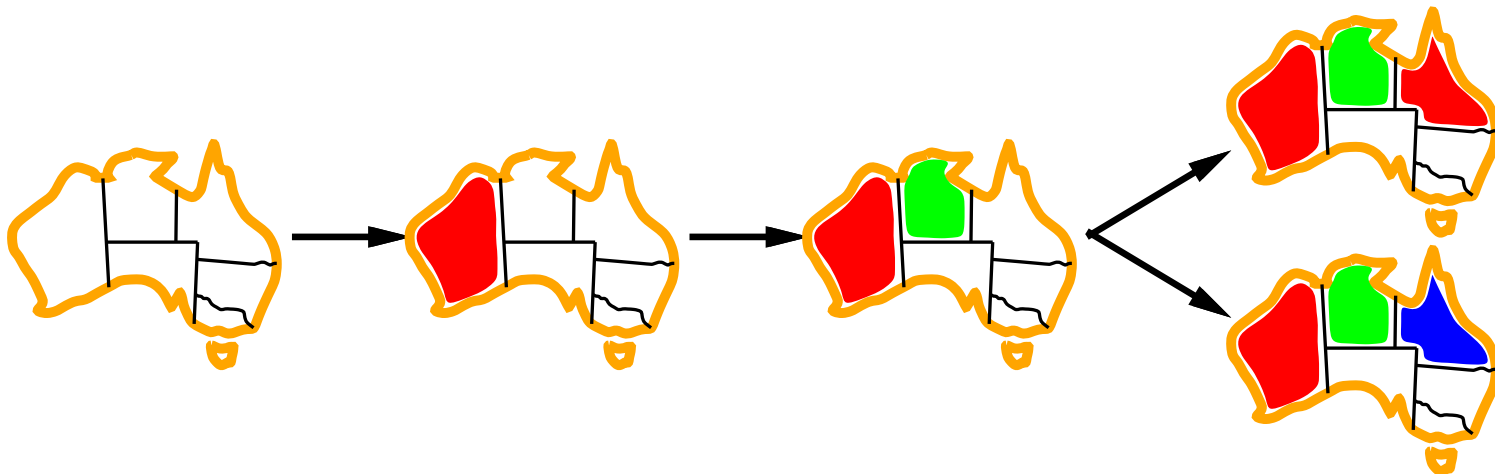
“Welke variabele moet ik kiezen?”

De **Most Constraining Variable (MCV)** heuristiek (= **degree**-heuristiek) kiest de variabele met de meeste constraints op de overblijvende variabelen, en kleurt in het begin SA:



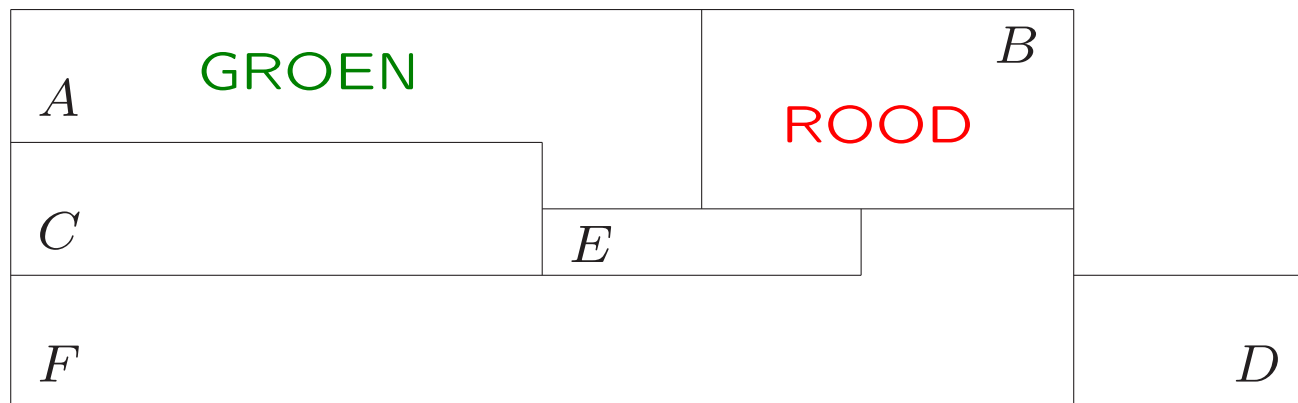
“Welke variabele moet ik kiezen?”

De **Least Constraining Value** (LCV) heuristiek kiest, gegeven een variabele, de waarde die de meeste waarden voor de overblijvende variabelen mogelijk laat, en kleurt in de derde stap Q (als je die variabele dus al gekozen hebt) **rood** zodat SA nog één mogelijkheid heeft (bij **blauw** geen!):



“Welke waarde moet ik kiezen?”

De drie voorgaande heuristieken samengevat in een voorbeeld, weer met drie kleuren:



MRV (Minimum Remaining Values): kleur *E* nu (en wel blauw)

MCV (Most Constraining Variable): kleur nu *F* (of wellicht, als je anders telt, *E*); kleur *als eerste E* of *F* (veel burens)

LCV (Least Constraining Value): *als je nu C* wilt kleuren, dan met rood

Een **Sudoku** is een CSP. We hebben in het 9×9 -geval namelijk 81 variabelen $\{X_{11}, X_{12}, \dots, X_{19}, X_{21}, \dots, X_{99}\}$, alle met domein $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Voor de reeds gevulde vakjes bestaat het domein uit het gegeven getal. De constraints eisen dat alle rijen, kolommen en de negen 3×3 -blokken precies alle getallen bevatten.

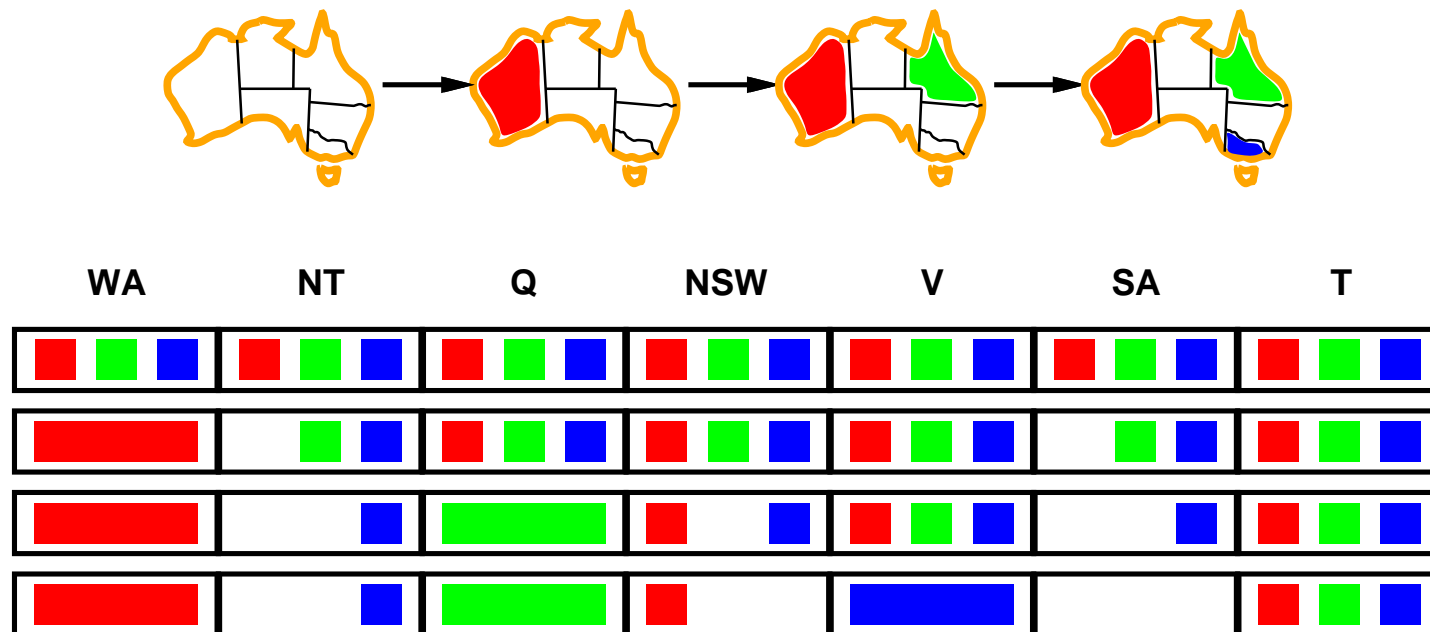
De MRV-heuristiek kiest een vakje waar het minste aantal waarden nog mogelijk is.

De MCV-heuristiek kiest een vakje dat met zoveel mogelijk andere nog “open” vakjes interfereert.

De LCV-heuristiek kiest een waarde (gegeven een vakje) die het meeste overlaat voor de andere vakjes.

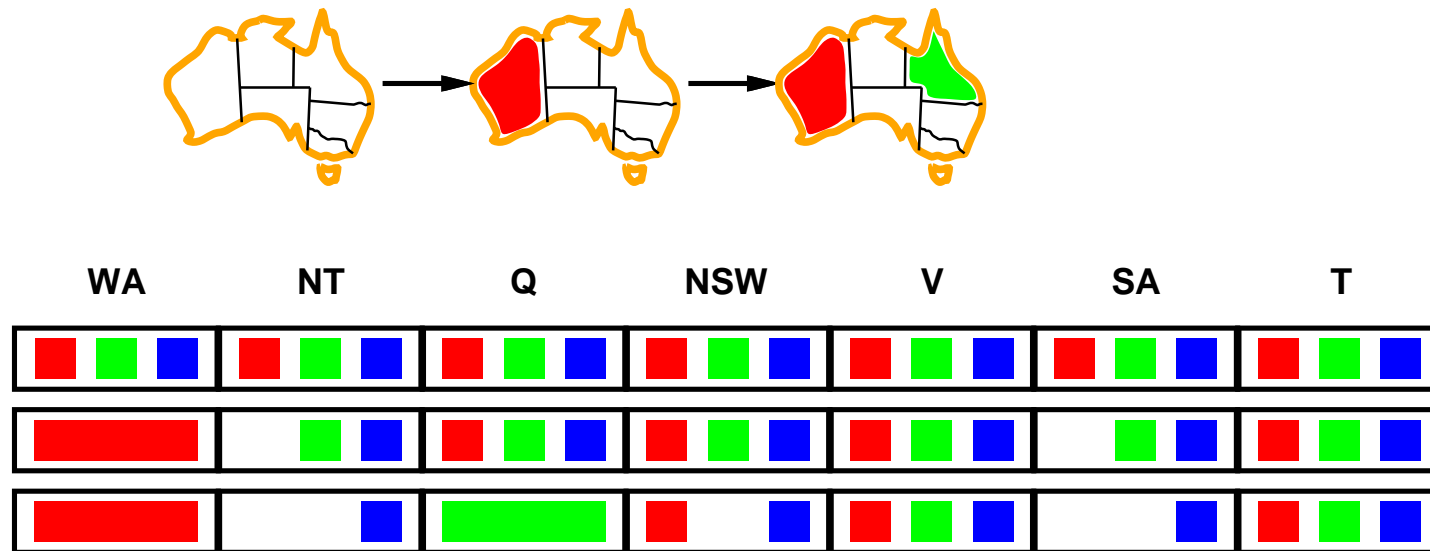
Vergelijk: Japanse puzzels (Nonogrammen), Minesweeper.

Bij **forward checking** houd je de nog toegestane waarden bij voor de nog niet toegekende (“vrije”) variabelen. Je kunt stoppen zodra er een variabele is zonder toegestane waarden.



Dit gaat goed samen met de MRV.

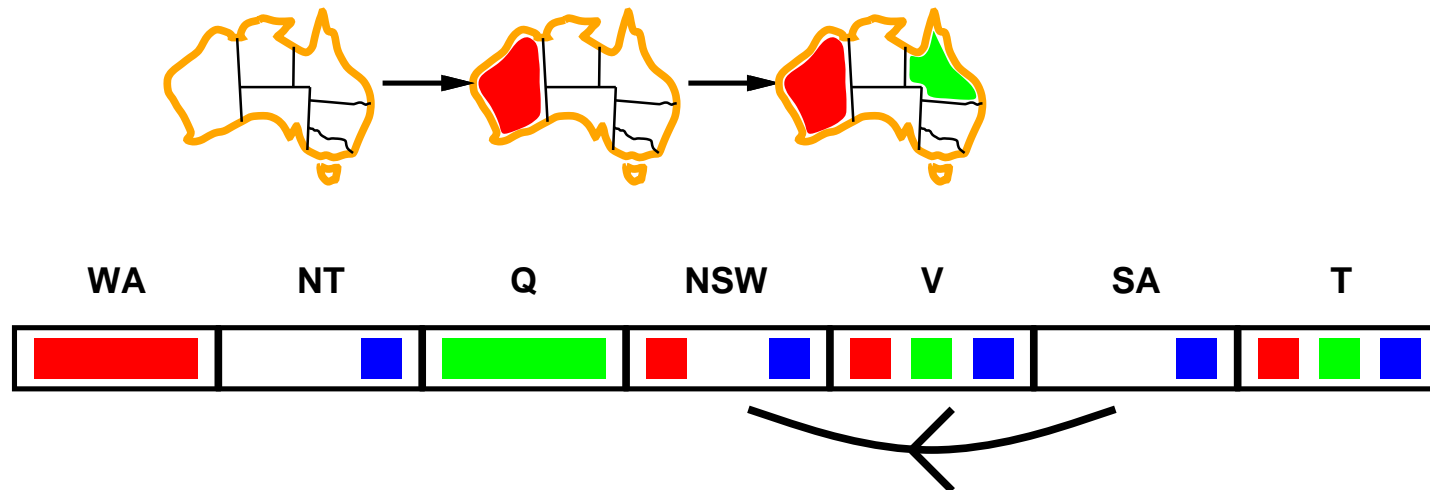
Forward checking ontdekt niet alle inconsistenties:



NT en *SA* kunnen niet beide **blauw** zijn!

Constraint propagatie probeert herhaald lokaal constraints te forceren.

De eenvoudigste propagatie-vorm maakt pijlen (“arcs”) consistent: $X \rightarrow Y$ heet **consistent** als voor elke waarde van X er nog minstens één toegestane waarde van Y is.



Deze pijl is consistent: als SA **blauw** is, kan NSW nog **rood** zijn. Andersom niet: als NSW **blauw** is, kan SA niet netjes gekleurd worden. En tussen NT en SA is geen (consistente) pijl: klaar!

Er bestaat een **arc consistency** algoritme dat meer doet dan Forward checking, zie boek. Het wordt herhaald toegepast. Per gecheckte pijl worden andere pijlen, die door het eventueel verwijderen van foute waarden inconsistent dreigen te worden, ook weer gecheckt.

Complexiteit: er zijn $O(n^2)$ pijlen, elk wordt hooguit d keer op de agenda gezet, en de check op consistentie kost $O(d^2)$, bij elkaar $O(n^2d^3)$.

Je kunt niet “alles” in polynomiale tijd detecteren (want 3-SAT is NP-volledig — zie het college Complexiteit!).

Er zijn allerlei gespecialiseerde **CSP-solvers**, en zelfs “Constraint Programming”.

Technieken als hill-climbing en Simulated Annealing werken met “complete” toestanden: alle variabelen hebben een waarde.

Voor CSP's moet je toestanden met “geschonden” constraints toestaan, en operatoren maken die variabelen van waarde wijzigen.

Je kunt random een “foute” variabele kiezen, en (met de **min-conflicts** heuristiek) die waarde kiezen die de minste constraints schendt.

Voorbeeld: dames op schaakbord, zie vorig Hoofdstuk.

Ook mogelijk: Genetische algoritmen, zie later.

Het huiswerk voor de volgende keer (dinsdag 5 april 2011): lees **Hoofdstuk 7 en 8**, p. 234–252, p. 265–267 en p. 285–306 van [RN], over het onderwerp Logische agenten. Kijk ook eens naar de vragen bij dit hoofdstuk.

Denk tevens aan de derde opgave: **Neurale netwerken**; deadline dinsdag 12 april 2011!