Data Science 2
Meta-heuristieken (deel 1)

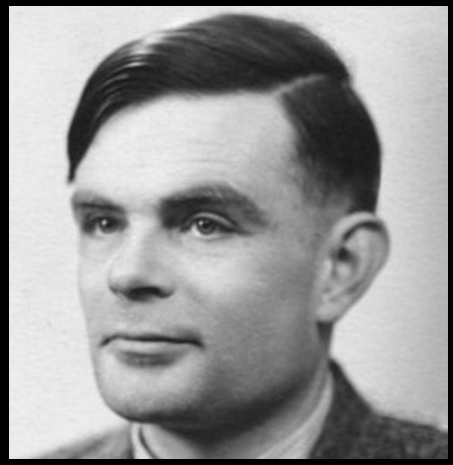
Wim De Keyser
Geert De Paepe
Jan Van Overveld

KdG Karel de Grote Hogeschool

Quote van de week

"If a machine is expected to be infallible, it cannot also be intelligent"

Alan Turing (1912-1954)



Alan Turing

Alan Turing was een **computerpionier**. Hij is het bekendst om de **turingtest**, maar heeft daarnaast belangrijk werk verricht op het vlak berekenbaarheid met o.a. de **turingmachine**, een mechanisch model van berekening en berekenbaarheid.

Ten tijde van de Tweede Wereldoorlog werkte Turing in het geheim bij de Government Code and Cipher School. Turing maakte deel uit van een team dat de codes kon ontcijferen die door het Enigma-apparaat, een Duits coderingssysteem, waren gegenereerd.

In 1952 werd Turing gearresteerd wegens homoseksuele handelingen (die tot 1967 in Engeland voor mannen strafbaar waren) en veroordeeld.

Op 24 december 2013 verleende koningin Elizabeth II Alan Turing gratie en werd zijn veroordeling wegens homoseksualiteit uit de boeken geschrapt.

De Bank of England maakte op 15 juli 2019 bekend dat het portret van Alan Turing op het 50 pond biljet zal verschijnen vanaf eind 2021.

Agenda

1. Inleiding – Optimalisatieproblemen
2. Algoritme versus heuristiek
   - ↳ Computationele complexiteit
3. Soorten heuristieken
4. Simulated annealing

Cursus/ Boek

Tekst op Canvas: "metaheuristieken.pdf"

canvas

Oefeningen

Inleiding - Optimalisatieproblemen

Inleiding - Optimalisatieproblemen

Problemen –in bedrijfscontext-

- steken dagelijks de kop op
- toenemende complexiteit

Probleem oplossen

= Beslissing nemen

= Keuze maken uit verschillende alternatieven

<u>Doel</u>: **beste resultaat**, rekeninghoudend met **bestaande beperkingen** (geld, tijd, beschikbare mensen, aanwezige kennis, grondstoffen, wetgeving,…)

Inleiding - Optimalisatieproblemen

Meeste problemen –in bedrijfscontext·

- hebben zeer veel mogelijke oplossingen
- zijn te complex

waardoor een menselijke besluitnemer
niet zomaar de 'beste' oplossing kan 'zien'

Ondersteuning voor besluitnemers vanuit Operationeel Onderzoek:

⇨ Wiskundig model van de realiteit opstellen,

⇨ 'Beste' oplossing voor het model bepalen,

⇨ Oplossing terug vertalen naar de realiteit

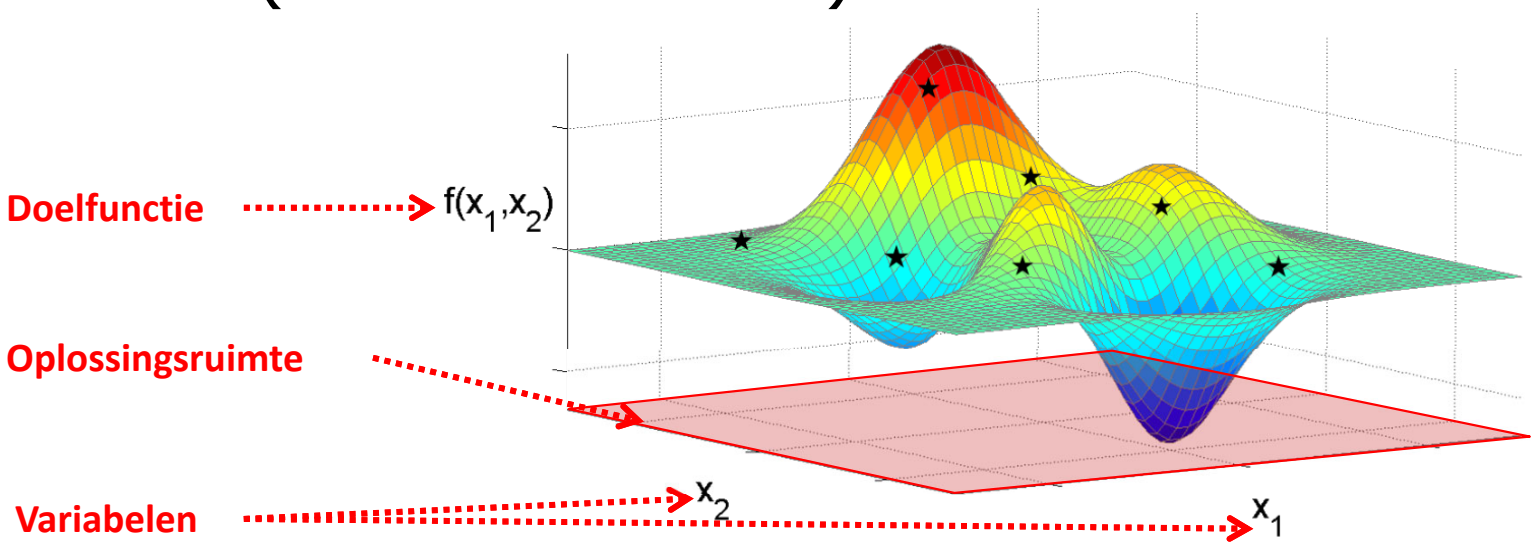
Inleiding - Optimalisatieproblemen

Optimalisatieprobleem omvat:

- **Variabelen**
- Omschrijving van de verzameling van alle mogelijke **oplossingen**(*)
- Lijst van beperkingen: **constraints**

} **oplossings-ruimte**

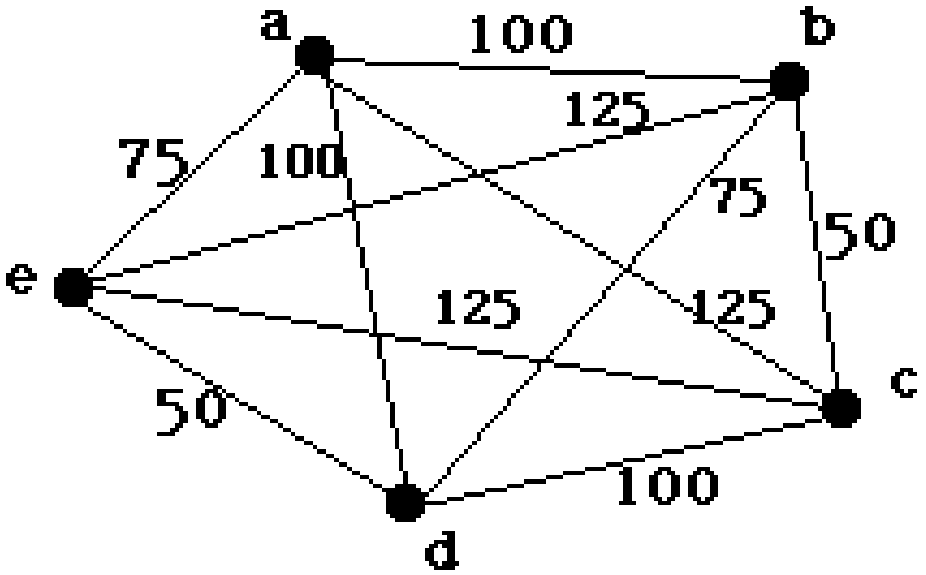
- Een te maximaliseren of te minimaliseren **doelfunctie** (of **kostfunctie**)



(*) een oplossing = toekennen van concrete waarden aan de variabelen

Voorbeeld: Handelsreizigersprobleem (Traveling Salesman Problem of TSP)

Als er *n* steden gegeven zijn die een handelsreiziger moet bezoeken, samen met de afstand tussen ieder paar van deze steden, zoek dan de kortste weg die kan worden gebruikt, waarbij iedere stad eenmaal wordt bezocht.



	a	b	c	d	e
a	0	100	125	100	75
b	100	0	50	75	100
c	125	50	0	100	125
d	100	75	100	0	50
e	75	100	125	50	0

Hoe ziet een oplossing eruit? Opeenvolging van de steden : (a,e,d,b,c)

Waarde doelfunctie? 75 + 50 + 75 + 50 + 125 = 375

Aantal mogelijke oplossingen? 5 . 4 . 3 . 2 . 1 = 120
(alle mogelijke permutaties, algemeen: n!)

Inleiding - Optimalisatieproblemen

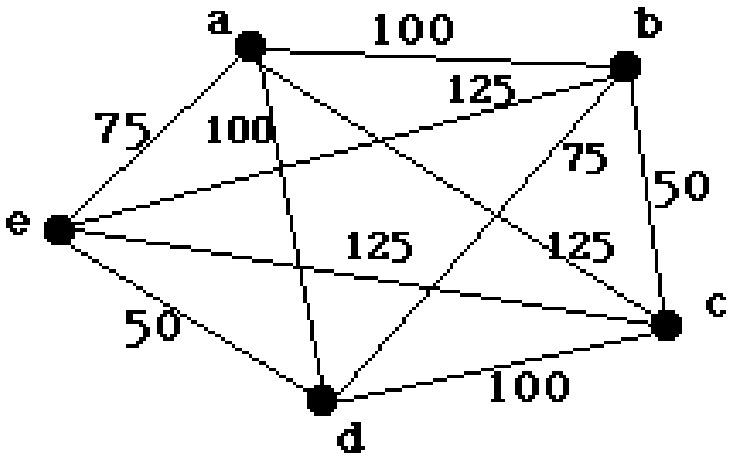
Hoe de ‘beste’ oplossing bepalen?

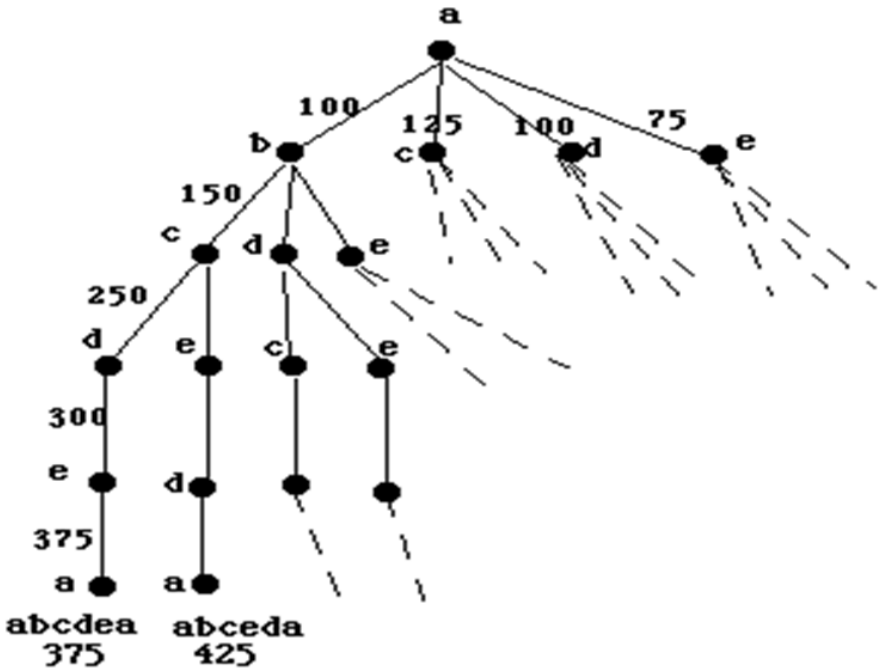
Voor de hand liggende benadering:

- alle oplossingen bepalen
- voor elke oplossing de doelfunctie bepalen
- de oplossing nemen met de laagste doelfunctie

⇨ **Enumeration method**

Oplossingsruimte kan onhandel-baar groot zijn



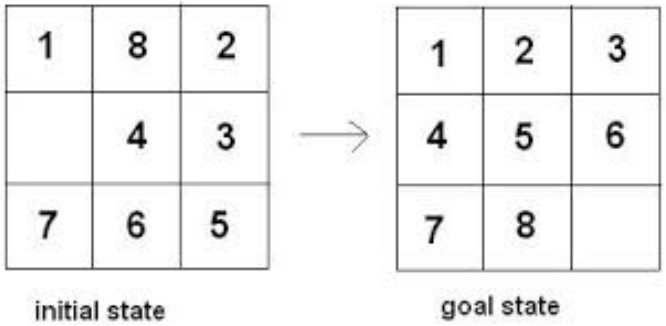


oplossingen kunnen worden voorgesteld als paden in een zoekboom

De 8 – puzzel

Vakjes in juiste volgorde zetten

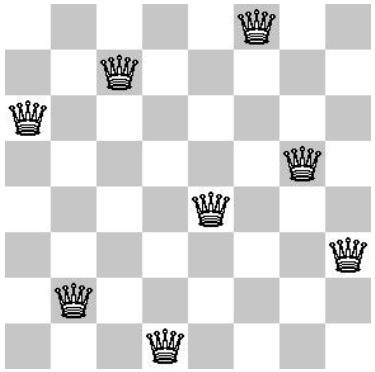
Hoe ziet *een* oplossing eruit?



N – Queens

Acht koninginnen op schaakbord plaatsen zonder dat ze elkaar kunnen aanvallen

Hoe ziet *een* oplossing eruit?

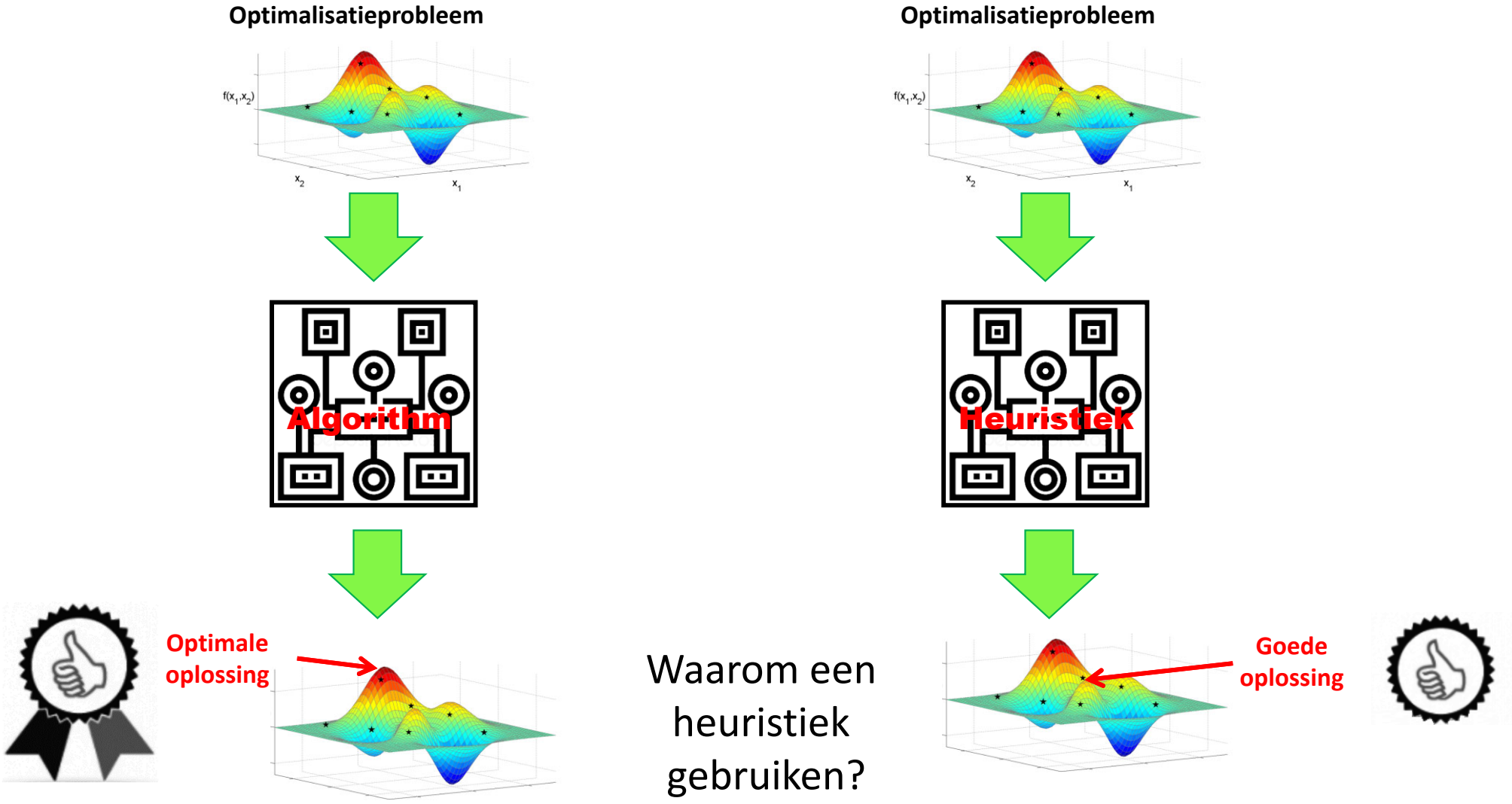


2

Algoritme versus heuristiek

Wat is een algoritme en wat is een heuristiek?

Doel optimalisatieprobleem: 'Beste' oplossing vinden tussen alle mogelijke oplossingen.

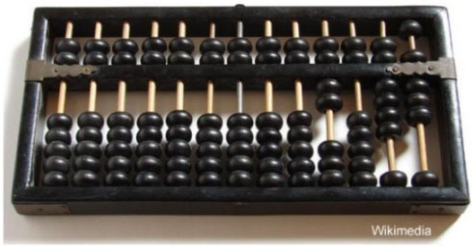


Computationele complexiteit

Doel: computationele problemen te klasseren in moeilijkheidscategorieën

Basisvraag: Welke **middelen** van een machine /computer worden er ingezet?

- (reken)tijd
- geheugen(plaats)



Hoe: in functie van de lengte van de input

Computationele complexiteit algoritme: inzet middelen worst-case startsituatie

Computationele complexiteit probleem: inzet middelen beste algoritme

Voorbeelden van computationele tijdscomplexiteit:

Tijd	Notatie	Voorbeeld
Constante tijd	O(1)	Het opvragen van een element uit een array
Lineaire tijd	O(n)	Het minimum in een ongeordende reeks getallen zoeken
Logaritmische tijd	O(log n)	Binary search
Lineair logaritmische tijd	O(n log n)	Merge sort
Kwadratische tijd	$O(n^2)$	Selection sort, Bubble sort, Quick sort
Exponentiële tijd	$O(2^n)$	Recursief berekenen van het n-de fibonacci getal

polynomiale tijd:

$$O(n^k) = a_0 + a_1 n + a_2 n^2 + \ldots + a_k n^k$$

Waarom een heuristiek nemen als er een algoritme bestaat?

Algoritme versus heuristiek

1.

geen polynomiale tijdscomplexiteit

wel een polynomiale tijdscomplexiteit

Vb: handelsreizigersprobleem

(gaten of uitsparingen in een plaat boren, ontwerpen printplaten,…)

dynamic programming algoritme: $O(n^2\ 2^n)$

nearest neighbour heuritiek: $O(n^2)$

n	n!	$n^2\ 2^n$	n^2
5	120	800	25
10	3.628.800	102.400	100
15	1.307.674.368.000	7.372.800	225
20	2.432.902.008.176.640.000	419.430.400	400

2.

Geen algoritme beschikbaar

Altijd een eenvoudige heuritiek te bedenken

Soorten heuristieken

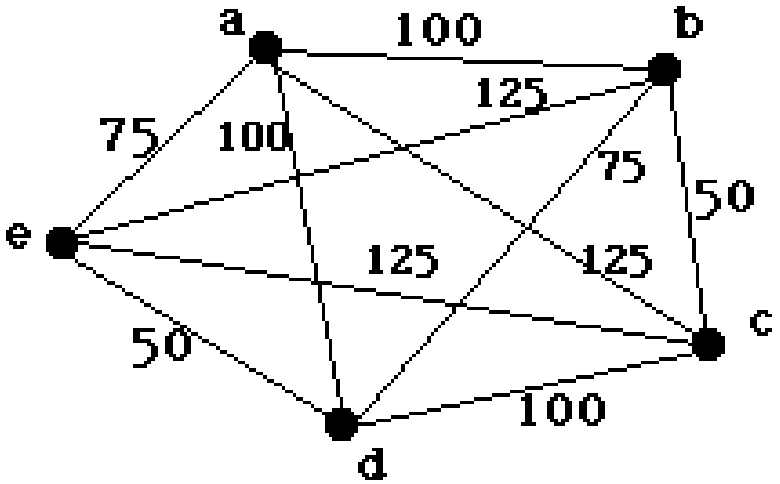
'Custum made'-heuristieken

- ontwikkeld voor een specifiek optimalisatie-probleem
- niet herbruikbaar voor andere optimalisatie-problemen.
- gebruikt/exploiteert specifieke aspecten en eigenschappen niet noodzakelijk aanwezig in andere optimalisatieproblemen.

Eenvoudige heuristieken

D.m.v. een eenvoudig toe te passen criterium snel een goede oplossing bepalen

vb *nearest neighbour* benadering van het handelsreizigersprobleem:



Startpunt : a
a - dichtste buur: e
e - dichtste buur (nog niet bezocht): d
d - dichtste buur (nog niet bezocht): b
b - dichtste buur (nog niet bezocht): c
c – terug naar startpunt a

Oplossing : (a, e, d, b, c) met afstand 375

- Oplossing kan 'ver' van optimale oplossing liggen.
- 'Quick & dirty'-oplossing –beter dan random-oplossing-
- Worden vaak als startoplossing voor andere heuristieken (zie verder) gebruikt

Eenvoudige heuristieken

- Twee types eenvoudige heuristieken
  - ***'constructie'-heuristiek***:
    stapsgewijs een oplossing opgebouwd

  - ***'twee-fasen'-heuristiek***:

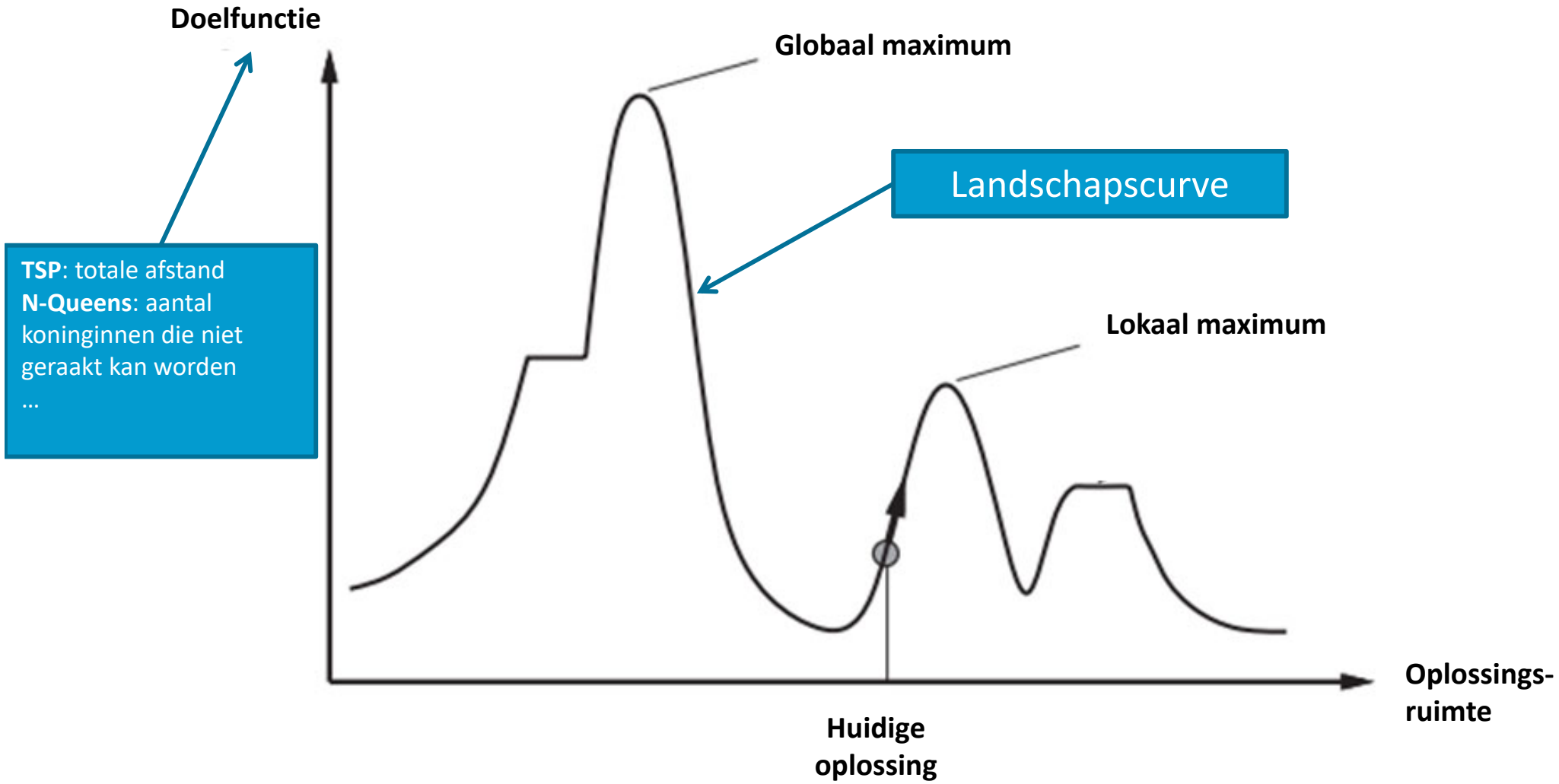
- Strategie 1 - 1° fase: constructie initiële oplossing
  2° fase: verbeteren initiële oplossing
- Strategie 2 – 1° fase: het probleem in deelproblemen splitsen en initiële deeloplossing voor elk deelprobleem bouwen
  2° fase: deeloplossingen integreren in één oplossing

'Lokale zoek'-heuristieken

Zoekmethoden die steeds in de 'buurt' zoeken van de vorige oplossing naar een betere oplossing. Een of meerdere stopcriteria worden gehanteerd.

- Slechts één of enkele oplossingen bijhouden en die "verbeteren"
- Oplossingen in de buurt zijn oplossingen met 'kleine' aanpassingen
- Resultaat is de 'beste' oplossing die tijdens de zoektocht werd gevonden (niet noodzakelijk de laatste oplossing)
- Risico 'vast' te raken in een lokaal minimum

Soorten heuristieken – Vizualiseren '*lokaal zoek*'-heuristiek



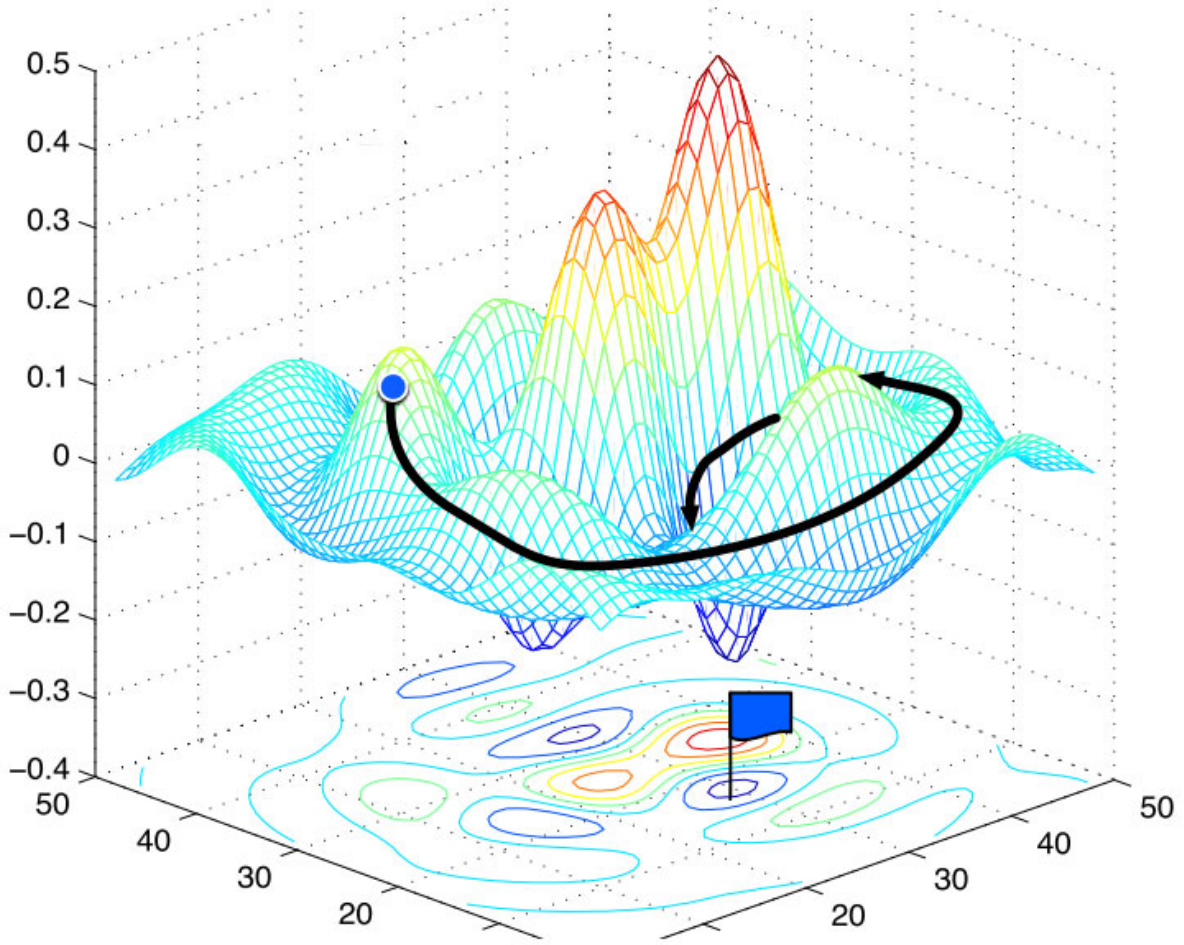
- Hoe voorkomen dat we vast komen te zitten?
  - Meerdere keren na elkaar uitgevoerd met andere initiële oplossingen
  - Zie meta-heuristieken

Soorten heuristieken – Visualiseren '*lokaal zoek*'-heuristiek

Oplossingruimte als n-dimensionaal "landschap"

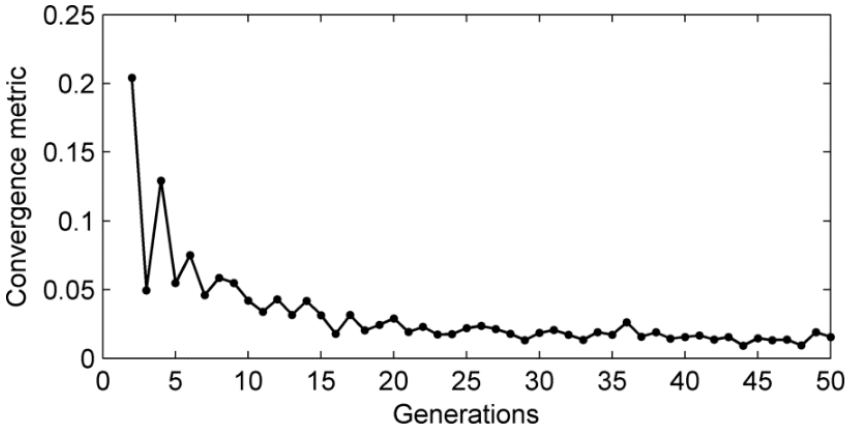
– Oplossingen zijn punten

– Hoogte van landschap is kwaliteit van de oplossing



Per stap evolueren de oplossingen richting een (lokaal) optimum

Meta-heuristieken



High level procedure toepasbaar op gelijk welk optimalisatie-probleem.

Het bevat de volgende elementen:

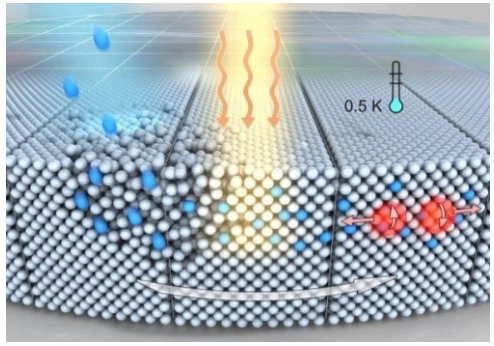
- Initiële oplossing bepalen
  (at random of d.m.v. een eenvoudige heuristiek)
- Toepassen ’lokaal zoek’-principe: huidige oplossing vervangen door ‘betere’ oplossing in de buurt
- Toelaten om af en toe toch naar een ‘slechtere’ buur te gaan (een manier om aan een lokaal optimum te ontsnappen)
- Gebaseerd op een analogie uit de fysica, de biologie of de ethologie
- Parameters sturen de duur van de heuristiek en de kwaliteit van de gevonden oplossing

Soorten heuristieken

Voorbeelden van meta-heuristieken

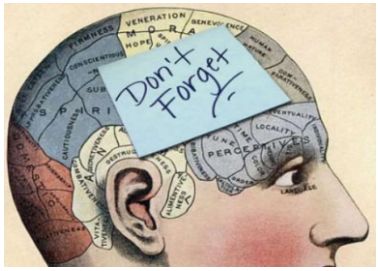
Simulated Annealing(Kirkpatrick, e.a. ,1970)

- gebaseerd op het afkoelen van een metaal
- potentiële oplossingen kristalliseren uit



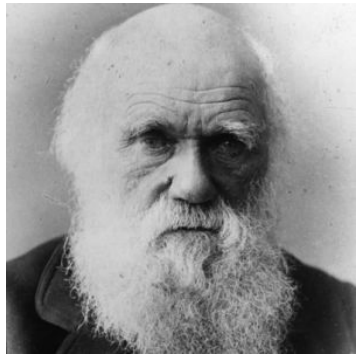
Tabu search (Glover, 1986)

- gebaseerd op de werking van het geheugen
- 'rondwandelen' in oplossingsruimte



Genetische algoritmen (Shannon, e.a., 1975)

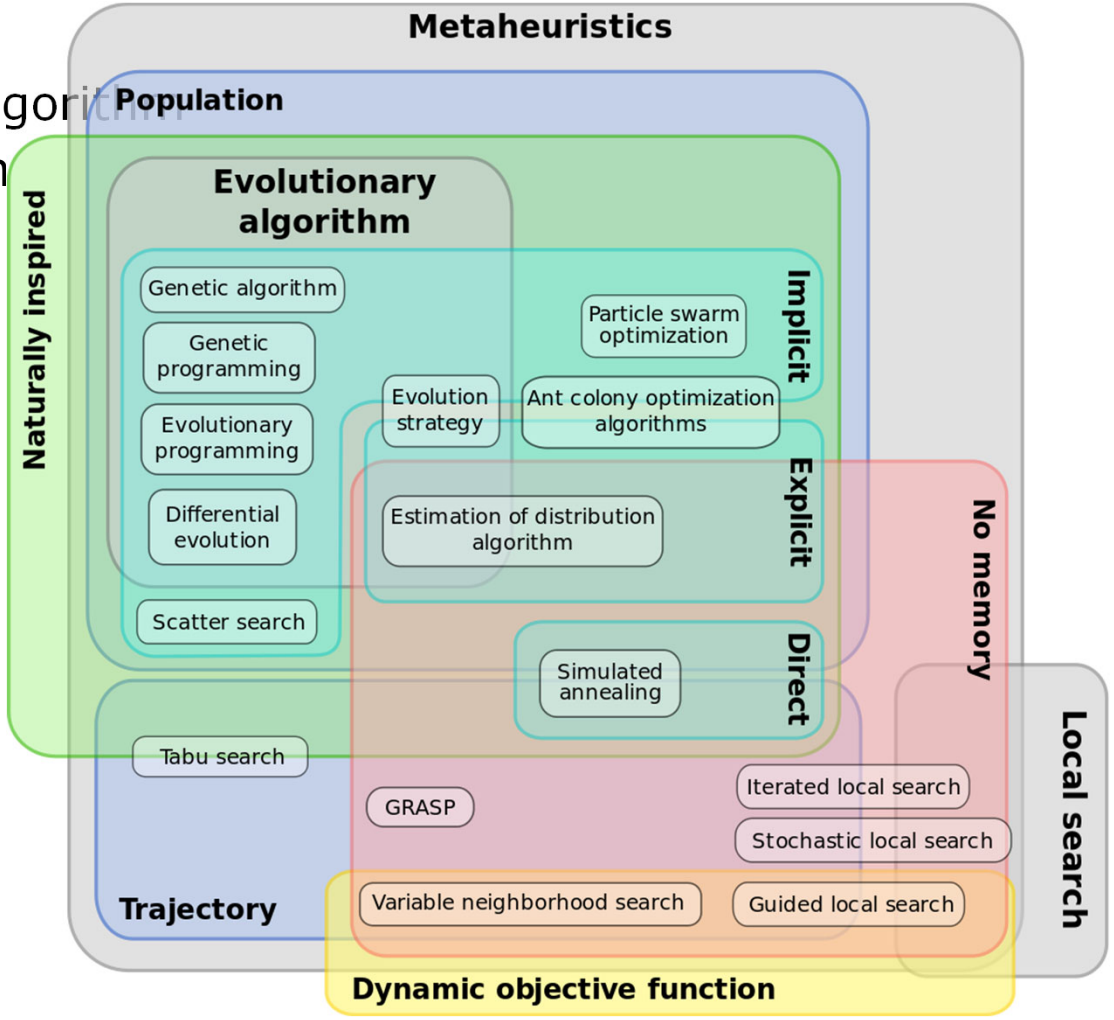
- gebaseerd op evolutieleer van Charles Darwin (°1809 - †1892)
- oplossingen verbeteren door natuurlijke selectie en reproductie



Soorten heuristieken

Andere voorbeelden van meta-heuristieken:

- ant colony optimisation algorithm
- bat algorithm
- particle swarm optimization
- artificial immune systems
- bacterial foraging optimization algorithm
- biogeography-based optimization
- coevolutionary algorithms
- cultural algorithms
- differential evolution algorithm
- greedy randomized adaptive search procedure
- scatter search
- ...



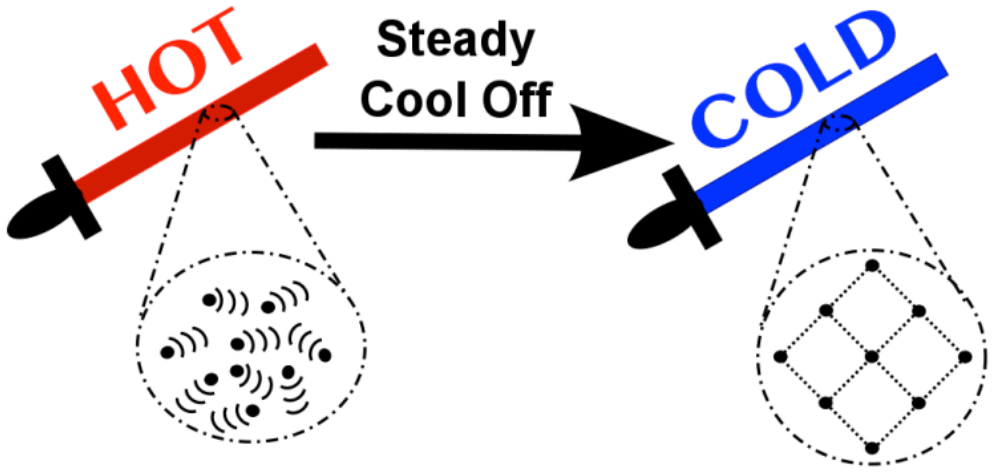
https://en.wikipedia.org/wiki/Metaheuristic#/media/File:Metaheuristics_classification.svg

4

Simulated Annealing

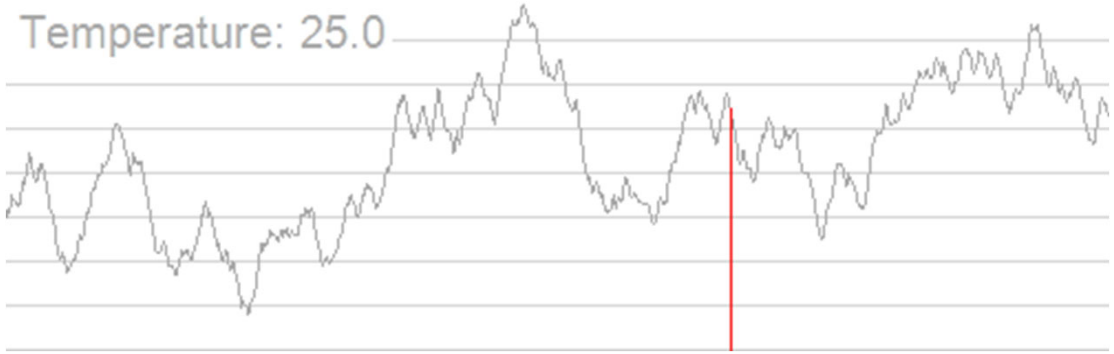
Gebaseerd op afkoelingsproces in materialen: annealing

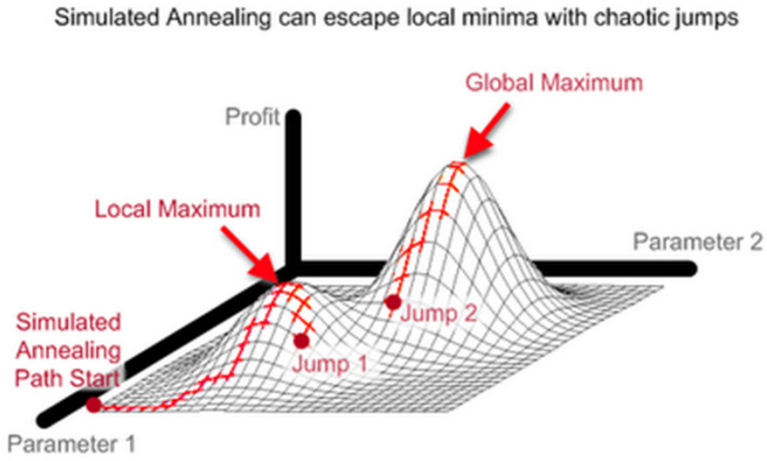
- atomen in materiaal bewegen, maar hoe lager de temperatuur, hoe minder
- Materiaal opwarmen om in juiste vorm te krijgen, nadien terug afkoelen
- afkoeling te snel ⇨ onzuiverheden (amorfe structuren)
  afkoeling geleidelijk ⇨ sterkere kristalstructuren
- Eindtoestand wordt dus geleidelijk aan bereikt.



Simulated annealing

- Willekeurige zoektocht naar oplossing d.m.v. "**sprongen**" (random walk)
- In principe enkel "sprong" naar een betere oplossing
  → soms ook minder goede (*Waarom?).*
- Na elke "sprong" **beste oplossing** onthouden.
- Kans op "sprongen" naar minder goede oplossingen worden kleiner met **dalende temperatuur**
  → oplossing kristalliseert uit

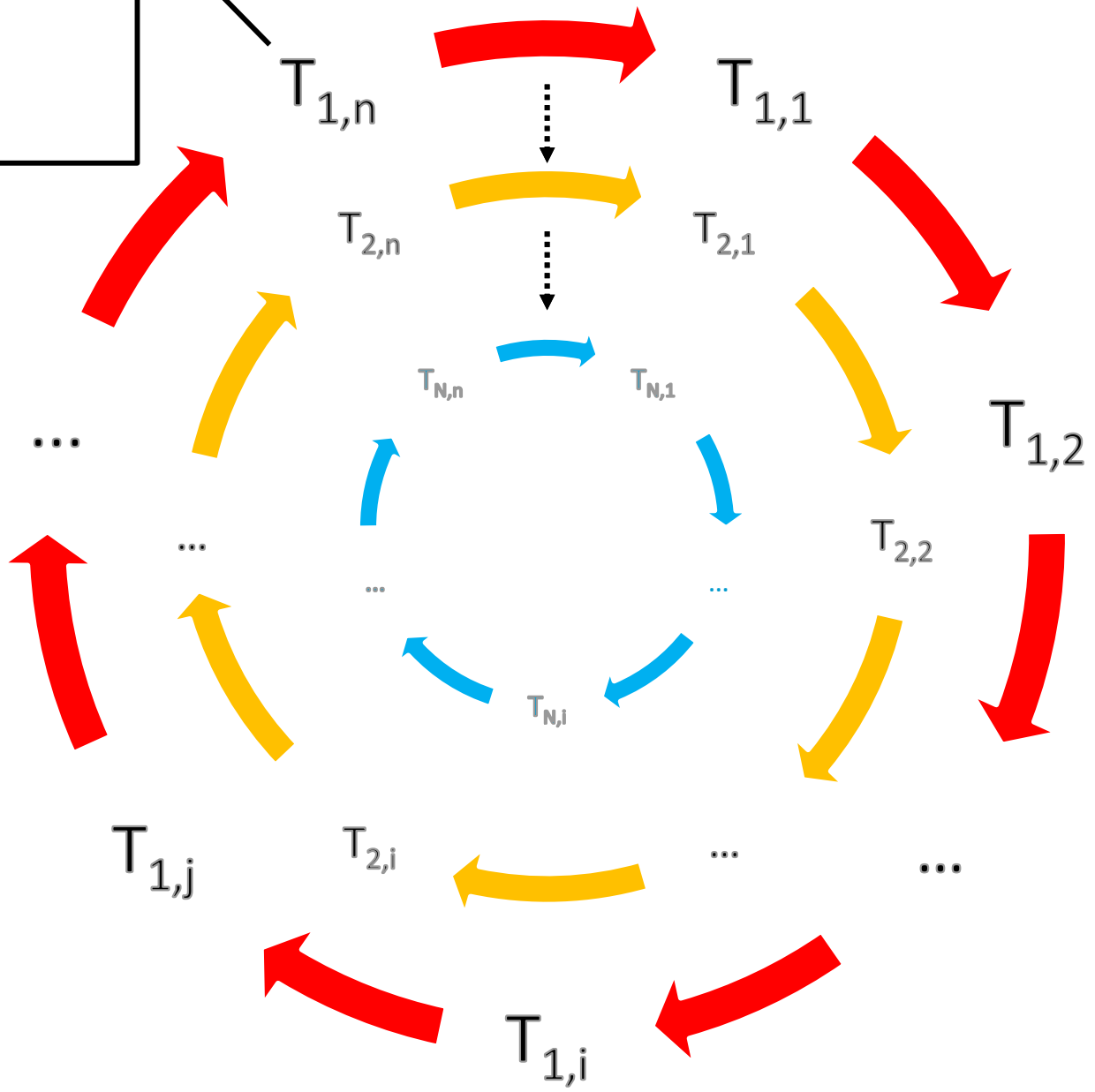




Simulated Annealing

Temperatuur T daalt met tijd t
Per temperatuurdaling
- n oplossingen maken
- beste behouden

N dalingen



Simulated Annealing – Pseudo-code

```
InitializeParameters (Temperature t,  TemperatureReduction α)
initialSolution (Solution s)
s* = s   //best found solution
while t > TMIN
    temperatureIteration = 0
    While temperatureIteration < maxIterations
        s'=SelectNeighbour(s)
        Δ = objectiveFunction(s') - objectiveFunction(s)
                    // objectiveFunction must be minimized
        if (Δ < 0)
        then s = s'
             if objectiveFunction(s') < objectiveFunction(s*)
             then s* = s'
        else if atRandom[0,1] < exp(-Δ/T)
             then s = s'
    end while
    t = α*t
end while
return s*
```

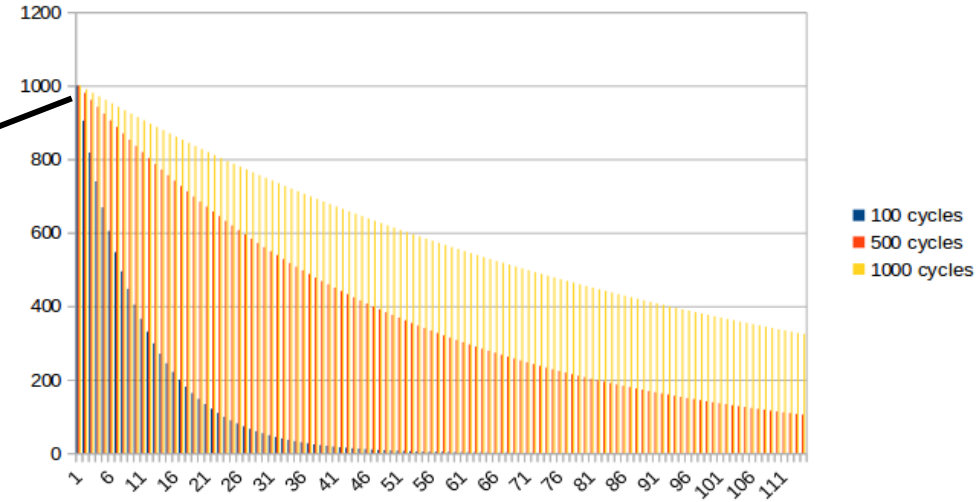
Simulated Annealing - Temperatuurdaling

Twee methoden

1. **Constante temperatuurdaling**: per tijdseenheid temperatuur vaste stapgrootte laten dalen

2. **Exponentiële temperatuurdaling:** kies een begin- (T_s) en eindtemperatuur (T_e) kiezen en verlaag temperaturen telkens $e^{(\frac{\log \frac{T_e}{T_s}}{c-1})}$ met c = aantal cycles (iteraties) per temperatuurwaarde

Daling vanaf 1000 °C



Simulated annealing

In Python: Meerdere Python libraries ondersteunen Simulated Annealing. Een daarvan is de `NDH<II@<G`.

Zie **GitHub - perrygeo/simanneal: Python module for Simulated Annealing optimization** voor meer more informatie omtrent deze package

```
>>> pip install simanneal      # alleen de eerste keer
```

Het doel van de `NDH<II@<G` library is om de probleem-specifieke berekeningen te scheiden van de Metaheuristiek-specifieke berekeningen:

Probleem-specifieke berekeningen

- **move**: hoe van een oplossing naar een buur-oplossing te gaan
- **Energy**: berekent de waarde van de objectieve functie voor een oplossing

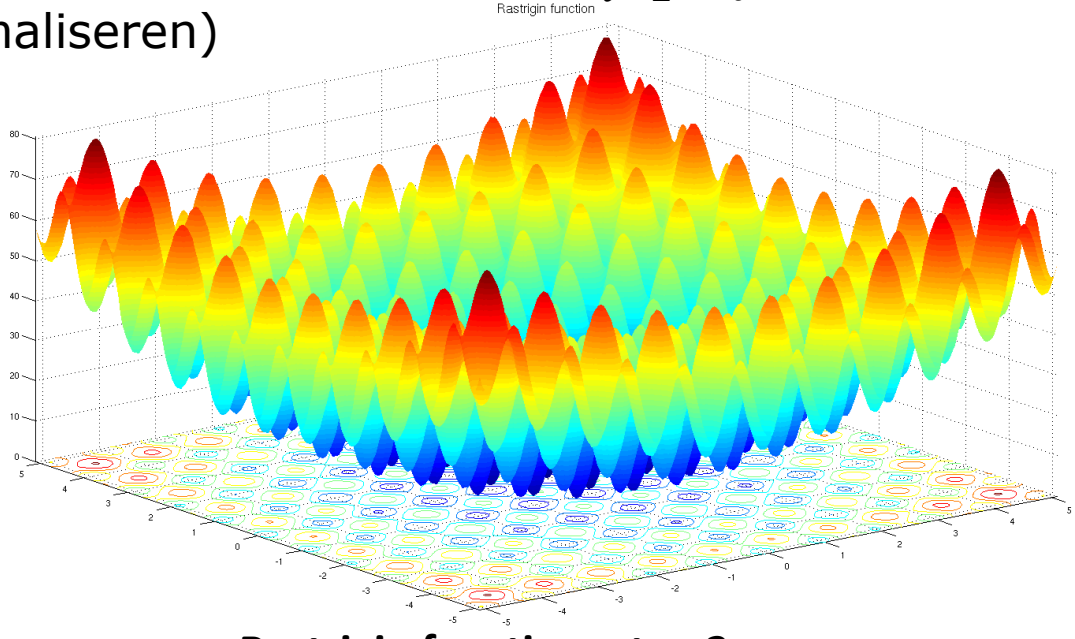
Metaheuristiek-specifieke berekeningen

- Opgeven van de annealing parameters, zo niet, worden er default waarden gebruikt
- Uitvoeren van de simulated annealing heuristiek

Laten we aan de hand van een voorbeeld *simanneal* illustreren:

Rastrigin functie - klassieke case om optimalisatie algoritmen en heuristieken te testen

- $x = (x_1, \ldots, x_i, \ldots, x_n)$ n continue variabelen
- $x_i \in [-5.12, 5.12]$ met i = 1,..,n
- Doelfunctie: $f(x) = 10.n + \sum_{i=1}^{n}(\boldsymbol{x_i}^{\,2} - 10.\cos(2.\pi.x_i))$
  (te minimaliseren)



Rastrigin functie met n=2

Simulated annealing

Voorbeeld: Rastrigin functie

```
>>>from simanneal import Annealer

>>> class RastriginProblem(Annealer):
        def move(self):
            i = np.random.randint(0,len(self.state))
            self.state[i] += np.random.normal(0, 0.1)
            self.state = np.clip(self.state, -5.12, 5.12)

        def energy(self):
            sum = 10 * len(self.state)
            for i in range(0,len(self.state)):
                sum = sum+(self.state[i]**2
                                    -10*math.cos(2*math.pi*self.state[i]))
            return sum

>>> init_sol = np.random.uniform(-5.12, 5.12, size=2) #initiele oplossing
>>> rastrigin = RastriginProblem(init_sol)
>>> # opgeven van de annealing parameters. Zo niet: default waarden
>>> rastrigin.anneal()
```

Simulated annealing

We hebben in ons voorbeeld verschillende default waarden gebruikt, maar deze kunnen worden gewijzigd - voordat het annealing proces wordt uitgevoerd -:

- Tmax = 25000.0 # Max (start) temperatuur
- Tmin = 2.5 # Min (eind temperatuur)
- temperature steps = 50000 # Aantal iteraties
- updates = 100 # Aantal updates (per default wordt een update geprint op stdout)

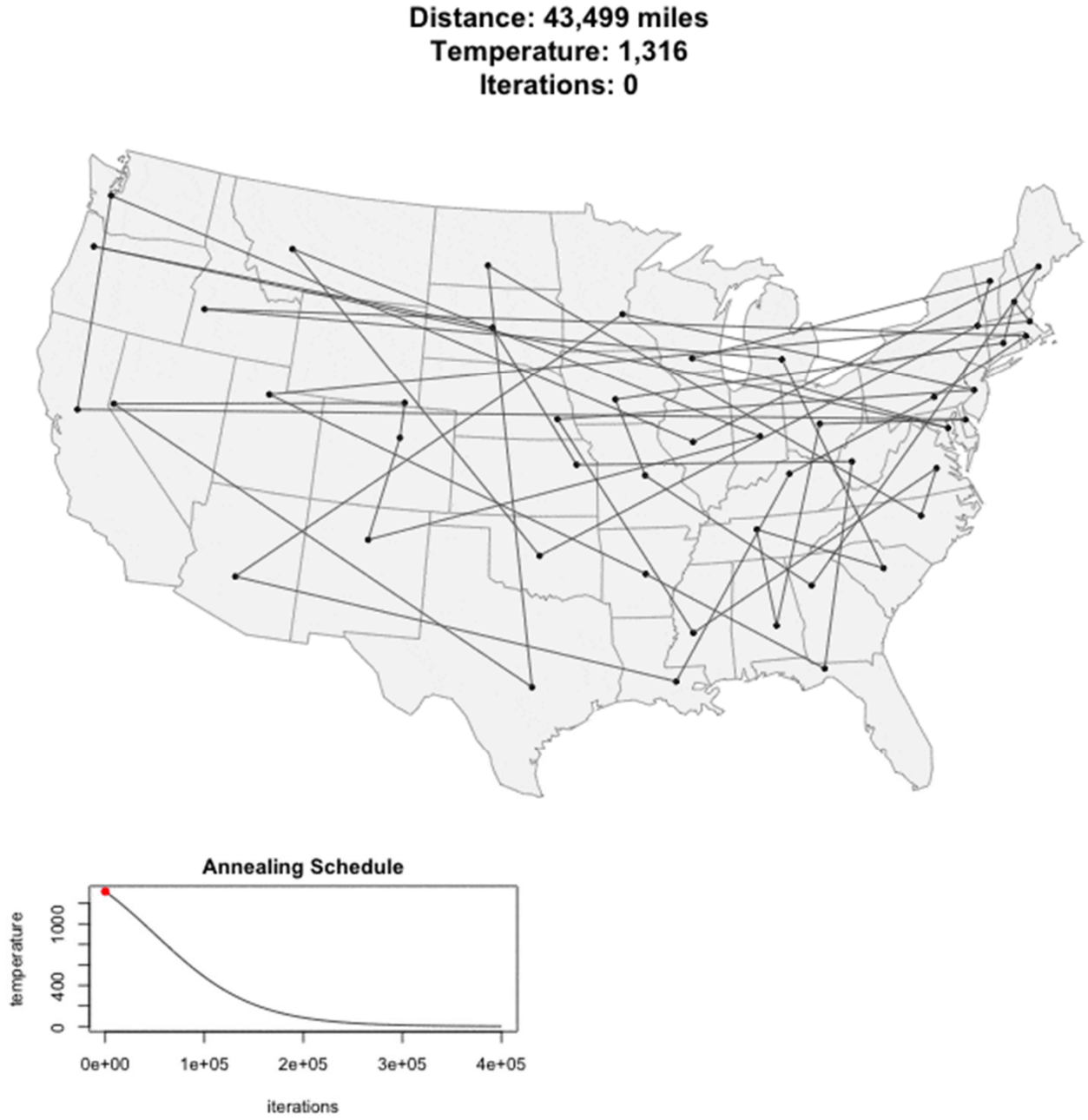
Dit zijn attributen van het object rasterin. Vb.:

```
>>> rastrigin.Tmin = 25.0
```

Opmerkingen:

- Per default, wordt de *energy* functie **geminimaliseerd**, dus dient het resultaat van de objectieve functie met -1 te worden vermenigvuldigd indien de objectieve functie gemaximaliseerd dient te worden
- **Boundaries** op de waarden van de oplossing kunnen in de *move* functie worden opgenomen
- **Andere constraints** naast boundaries op de waarden van een oplossing dienen in de *energy* functie (= objectieve functie) te worden opgenomen.

Simulated annealing - TSP



https://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/

KdG Karel de Grote Hogeschool