

## Kunstmatige Intelligentie (AI)

Hoofdstuk 4.1 van Russell/Norvig = [RN]  
Locaal zoeken en Optimalisatie

voorjaar 2021

College 12, 3 mei 2021

[www.liacs.leidenuniv.nl/~kosterswa/AI/genetisch.pdf](http://www.liacs.leidenuniv.nl/~kosterswa/AI/genetisch.pdf)

Als het pad naar de oplossing er niet toe doet (zoals bij het dames-probleem), kunnen we **local search** technieken gebruiken. Je verbetert dan in elke stap één of meer huidige oplossingen. Bijkomend voordeel: je hebt altijd een complete (redelijke) oplossing.

Enkele problemen hiermee:

- lokale maxima
- plateau's
- “heuvelruggen” (= ridges)

We zullen in het bijzonder **Genetische algoritmen** bekijken.

We combineren de hill-climber (= gradient descent) met random bewegingen als volgt tot **Simulated Annealing**:

```
for tijd ← 1 to ∞ do
  Temperatuur ← schedule[tijd]
  if Temperatuur = 0 then return huidig
  volgende ← random opvolger van huidig
   $\Delta E$  ← waarde[volgende] – waarde[huidig]
  if  $\Delta E > 0$  then huidig ← volgende
  else huidig ← volgende met kans  $e^{\Delta E / \textit{Temperatuur}}$ 
```

Het algoritme is geïnspireerd op natuurkundige ideeën en het **Metropolis-algoritme** uit 1953.

Er zijn allerlei soorten **evolutionaire algoritmen** (deel van het vakgebied **natural computing**, waar ook bijvoorbeeld **DNA-computing** onder valt, en misschien zelfs **quantum computing**) die ideeën uit de evolutietheorie gebruiken, zoals **evolution strategies**. Zie ook de speciale colleges Natural Computing en Evolutionary Algorithms.

Wij doen hier kort de wat meer traditionele **Genetische algoritmen (GAs)**.



Genetische algoritmen (GAs) zijn een vorm van **reinforcement** leren, waarbij gewerkt wordt met beloningen en straffen in plaats van met de “goede antwoorden”.

Een GA heeft een **populatie** met kandidaat-oplossingen (**individuen** of **chromosomen**) voor het betreffende probleem. Deze worden met elkaar (en zichzelf) gecombineerd tot hopelijk betere oplossingen. Essentieel is een **fitness-functie** die de kwaliteit van oplossingen beoordeelt.

Het wordt lastiger als het om **multi-objective optimization** gaat, met meerdere doelen. Je krijgt dan **Pareto-fronten**.

Het algemene schema van een GA ziet er zo uit:

```
Initialiseer populatie
Evalueer populatie
while not klaar do
    Selecteer ouders
    Genereer met crossover kinderen (recombineer)
    Muteer kinderen
    Evalueer kinderen
    Bepaal nieuwe populatie
return beste element uit populatie
```

Maar wanneer ben je klaar?

Er zijn vele varianten. Twee hoofdstromen zijn:

**generationeel** de kinderen vervangen de ouders: er komt steeds een nieuwe generatie (populatie)

**steady state** de nieuwe populatie bestaat uit de besten van ouders en kinderen

In het generationele geval behoudt een **elitair** algoritme een stel beste van de ouders. En die kun je ook weer laten verouderen: hun fitness daalt met hun leeftijd (**ag(e)ing**).

Maar de belangrijkste variatie zit in de **genetische operatoren**:

**selectie** hoe kies je kandidaten voor reproductie?

**crossover** hoe combineer je ouders?

**mutatie** hoe wijzig je individuen op een zinvolle manier?

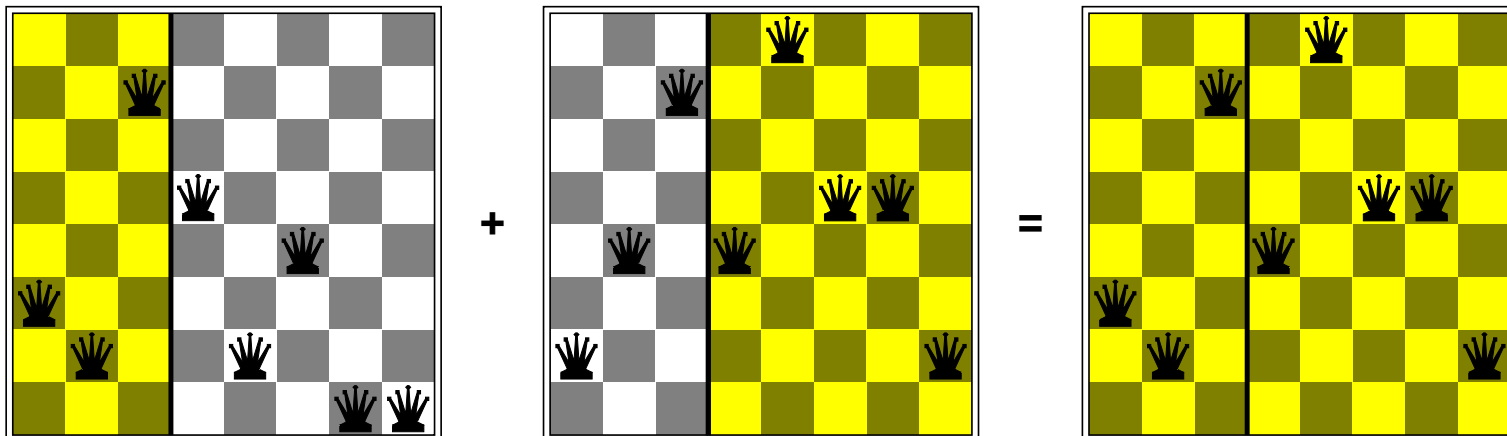


Vaak wordt selectie met een **roulette-wiel** gedaan. De kans dat een individu gebruikt wordt (bijvoorbeeld voor reproductie) is evenredig met zijn/haar fitness.

Heb je  $n$  individuen met gemiddelde fitness  $\bar{f}$ , dan wordt een individu met fitness  $f$  getrokken met kans  $f/(n * \bar{f})$ . Je draait aan een roulette-wiel waarvan de grootte van de taartpunten proportioneel is met de fitness.

Een alternatief is **rank-based** selectie, waar alleen de volgorde van de fitness-waarden bepalend is.

Bij crossover wil je twee (of meer) ouders combineren. Voor de hand ligt de eerste helft van de ene ouder en de tweede helft van de andere ouder te nemen:



De volgende soorten crossover worden vaak gebruikt:

**single-point crossover** (als hiervoor)

$$\left. \begin{array}{l} \underline{11101001000} \\ 00001\underline{010101} \end{array} \right\} \Rightarrow \begin{array}{l} 11101010101 \\ 00001001000 \end{array}$$

**two-point crossover**

$$\left. \begin{array}{l} \underline{11101001000} \\ 0000\underline{1010101} \end{array} \right\} \Rightarrow \begin{array}{l} 11001011000 \\ 00101000101 \end{array}$$

**uniform crossover**

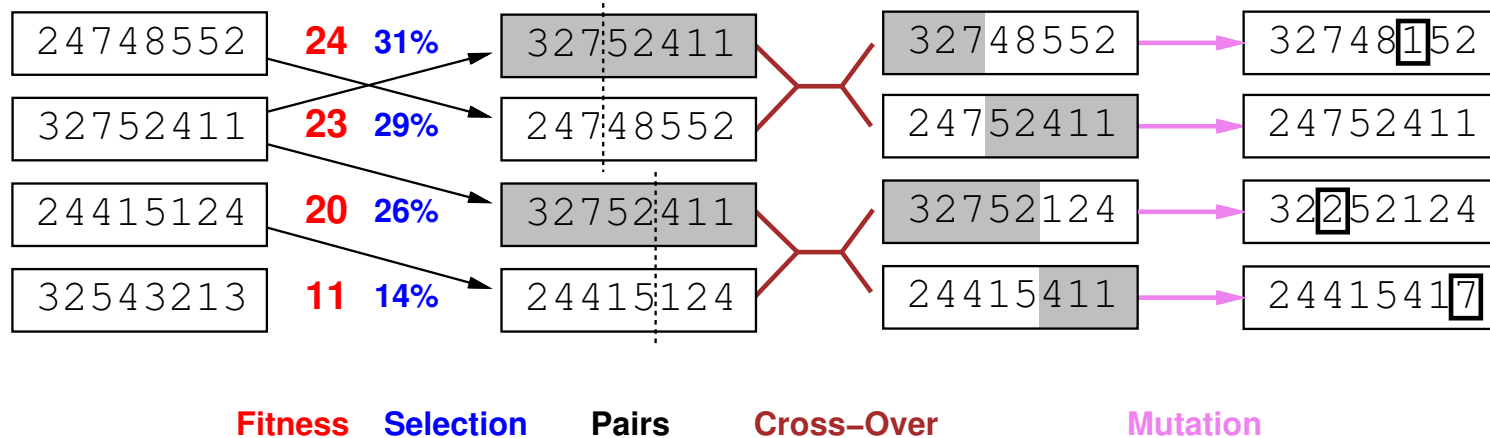
$$\left. \begin{array}{l} \underline{11101001000} \\ 0\underline{0001010101} \end{array} \right\} \Rightarrow \begin{array}{l} 10001000100 \\ 01101011001 \end{array}$$

Bij mutatie wordt elke bit van een individu met een zekere (kleine) kans, de **mutation rate**, omgeklapt. Een andere optie is om van elk individu een random bit om te klappen.

Meestal wordt mutatie (en ook crossover) speciaal gebouwd voor een probleem. Je kunt bijvoorbeeld ook random bits omwisselen, en dan het aantal enen constant houden — als dat zinvol lijkt.

En er zijn ook nog **reparatie-operatoren**, die een willekeurige bitstring ombouwen naar een potentiële oplossing, bijvoorbeeld met een “hill-climber”.

Een voorbeeld-stap in een GA is:



Hier hebben we overigens geen bits, maar getallen in de individuen. Dit kun je zien als **genen**. De string heet soms wel het **genotype**, dat wat hij voorstelt het **phenotype**.

Resteren nog twee problemen: de **representatie** (hoe stop je kandidaat-oplossingen in een string?) en de **fitness-** of **evaluatie-functie** (hoe waarderen we die individuen?).

Soms is het zo dat iedere string kan voorkomen, soms niet. Soms zijn reparatie-operatoren nodig, soms niet.

Meestal is de fitness-functie de som van een aantal componenten. Harde constraints geven grote (of juist lage) bijdragen. Er kunnen zoals gezegd zelfs meerdere fitness-functies gebruikt worden: “multi-objective optimization”.

Een beroemd probleem is het **Traveling Salesperson/man Problem (TSP)**: in een gegeven complete graaf  $G$ , met gewichten = afstanden op de takken, moeten we een gesloten pad met minimale lengte vinden dat alle knopen aandoet.

Je kunt bijvoorbeeld de volgende pad-representatie gebruiken: (5 1 7 8 9 4 6 2 3) betekent dat je van 5 naar 1 naar 7 naar ... naar 3 naar 5 gaat.

En de fitness-functie is de totale lengte van het pad.

Er zijn vele mogelijkheden voor crossover bij TSP, zoals:

### partially mapped crossover (PMX)

$$\left. \begin{array}{l} (1\ 2\ 3\ | 4\ 5\ 6\ 7\ | 8\ 9) \\ (4\ 5\ 2\ | 1\ 8\ 7\ 6\ | 9\ 3) \end{array} \right\} \Rightarrow \begin{array}{l} (4\ 2\ 3\ | 1\ 8\ 7\ 6\ | 5\ 9) \\ (1\ 8\ 2\ | 4\ 5\ 6\ 7\ | 9\ 3) \end{array}$$

### order crossover (OX)

$$\left. \begin{array}{l} (1\ 2\ 3\ | 4\ 5\ 6\ 7\ | 8\ 9) \\ (4\ 5\ 2\ | 1\ 8\ 7\ 6\ | 9\ 3) \end{array} \right\} \Rightarrow \begin{array}{l} (2\ 1\ 8\ | 4\ 5\ 6\ 7\ | 9\ 3) \\ (3\ 4\ 5\ | 1\ 8\ 7\ 6\ | 9\ 2) \end{array}$$

### cycle crossover (CX)

$$\left. \begin{array}{l} (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) \\ (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5) \end{array} \right\} \Rightarrow \begin{array}{l} (1\ 2\ 3\ 4\ 7\ 6\ 9\ 8\ 5) \\ (4\ 1\ 2\ 8\ 5\ 6\ 7\ 3\ 9) \end{array}$$

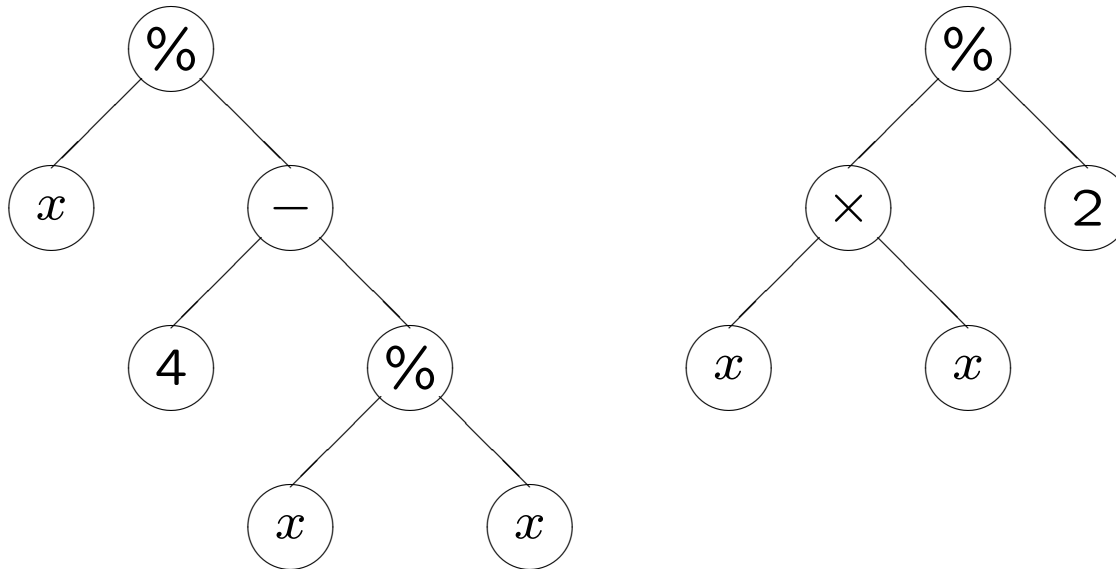


In **Genetic Programming** (GP; John Koza, 1992) manipuleer je bomen (oftewel programma's) in plaats van strings.

Een voorbeeld. Stel je wilt de functie  $f(x) = x^2/2$ , gegeven door 10 paren (0.000,0.000), (0.100,0.005), ..., (0.900,0.405), maken via bomen met knopen  $x$ ,  $+$ ,  $-$ ,  $\times$ ,  $\%$  ("protected division"),  $-5$ ,  $-4$ , ...,  $4$ ,  $5$ .

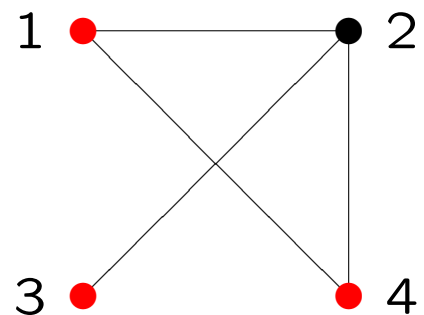
We kunnen een populatiegrootte van 600 nemen, crossoverkans 90%, mutatiekans 5%, en toernooi-selectie (kies er steeds random 4, en neem daarvan de beste; die 4 geeft "selection pressure"). Crossover: wissel twee willekeurige sub-bomen, mutatie: vervang willekeurige sub-boom door een random-boom.

Tijdens generatie 0 heb je bijvoorbeeld de linkerboom, die de functie  $f_0(x) = x/3$  voorstelt. Tijdens generatie 3 zou je de boom rechts kunnen krijgen, die  $f_3(x) = x^2/2$  voorstelt.



Stel je moet een Genetisch algoritme maken dat zo goed mogelijk een ongerichte graaf zodanig inkleurt dat zo weinig mogelijk aangrenzende knopen dezelfde kleur hebben, en dat nummers van burenen met dezelfde kleur zo veel mogelijk verschillen. Hoe kies je representatie en operatoren? Invoer (het probleem, links) en mogelijke uitvoer:

```
4 2
0 1 0 1
1 0 1 1
0 1 0 0
1 1 0 0
```



Dat betekent 4 knopen, 2 kleuren.

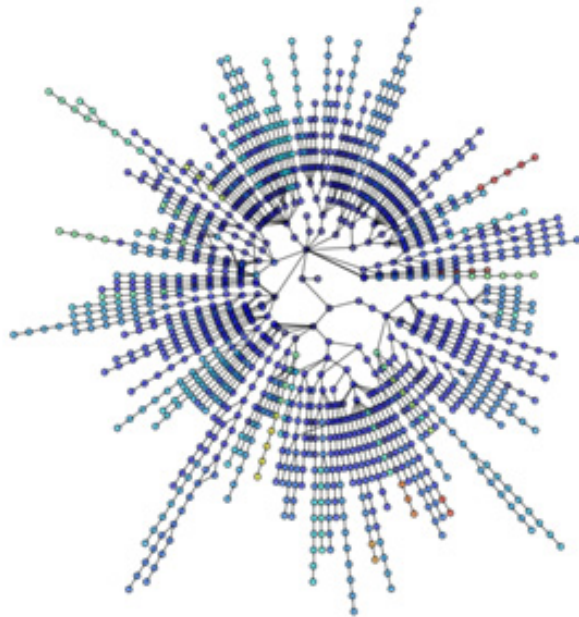
We moeten nu een Genetisch algoritme maken dat probeert een  $n \times n$  **magisch vierkant** te fabriceren. Dit bestaat uit een vierkante opstelling van de  $n^2$  positieve gehele getallen  $1, 2, 3, \dots, n^2$ , waarbij alle rijen en kolommen (en wellicht ook nog andere combinaties) sommeren tot hetzelfde getal.

8	1	6	16	3	2	13
3	5	7	5	10	11	8
4	9	2	9	6	7	12
			4	15	14	1



Hoe doe je dat?

En nog een voorbeeld: maak een Genetisch algoritme dat probeert een graaf zo goed mogelijk te tekenen: zo weinig mogelijk snijdende takken, en afstanden lijkend op de “echte”.



Hoe maak je een Genetisch algoritme dat probeert een  $4 \times 4$  en een  $9 \times 9$  **Sudoku** op te lossen?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

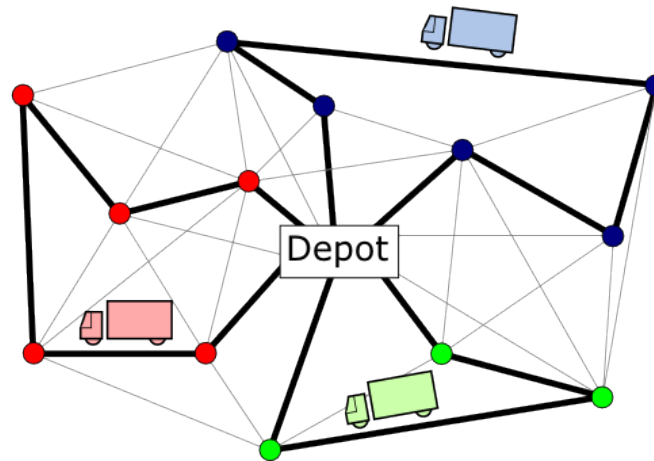
rijen, kolommen  
en  $3 \times 3$  “blokken”  
bevatten  $1, 2, \dots, 9$

Hoe maak je een Genetisch algoritme dat probeert een  $4 \times 4$  en een  $9 \times 9$  **Sudoku** op te lossen?

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

bron: Wikipedia

Ontwerp een **Genetisch algoritme** dat zo goed mogelijk probeert een routerings-probleem op te lossen.



Mutatie? Crossover?

Multi-objective optimization? Pareto-front?



Het huiswerk voor de volgende keer (maandag 10 mei 2021): lees **Hoofdstuk 13 en 14**, p. 385–402 en p. 412–436 van [RN] door (in de derde druk p. 480–499 en p. 510–529) over het onderwerp Onzekerheid en Bayesiaanse netwerken.

Denk na over de vierde opgave: [Neurale netwerken](#); deadline: **dinsdag 18 mei 2021**.