Kunstmatige Intelligentie (AI)

Hoofdstuk 6 van Russell/Norvig = [RN]

Constrained Satisfaction Problemen (CSP's)

voorjaar 2024

College 8, 3 april 2024

www.liacs.leidenuniv.nl/~kosterswa/AI/csps.pdf

Bij Constraint Satisfaction Problemen (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.

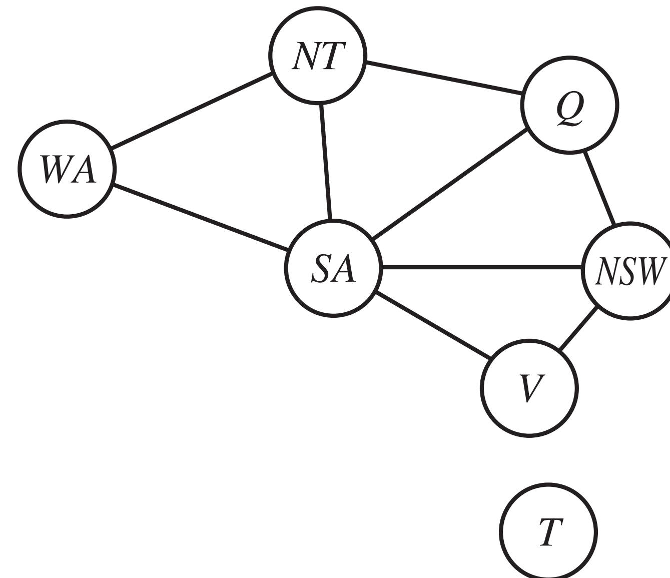
Variabelen:
$X_1 = WA$, $X_2 = NT$,
$X_3 = Q$, $X_4 = NSW$,
$X_5 = V$, $X_6 = SA$
and $X_7 = T$.

Domeinen:
$D_i = \{rood, groen, blauw\}$
voor $i = 1, 2, \ldots, 7$.

Constraints:
aangrenzende gebieden moeten verschillende kleuren hebben,
oftewel: $WA \neq NT$, . . . (als de "taal" dat toelaat)
oftewel: $(WA, NT) \in \{(rood, groen), (groen, blauw), \ldots\}, \ldots$

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 Booleaanse 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 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 $SA \neq groen$; je kunt ze wegwerken door de domeinen aan te passen

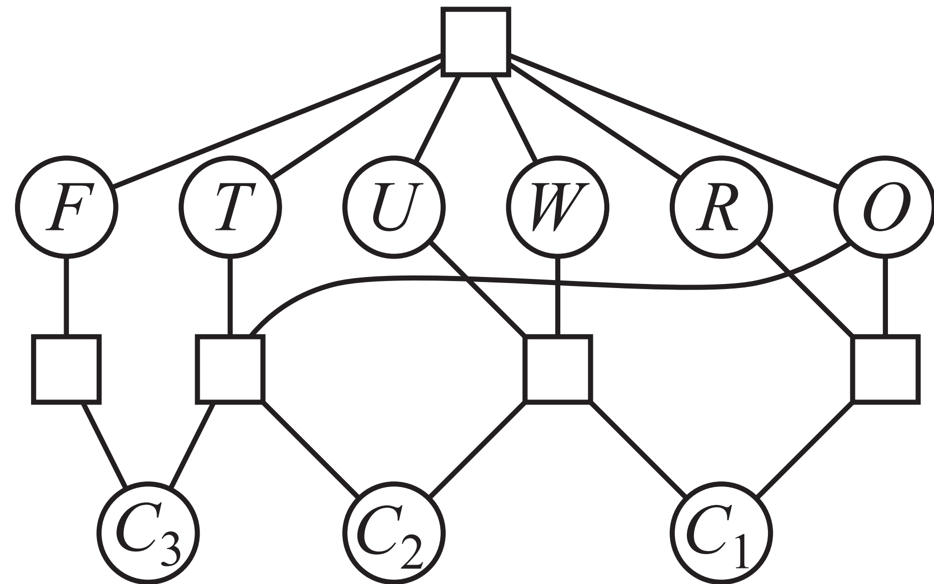
binaire constraints hebben betrekking op paren variabelen, bijvoorbeeld $SA \neq WA$

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

voorkeuren — soft constraints, bijvoorbeeld $groen$ is beter dan $rood$ $\longrightarrow$ "constrained optimization problems"

Voorbeeld — Rekenpuzzel

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



hypergraaf

Variabelen: $F, T, U, W, R, O, C_1, C_2, C_3$

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

Constraints: $F \neq T,\ F \neq U,\ \ldots\ (Alldiff(F, T, U, W, R, O))$
$O + O = R + 10 \cdot C_1,\ W + W + C_1 = U + 10 \cdot C_2,$
$T + T + C_2 = O + 10 \cdot C_3,\ C_3 = F$

Voorbeeld — Echte problemen

- 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 (de grootte van het domein) mogelijkheden in plaats van nd — als je tenminste één van de n variabelen hebt weten te kiezen. Dus vertakkingsgraad $b = d$ en er zijn (zoals verwacht) d^n bladeren.

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

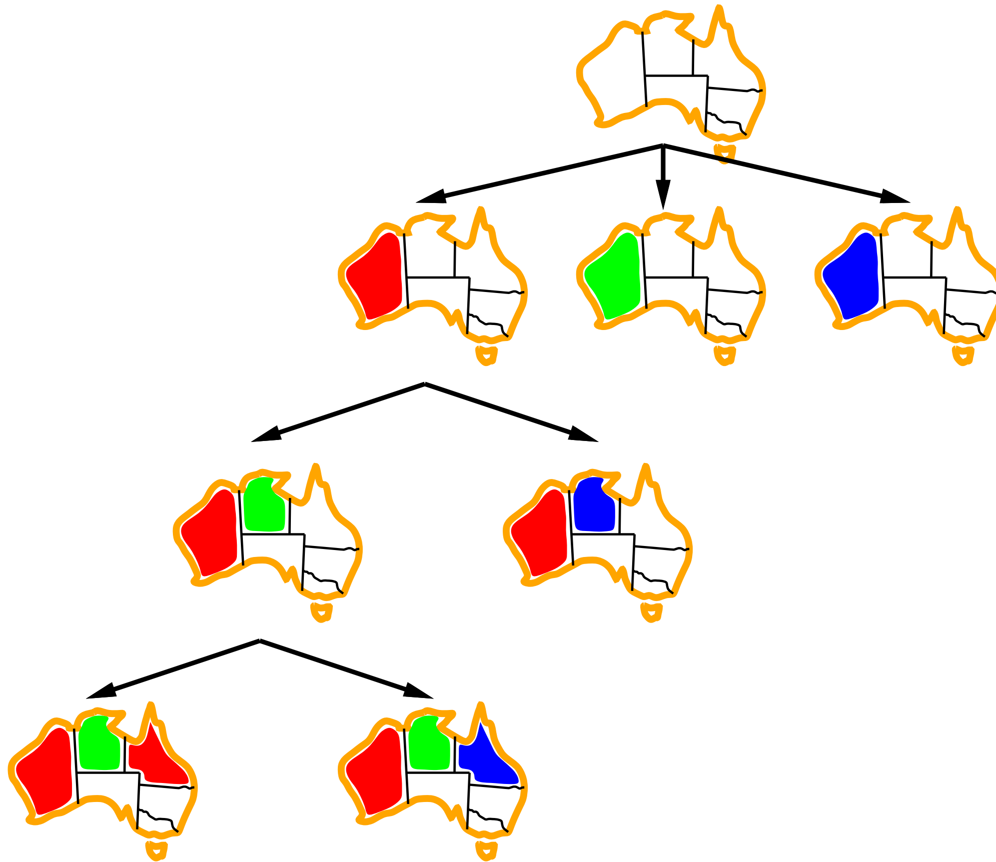
Dit is *het* basisalgoritme voor CSP’s. Er zijn allerlei verbeteringen mogelijk, zoals we zullen zien.

Backtracking levert de volgende recursieve functie op:

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

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

Backtracking — Voorbeeld



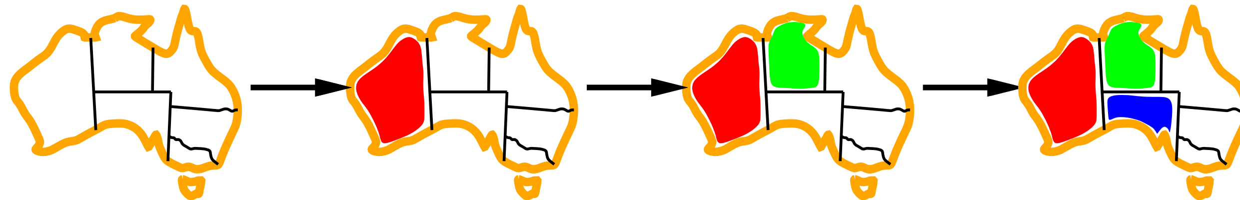
Drie hoofdvragen zijn:

- 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?

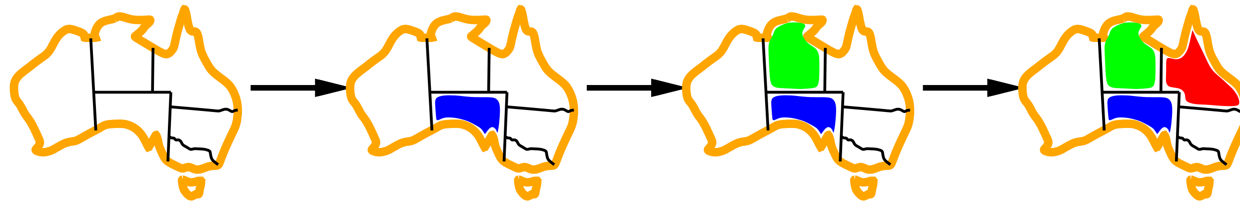
- Als een pad faalt, hoe kun je dan dit zelfde probleem in de toekomst vermijden?

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:



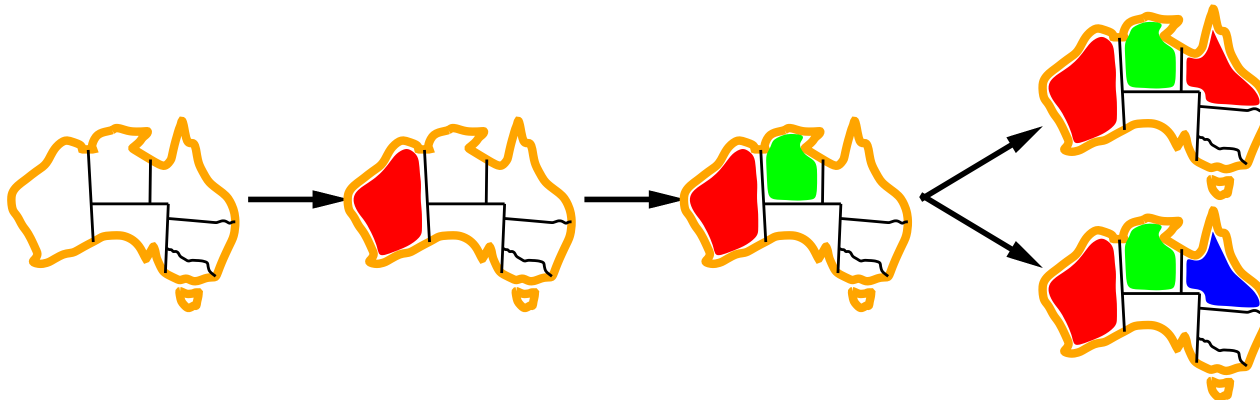
"Welke variabele moet ik kiezen?"

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*:



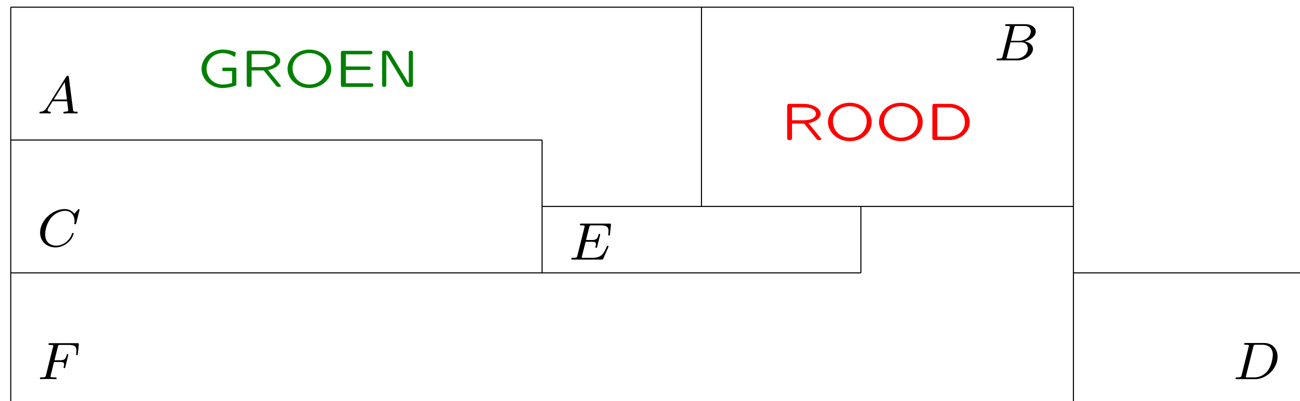
"Welke variabele moet ik kiezen?"

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!):



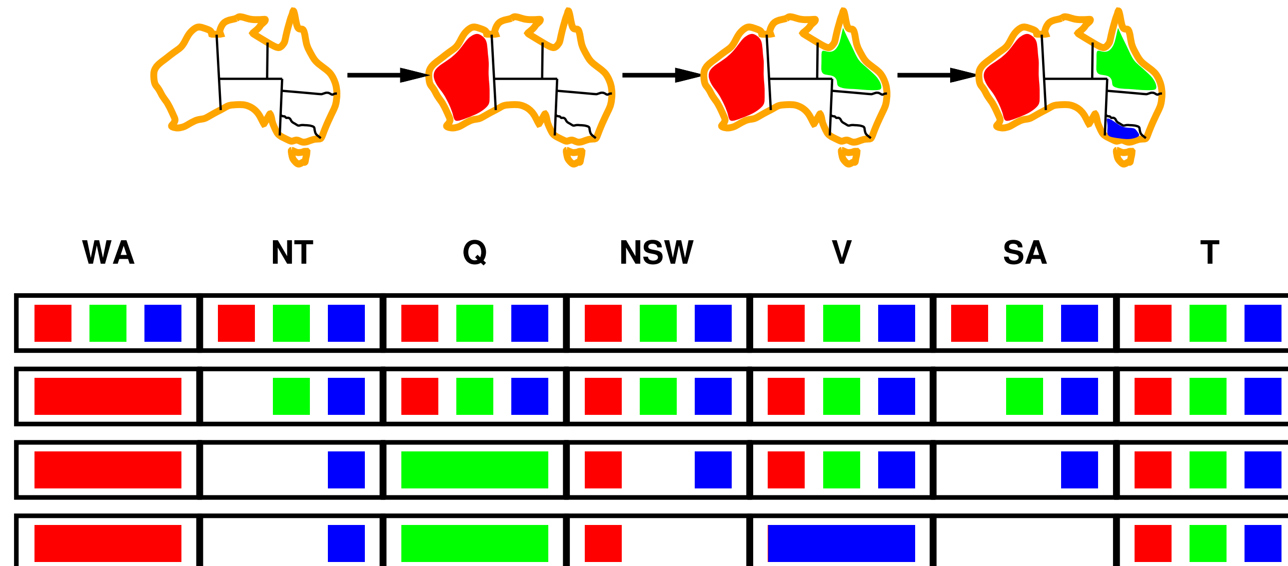
"Welke waarde moet ik kiezen?"

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



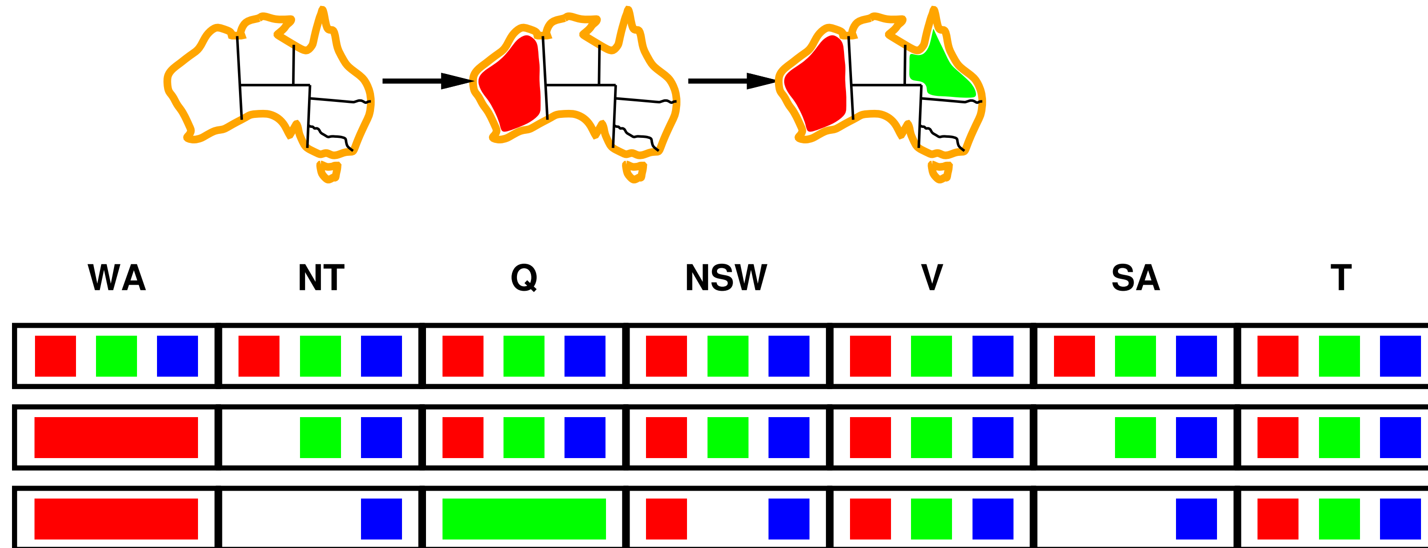
MRV (Minimum Remaining Values): kleur E nu (en wel blauw)
MCV (Most Constraining Variable): kleur *nu* F (of wellicht, als je anders telt, E); kleur *als eerste* E of F (veel buren)
LCV (Least Constraining Value): *als* je nu C wilt kleuren, dan met rood

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.

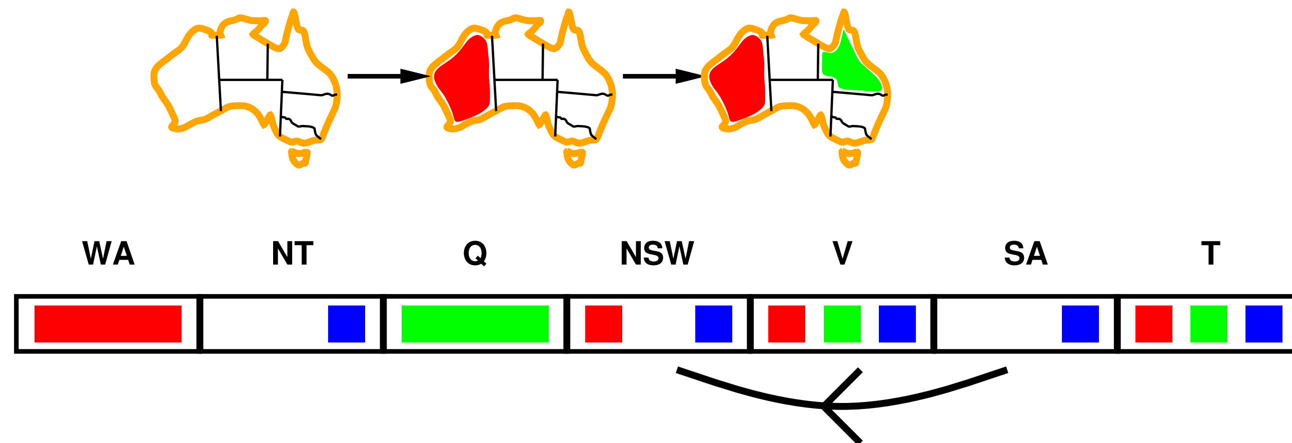
Forward checking ontdekt niet alle inconsistenties:



NT en *SA* kunnen niet beide blauw zijn!

Constraint propagatie probeert herhaald locaal constraints te forceren.

De eenvoudigste propagatie-vorm maakt pijlen ("arcs") consistent: $X \to Y$ 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 (AC-3) 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". En "SAT-solvers", zie verderop.

Een Sudoku is een CSP. We hebben in het 9×9-geval 81 variabelen $\{X_{11}, X_{12}, \ldots, X_{19}, X_{21}, \ldots, X_{99}\}$, alle met domein $D = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. 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 uit D bevatten: 27 keer *Alldiff*.

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.

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

Met arc consistency kun je afleiden dat E6 een 4 moet zijn. En I6 moet een 7 zijn.

Om een SAT-solver te kunnen gebruiken, moet het probleem eerst als Satisfiability (SAT) probleem worden geformuleerd, zie het college Complexiteit.

Gegeven zijn Booleaanse variabelen x_1, x_2, x_3, x_4 (of meer), en verder "clausules" als $C_1 = x_2 \vee \overline{x_3} \vee x_4$ en $C_2 = \overline{x_1} \vee x_3$. (Hierbij heten x_3 en $\overline{x_3} = \neg x_3$ "literals".) De vraag is of $C_1 \wedge C_2$ waargemaakt kan worden, hier bijvoorbeeld door $x_1 = x_3 = \texttt{false}$ en $x_2 = x_4 = \texttt{true}$. Dit heet wel CNF: Conjunctive Normal Form.

Het beslissingsprobleem SAT is "NP-volledig", dus lastig!

Een SAT-solver probeert op efficiënte wijze een SAT-probleem op te lossen, zie MiniSAT, Lingeling, . . .

Aan de basis ligt doorgaans het DPLL-algoritme (Davis-Putnam-Logemann-Loveland) uit 1962:

- "unit propagation" behandelt clausules met één literal
- "pure literal elimination" handelt literals af die alleen "true of false" voorkomen
- "splitting rule" probeert een variabele true/false te maken, met behulp van backtracking

Vele verbeteringen: Conflict-Driven Clause Learning, . . .

Sudoku, met $s_{xyz} = \texttt{true} \Leftrightarrow$ plek (x, y) bevat getal z:

- There is at least one number in each entry:
  $\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$
- Each number appears at most once in each row:
  $\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{xyz} \vee \neg s_{iyz})$
- Each number appears at most once in each column:
  $\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{xyz} \vee \neg s_{xiz})$
- Each number appears at most once in each 3x3 sub-grid:
  $\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=y+1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z})$
  $\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=x+1}^{3} \bigwedge_{l=1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+k)(3j+l)z})$.

- There is at most one number in each entry:
  $\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigwedge_{z=1}^{8} \bigwedge_{i=z+1}^{9} (\neg s_{xyz} \vee \neg s_{xyi})$
- Each number appears at least once in each row:
  $\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{x=1}^{9} s_{xyz}$
- Each number appears at least once in each column:
  $\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{y=1}^{9} s_{xyz}$
- Each number appears at least once in each 3×3 sub-grid:
  $\bigvee_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} s_{(3i+x)(3j+y)z}$.

In the extended encoding, the resulting CNF formula will have 11,988 clauses, apart from the unit clauses representing the pre-assigned entries. From these clauses, 324 clauses are nine-ary and the remaining 11,664 clauses are binary. The nine-ary clauses result from the four sets of at-least-

Uit: I. Lynce, J. Ouaknine, Sudoku as a SAT problem, ISAIM 2006.

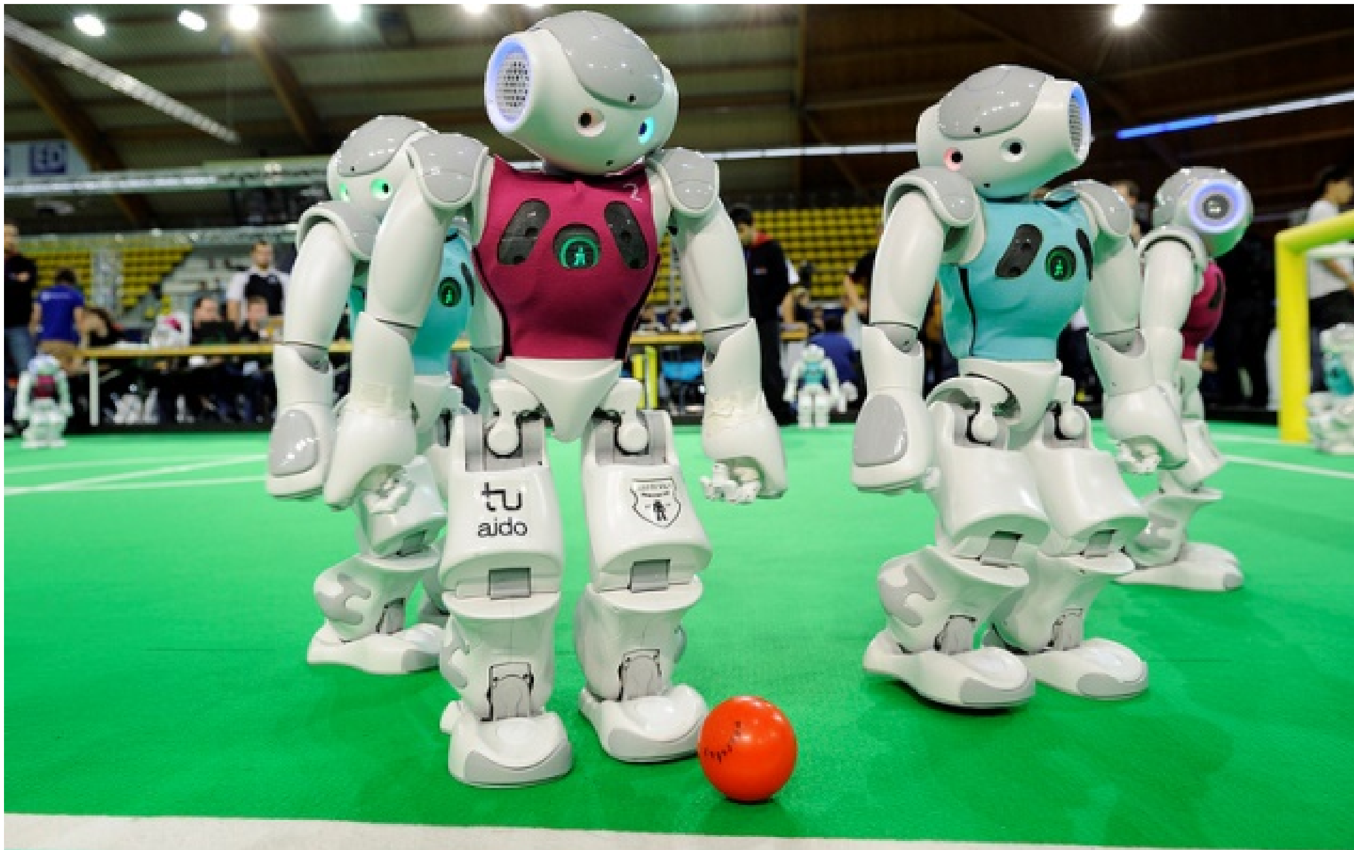
Technieken als hill-climbing en Simulated Annealing werken met "complete" toestanden: alle variabelen hebben een waarde.

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 elders.

Ook mogelijk: Genetische algoritmen, zie later.

RoboCup Ξ



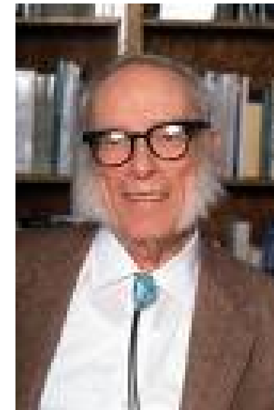
www.robocup.org

Ξ = geen tentamenstof

Een robot is een "actieve, kunstmatige agent wiens omgeving de fysieke wereld" is. Het woord stamt uit 1921 (of eerder), en is gemaakt door de Tsjechische broers Čapek. En softbots: RoboCom, internet programma's.

Van de science fiction schrijver Isaac Asimov (auteur van "I, Robot") zijn de drie (later vier) wetten van de robotica:

1. Een robot mag een mens geen kwaad doen.
2. Een robot moet menselijke orders gehoorzamen (tenzij dat tegen 1. ingaat).



3. Een robot moet zichzelf beschermen (tenzij dat tegen 1. of 2. ingaat).

Een leuke robotsimulatie, van Michael Genesereth en Nils Nilsson, is de volgende.

Bedenk een taak, bijvoorbeeld een toren maken van een paar gekleurde blokken. Stel je nu een "robot" voor die uit *vier mensen* bestaat:

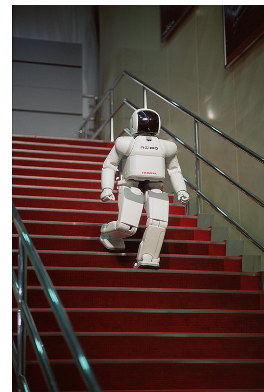
Brein krijgt input van Ogen, maar kan zelf niet zien; geeft opdrachten aan Handen; maakt plannen

Ogen weet niet wat het doel is

Handen, Links en Rechts voeren simpele opdrachten uit; ze zijn geblinddoekt

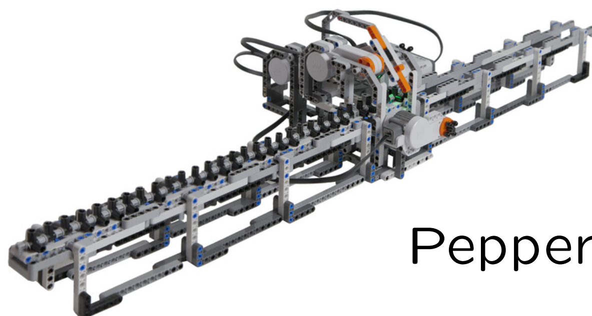
Curiosity

Sony Aibo



Hiroshi Ishiguro's robot (en hijzelf)
"uncanny valley"

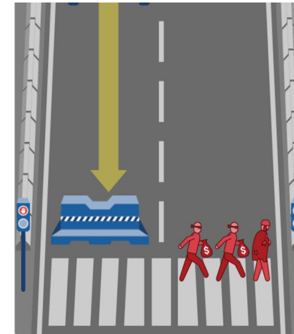
Honda ASIMO

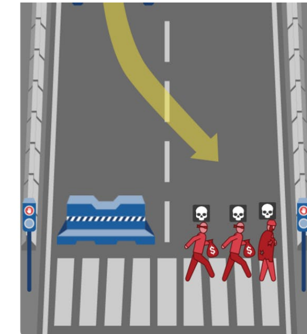


Lego Turing-machine

Pepper-robot van Softbank (2016)

Momenteel doen zelfrijdende systemen het erg goed.





↑computer-zien, kennis, de wet, ethiek(†)↑, . . .

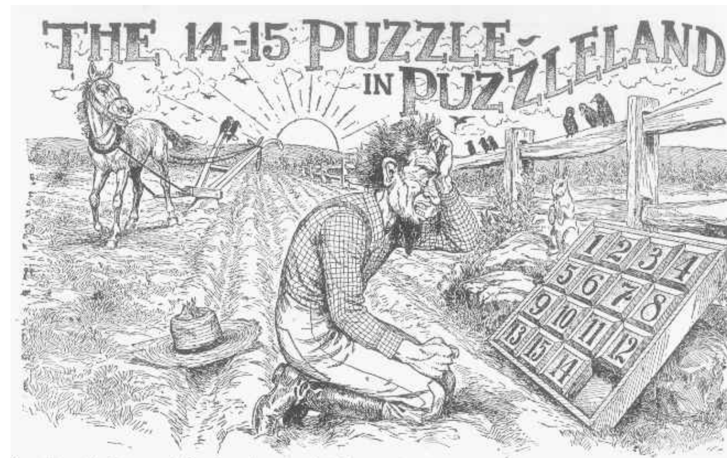
Udacity en Coursera ⟶ MOOC's over zelfrijdende auto's
(†) zie Michael Sandel's video's op YouTube

De subsumption architectuur is een idee van Rodney Brooks van MIT uit 1986, zie zijn artikel (via de AI-website).

Een robot bestaat hier uit verschillende *finite state machines* met klokken. Deze worden netjes gesynchroniseerd en gecombineerd. Zo vermijd je problemen met gigantische configuratieruimtes. Ze worden ook gebruikt om NPC's ("non-player characters") in computerspellen te modelleren.

Het werkt goed voor één kleine taak, maar wat er gebeurt is soms lastig te snappen. Zie je hiervan iets terug bij RoboCom?

De derde programmeeropgave gaat over puzzels en A*:



www.liacs.leidenuniv.nl/~kosterswa/AI/aster2024.html

Het huiswerk voor de volgende keer (10 april 2024): lees Hoofdstuk 19.1/4, p. 651–672 en p. 677–679 van [RN] door (in de derde druk p. 693–758 en p. 768–776) over het onderwerp Leren.

Maak (ook tijdens het werkcollege van donderdag 4 april 2024) in het bijzonder de sommen 11, 16 en 17 van

www.liacs.leidenuniv.nl/~kosterswa/AI/opgaven1.pdf

Er zijn video's met uitwerkingen.

Voor Robotica (Ξ = geen tentamenstof) lees Hoofdstuk 26 van [RN] (in de derde druk Hoofdstuk 25).

Denk tevens aan de derde opgave: A*; deadline woensdag 17 april 2024.