

Data Science 2

Meta-heuristieken (deel 2)

Wim De Keyser

Geert De Paepe

Jan Van Overveld

Quote van de week

"Heuristic is an algorithm in a clown suit. It's less predictable, it's more fun, and it comes without a 30-day, money-back guarantee."

Steve McConnell (19??-)



Agenda

1. Genetische algoritmen



Genetische algorithmen

Abstract black geometric shapes in the top right corner, including a diagonal bar and a vertical bar meeting a horizontal base.

Genetisch Algoritme - Terminologie

Belangrijke begrippen

Gen: eigenschap van een oplossing.

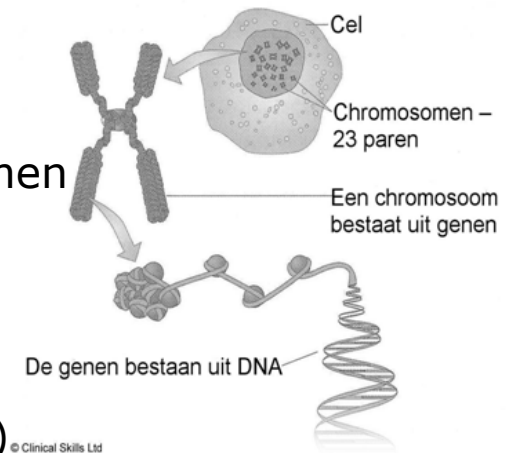
Chromosoom: reeks van genen.

Individu: gedetermineerd door zijn chromosomen

Populatie: verzameling van individuen

Generaties: opeenvolging van populaties

Mating: ontstaan van een nieuw individu (kind) van 2 oude individuen (ouders) door kruising van hun chromosomen.




Paring: ontstaan van nieuw individu uit 2 oude individuen door crossover van hun chromosomen.

Mutatie: willekeurige gen-verandering in chromosoom van een individu.

Fitness functie: functie die de geschiktheid van de individuen bepaalt

Selectiemechanisme: bepaalt welke individuen mogen paren

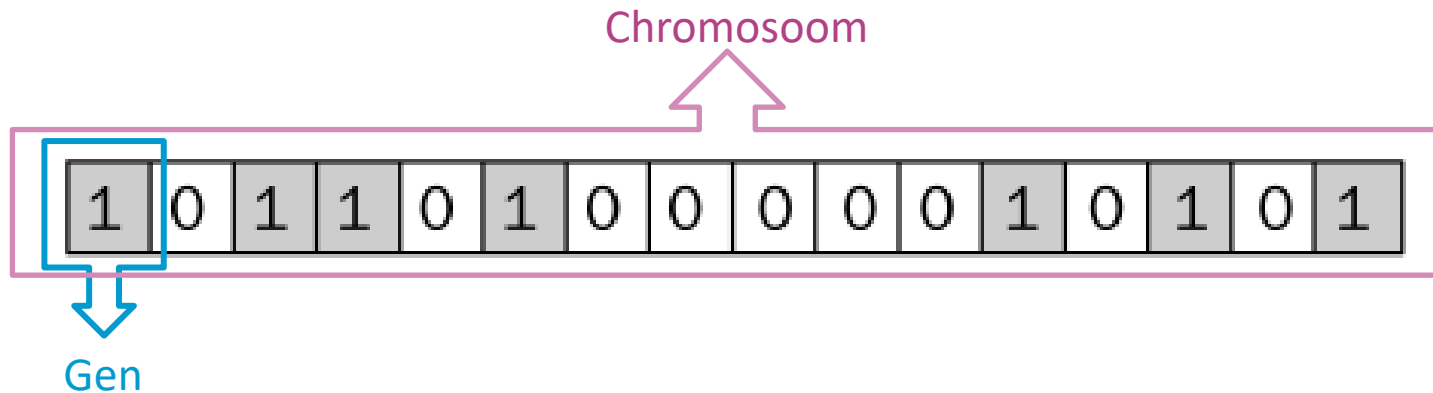
Genetisch Algoritme - Stappenplan

1. Stel het probleem voor als een **chromosoom**.
 2. Bepaal de **fitnessfunctie** voor je probleem
 3. Maak willekeurige **initiële populatie** van chromosomen
 4. Bepaal de **geschiktheid** van alle individuen in de huidige populatie
 5. Selecteer twee individuen volgens het **selectiemechanisme**
 6. Maak twee nieuwe individuen m.b.v. **crossover**. Pas indien nodig **mutatie** toe.
 7. Plaats de twee nieuwe individuen in de populatie van de **nieuwe individuen**
 8. Ga terug naar stap 5 tot er evenveel **nieuwe individuen zijn** als er uit de huidige populatie verwijderd zullen worden.
 9. **Vervang** in de huidige populatie individuen die minder geschikt zijn door de nieuwe individuen. Dit wordt nu de volgende generatie.
 10. Herhaal vanaf stap 4 en ga door totdat een **eindcriterium** bereikt is.
- 

Stap 1: Chromosoomvoorstelling

Chromosoom bevat “genetische” informatie voor een individu

- Een vector van 0'en en 1'en van *vaste* lengte
- Populatie bestaat uit heel reeks individuen/chromosomen



0 = eigenschap is afwezig

1 = eigenschap is aanwezig

Van elk chromosoom kan geschiktheid berekend worden met de fitnessfunctie (zie verder)

Probleem: niet altijd makkelijk om een goede chromosoomvoorstelling te vinden

Stap 2: Fitnessfunctie

Fitnessfunctie f kwalificeert een individu/chromosoom

- Betere individuen moeten een *gunstigere* waarde krijgen
- $fitheid = f(individu)$

Wordt per generatie toegepast op elk individu

- Chromosoom van individu wordt geëvalueerd.

Bepaalt *indirect* welke individuen mogen paren

- Indirect, want het selectiemechanisme (stap 5) bepaalt dat

Stap 5 : Selectiemechanisme

Welke individuen mogen paren?

Natuurlijke selectie: meest geschikte exemplaren moeten meer kans hebben om te paren.

- Minder geschikte exemplaren hebben nog steeds kans om te mogen paren.
Kunnen delen van oplossing bevatten.
- Selectiemechanisme bepaalt kansen

Selectiemechanismen: Roulettewielselectie, Tournamentselectie, ...

Elitisme: klein percentage van beste individuen gaat per definitie over naar volgende generatie zonder onderhevig te zijn aan selectie

Voorbeeld Stap 5

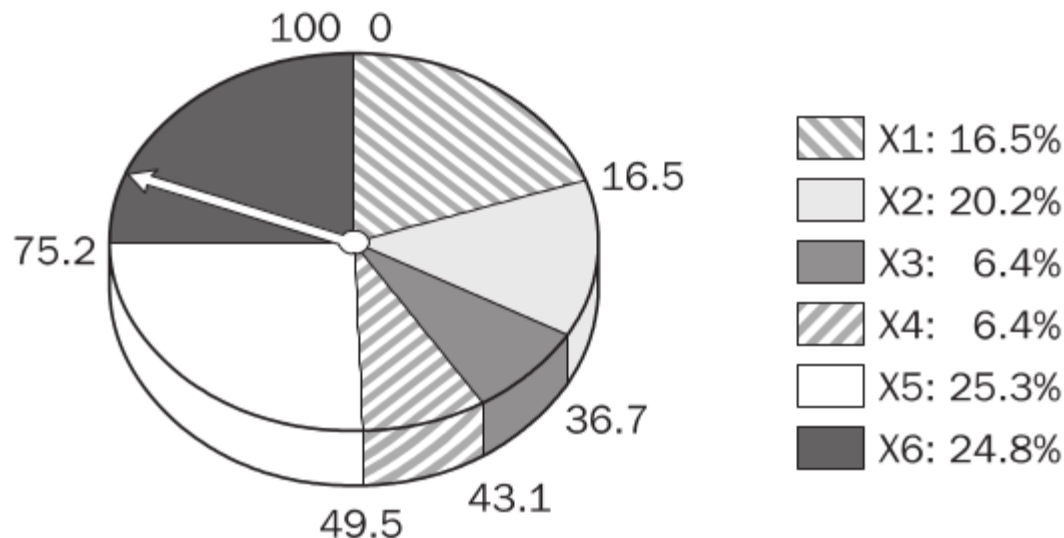
Roulettewiel

Cumulatieve som van fitheidscores van alle individuen in populatie

→ hierdoor ontstaan intervallen (zie Data Science 1)

→ Individu met grotere fitheid heeft grotere kans om gekozen te worden.

Een virtuele draai aan het wiel of pijl selecteert een individu ($X_1 \rightarrow X_6$)



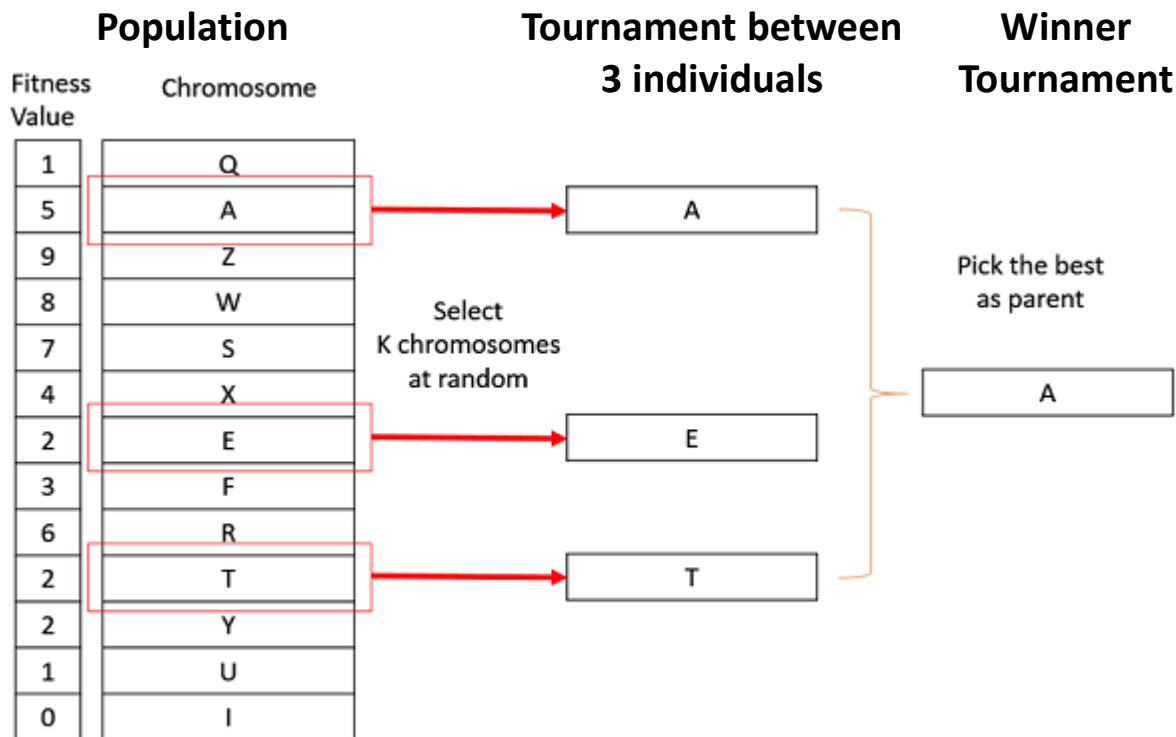
Voorbeeld Stap 5

Tournament

"Organiseer" tornooien tussen k individuen van een populatie

→ de beste wordt geselecteerd voor 'mating'

→ iedereen heeft een kans om te worden geselecteerd voor het tornooi



Stap 6 : Biologische replicatie

Mensen hebben twee exemplaren van elk chromosoom (**diploïde wezen**)

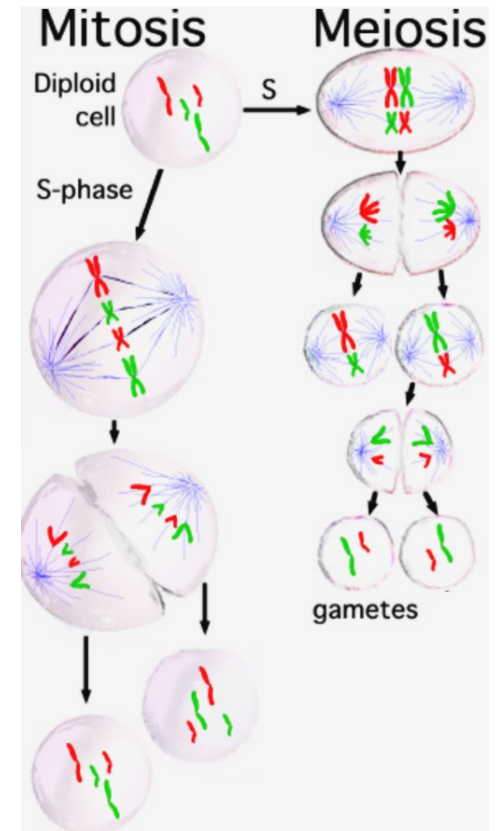
→ een veiligheidskopie van elk chromosoom

→ 23 chromosoomparen en 46 chromosomen in totaal.

Voorbeeld: chromosoom 21 komt in feite tweemaal voor: deel van vader en deel van moeder.

Er zijn **twee mechanismen** om cellen te kopiëren:

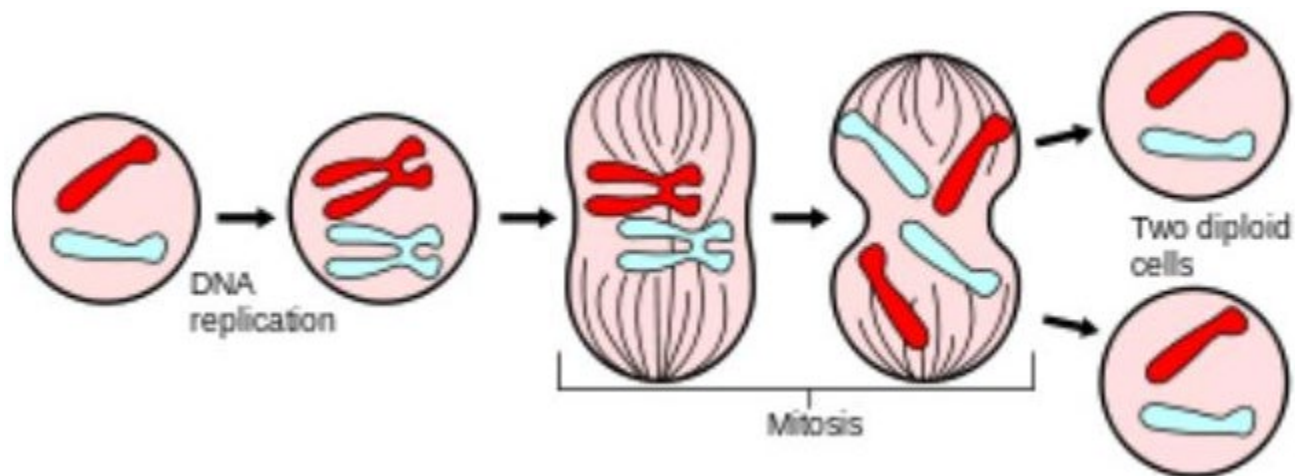
- 1. Mitose** is normale verdubbeling van een diploïde cel. *Identieke* kopieën worden gevormd door de DNA strengen te kopiëren
- 2. Meiose** is de manier om van één diploïde cel vier **haploïde** cellen te maken, zogenaamde gameten of geslachtscellen. Deze gameten bevatten dus slechts 23 chromosomen.



Stap 6: Mitose

Mitose is het kopiëren van genetische informatie

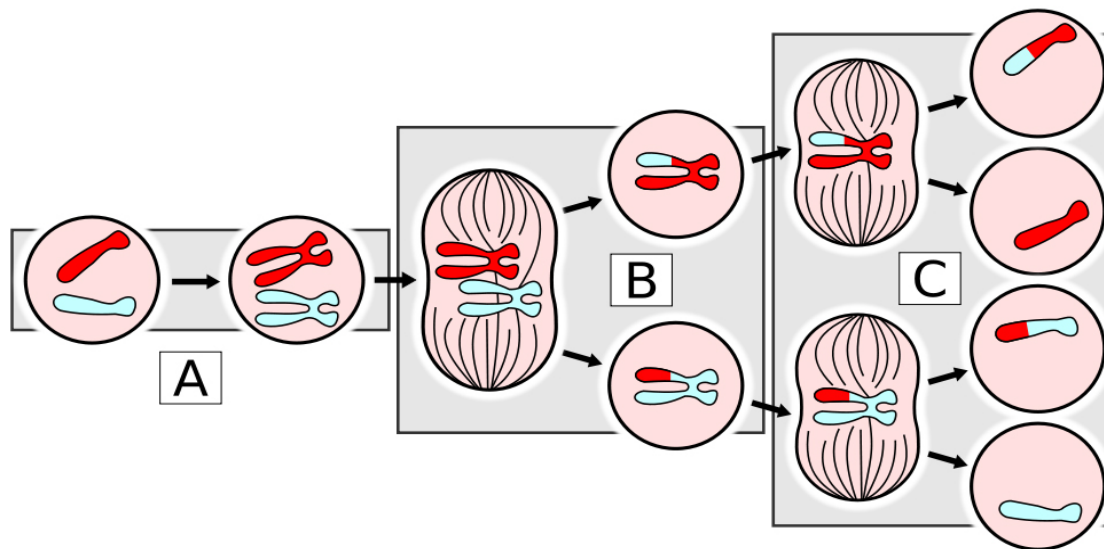
- De DNA keten waaruit chromosoom bestaat, breekt open zoals een rits. Ontbrekende delen worden bijgemaakt. Hier kunnen fouten gebeuren (**mutatie**).
- Chromosoom ont dubbelt zich in twee gelijkwaardige chromatiden die nog aan elkaar hangen ter hoogte van centromeer.
- De chromatiden worden uit elkaar getrokken en vormen twee identieke nieuwe dochterchromosomen.



Stap 6: Meiose

Meiose vindt plaats in de geslachtsorganen en gebeurt in twee fasen:

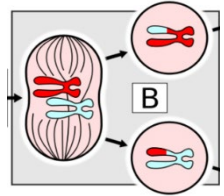
1. Meiose I: Elk paar chromosomen verdubbelt zich (zoals bij mitose, fig. A). Hier kan **cross-over** plaats vinden. Dit is een uitwisseling van *overeenkomend* genen uit het chromosoompaar. Dit leidt tot een mix van oudergenen (rood en lichtblauw). De chromosoomparen worden uiteen gehaald tot twee diploïde dochtercellen (fig. B).
2. Meiose II: In elk van de twee ontstane dochtercel worden de chromatiden uiteen getrokken en zo ontstaan vier haploïde geslachtscellen (fig. C).



Stap 6: Replicatie bij GA's – cross-over

Replicatie en cross-over van twee ouderchromosomen

Genetische informatie van de ouders wordt doorgegeven aan de kinderen.



	1	2	3	4	5	6	7	8	9	10
Jouw vader	0	0	0	1	1	0	1	0	1	1
Jouw moeder	1	1	0	0	0	0	0	1	1	1

Willekeurige snijvlakken in de chromosomen

	1	2	3	4	5	6	7	8	9	10
Jouw 1 ^e kind	0	0	0	0	0	0	0	0	1	1
Jouw 2 ^e kind	1	1	0	1	1	0	1	1	1	1

Stap 6 : Replicatie bij GA's - mutatie

Mutatie

Een toevallige fout ontstaat in het DNA

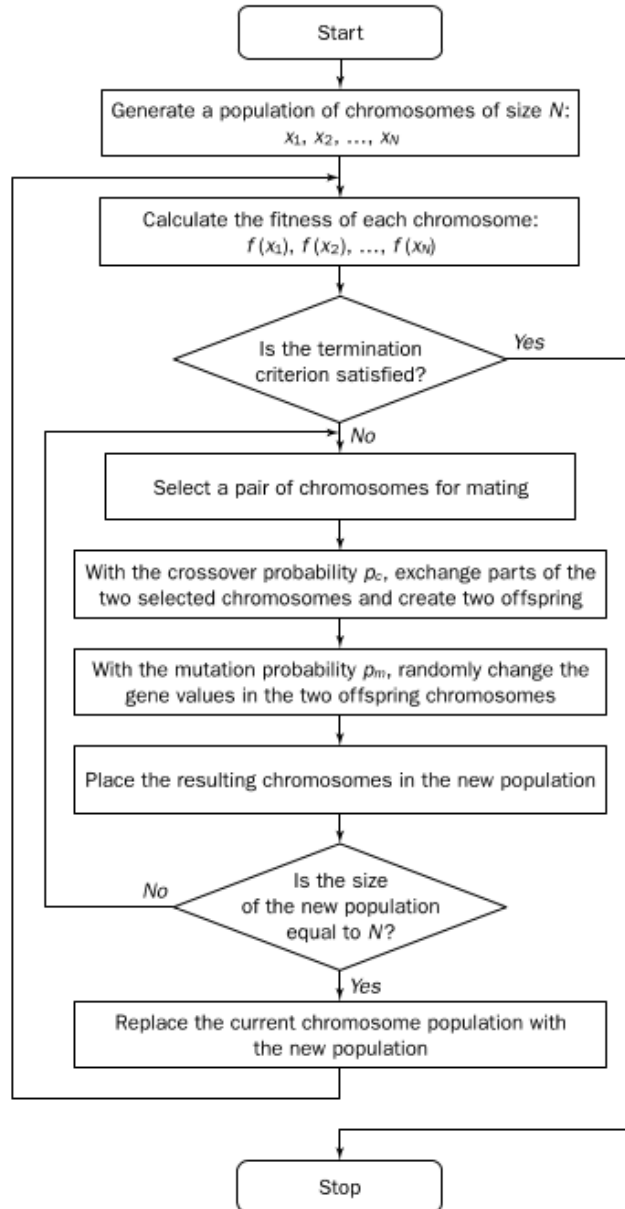
	1	2	3	4	5	6	7	8	9	10
Individu	0	0	0	1	1	0	1	0	1	1



Kans op mutatie
is klein.
 $p_M = 1\% \text{ à } 20\%$

	1	2	3	4	5	6	7	8	9	10
Individu	0	0	0	1	0	0	1	0	1	1

Genetische algoritmen - Flow Chart



Genetische algoritmen – Pseudo-code

```
InitializeParameters (maxLengthPopulation, maxReproductions, maxMutations,
                    stopCriterionParameter)
initialPopulation (PopulationListOfSolutions, populationList, maxLengthPopulation)
s* = bestSolutionInPopulation (populationList)  //best found solution
repeat
    offSpringList = {}
    for i = 1 to maxReproductions
        SelectTwoParents (populationList, s1, s2)
        offSpringi = createOffSpring (s1, s2)    //using crossover
        addToOffSpringList (offSpringi)
    end for
    for j = 1 to maxMutations
        offSpringj = selectAnOffSpring (offSpringList)
        offSpringj = mutate (offSpringj)
        replaceInOffSpringList (offSpringj)
    end for
    for j = 1 to lenght (offSpringList)
        offSpringj = selectAndRemovefirst (offSpringList)
        weakest = weakestIndividualInPopulation (populationList)
        if objectiveFunction (offSpringj) < objectiveFunction (weakest)
            // objectiveFunction must be minimized
            then replaceInPopulation (populationList, weakest, offSpringj)
    end for
    s' = bestSolutionInPopulation (populationList)
    if objectiveFunction (s') < objectiveFunction (s*)
        then s* = s'
    Update (stopCriterionParameter)
until stopCriterion (stopCriterionParameter)
return s*
```

Genetische algoritmen - Genetische algoritmen in Python

Installeer, laad en bestudeer de [inspyred](#) package van Python. De volgende stappen zijn noodzakelijk om goede oplossingen te kunnen vinden:

1. Creëer de initiële populatie met behulp van de gespecificeerde kandidaatzaden en de **GENERATOR**
2. Evalueer de initiële populatie met behulp van de **EVALUATOR**
3. Zet de teller voor het aantal generaties op 0
4. Roep de **OBSERVER** op met de eerste populatie
5. Zolang de **TERMINATOR** niet waar is, wordt de lus uitgevoerd:
 - Kies ouders via de **SELECTOR**
 - Bepaal nakomelingen van de ouders
 - Voor elke **VARIATOR**, voer de lus uit:
 - Stel nakomelingen in op de output van de **VARIATOR**
 - Evalueer nakomelingen met behulp van de **EVALUATOR**
 - Vervang individuen in de huidige populatie met behulp van de **REPLACER**
 - Migreren van individuen in de huidige populatie met behulp van de **MIGRATOR**
 - Archiveer individuen in de huidige populatie met behulp van de **ARCHIVER**
 - Verhoog de teller voor het aantal generaties
 - Roep de **OBSERVER** op met de huidige populatie

Genetische algoritmen - Genetische algoritmen in Python

De inspyre library scheidt de probleem-specifieke berekeningen van de metaheuristiek-specifieke berekeningen:

Probleem-specifieke berekeningen

- Een **GENERATOR** die definieert hoe oplossingen worden gemaakt
- Een **EVALUATOR** die definieert hoe fitnesswaarden worden berekend voor oplossingen

Metaheuristiek-specifieke berekeningen

- Een **OBSERVER** die definieert hoe de gebruiker de toestand van de evolutie kan volgen
- Een **TERMINATOR** die bepaalt of de evolutie moet eindigen
- Een **SELECTOR** dat bepaalt welke individuen ouders moeten worden
- Een **VARIATOR** die bepaalt hoe nakomelingen worden gecreëerd uit bestaande individuen
- Een **REPLACER** die bepaalt welke individuen moeten overleven in de volgende generatie
- Een **MIGRATOR** die definieert hoe oplossingen worden overgedragen tussen verschillende populaties
- Een **ARCHIVER** die definieert hoe bestaande oplossingen worden opgeslagen buiten de huidige populatie

Genetische algoritmen - Genetische algoritmen in Python

Wanneer je inspyre gebruikt, moet je altijd ten minste de probleem specifieke functies schrijven: een **generate-functie** en een **evaluate-functie**.

Deze hebben altijd een bepaalde vooraf opgelegde header:

```
def generate(random = None, args = None) -> []:  
    # hier komt uw code zodat bij elke aanroep naar deze functie  
    # een willekeurige oplossing wordt teruggegeven onder de vorm van een lijst
```

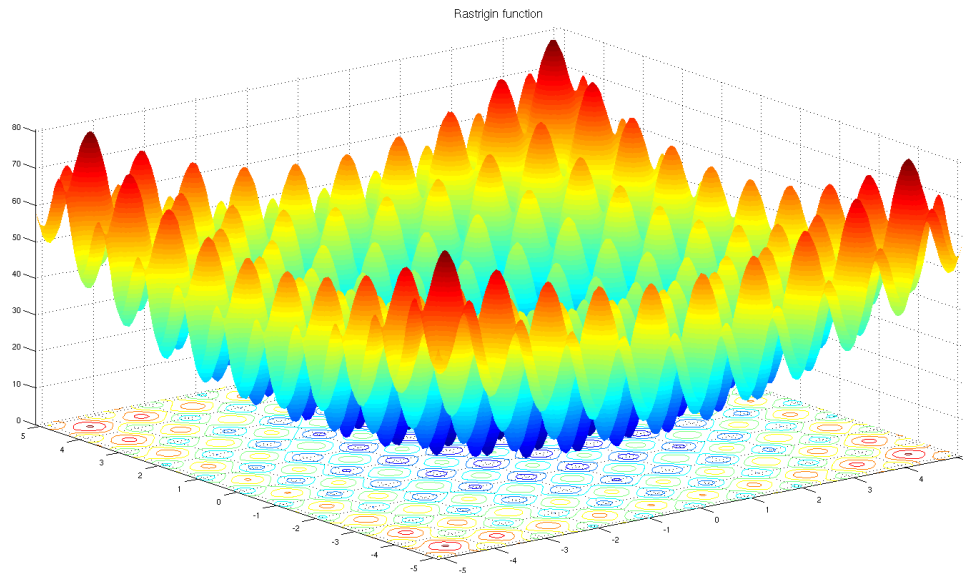
```
def evaluate(candidates, args = {}):  
    # hier komt uw code zodat bij elke aanroep naar deze functie de fitness van  
    # een oplossing wordt teruggegeven
```

Wanneer je inspyre gebruikt, moet je altijd de algoritmespecifieke parameters opgeven, zelfs als je ze niet nodig hebt.

Genetische algoritmen

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

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



Rastrigin functie met n=2

Genetische algoritmen

Voorbeeld: Rastrigin functie

#Probleem specifiek

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

def generate(random = None, args = None) -> []:
    size = args.get('num_inputs',10)
    return np.random.uniform(low=-5.12, high=5.12, size=size)

def evaluate(candidates, args = {}):
    fitness = []
    for candidate in candidates:
        fitness.append(obj_func(candidate))
    return fitness
```

Voorbeeld: Rastrigin functie

#metaheuristiek-specifiek

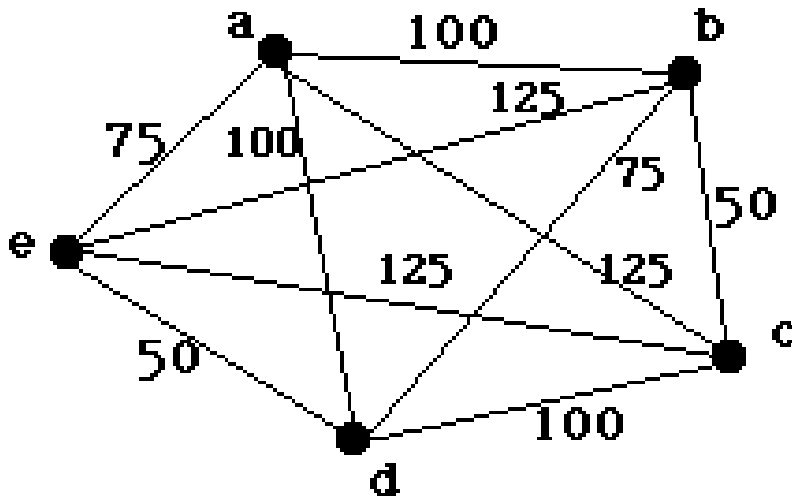
```
import inspyred
from inspyred import ec #ec staat voor Evolutionary computation
from random import Random
rand = Random()
ga = ec.GA(rand)
ga.terminator = ec.terminators.evaluation_termination
ga.variator = [ec.variators.n_point_crossover, ec.variators.bit_flip_mutation]
population: [ec.Individual] = ga.evolve(
    generator=generate,
    evaluator=evaluate,
    selector = ec.selectors.tournament_selection, #optioneel
    pop_size=100,
    maximize=False,
    bounder=ec.Bounder(-5.12, 5.12),
    max_evaluations=10000,
    mutation_rate=0.25,
    num_inputs=30)
population.sort(reverse=True)
print(population[0])
```


Genetische algoritmen

Voorbeeld: Traveling salesman problem omvormen tot een 0-1 lineair probleem

Afstandsmatrix (in een vector geplaatst):

```
>>> a = [0,100,125,100,75,100,0,50,75,125,125,50,0,100,125,  
        100,75,100,0,50,75,125,125,50,0]
```



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

Genetische algoritmen

Voorbeeld: Traveling salesman problem omvormen tot een 0-1 lineair probleem

➤ **Oplossing:** matrix zelfde vorm met 0 en 1

➤ **Constraints** (lineaire vergelijkingen)

- ↪ mag niet in een stad blijven
- ↪ moet uit elke stad 1 keer vertrekken
- ↪ moet in elke stad 1 keer aankomen

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

Matrix a met afstanden

➤ **Objectieve functie:**

- Te minimaliseren $\sum_{i=1}^n \sum_{j=1}^n (a_{ij} x_{ij})$
- Afstraffen als constraints verbroken worden

	a	b	c	d	e
a	0	0	0	0	1
b	1	0	0	0	0
c	0	1	0	0	0
d	0	0	1	0	0
e	0	0	0	1	0

Matrix x met oplossing: 1 stelt een verplaatsing voor

Genetische algorithmen

```
def objective_function(solution, weights):
    n=int(math.sqrt(len(solution)))
    #Leave each city once
    leaveOK=np.empty((n,1), dtype=int)
    for i in range(0,n):
        index = range(i,n*n, n)
        leaveOK[i] = 0
        for j in index: leaveOK[i] = leaveOK[i] + solution[j]
    #Arrive in each city once
    arriveOK=np.empty((n,1), dtype=int)
    for i in range(0,n):
        index = range(i*n,(i+1)*n, 1)
        arriveOK[i] = 0
        for j in index: arriveOK[i] = arriveOK[i] + solution[j]
    #Never stay in a city
    index = range(0,n*n, n+1)
    notStayingOK = 0
    for j in index: notStayingOK = notStayingOK + solution[j]
    #No subloops or infinite loop but one loop with length n
    loop_length = 0;
    city=0
    in_loop = True
    while(in_loop & (loop_length < n+1)):
        loop_length = loop_length + 1
        index = range(city*n,(city+1)*n, 1) # row of city
        next_city = 0
        while ((solution[index[next_city]] == 0) & (next_city < n-1)): next_city=next_city+1
        in_loop = (next_city != 0) & (solution[index[next_city]] == 1)
        city = next_city
    #Test if all 4 of the conditions are fulfilled
    if ((notStayingOK == 0) & (np.min(arriveOK) == 1) & (np.max(arriveOK) == 1) & (np.sum(arriveOK) == n)
        & (np.min(leaveOK) == 1) & (np.max(leaveOK) == 1) & (np.sum(leaveOK) == n)
        & (loop_length == n)):
        score=np.sum(np.multiply(solution, weights)) #value objective function
    else:
        score=1000*n #not a feasible solution, so very bad value for the objective function
    return score
```

Genetic Algorithm

Voorbeeld: Traveling salesman problem

#metaheuristiek-specifiek

```
import inspyred
from inspyred import ec
from random import Random

rand = Random()
ga = ec.GA(rand)
ga.terminator = ec.terminators.evaluation_termination
ga.variator = [ec.variators.n_point_crossover, ec.variators.bit_flip_mutation]
population: [ec.Individual] = ga.evolve(
    generator=generate,
    evaluator=evaluate,
    selector = ec.selectors.tournament_selection, # optioneel
    tournament_size=10
    pop_size=1000,
    maximize=False,
    bounder=ec.Bounder(0, 1),
    max_evaluations=20000,
    mutation_rate=0.01,
    num_cities=5,
    distance_matrix=distance_matrix)
population.sort(reverse=True)
print(population[0])
```

