faerbeproblem

October 21, 2025

[193]:
```python
#Faerbe-Problem
import random
import numpy as np

max_farben = 5
wahrscheinlichkeit_rekombinieren = 0.6
wahrscheinlichkeit_mutieren = 0.01
umwandlung = {
    1: [0, 0, 1],
    2: [0, 1, 0],
    3: [0, 1, 1],
    4: [1, 0, 0],
    5: [1, 0, 1]
}
rueckumwandlung = {
    (0, 0, 1):1,
    (0, 1, 0):2,
    (0, 1, 1):3,
    (1, 0, 0):4,
    (1, 0, 1):5
}


def generiere_start_population(population_size, individuum_size):
    population = []  # leere Liste Pop
    for index_pop in range(population_size):
        individuum = []  # leere Liste Indi
        for index_ind in range(individuum_size):
            ziffer = random.choice([1, 2, 3, 4, 5])
            individuum.append(ziffer)
        population.append(individuum) #append wg leerer Liste
    return population


def berechne_Fitness_individuum(population, population_size, individuum_size):
    global max_farben
    fitness_jed_indi = {}
```

```
    for index_pop in range(population_size):
        fitness_indi = 0
        anzahl_farben = 0
        #Farb Werte entpacken
        individuum = population[index_pop]
        farbeA, farbeB, farbeC, farbeD, farbeE, farbeF = individuum

        #Bedingungen pruefen, Werte verteilen
        if(farbeA != farbeB and farbeA != farbeC):
            fitness_indi += 2
        if(farbeA == farbeB and farbeA != farbeC or farbeA != farbeB and farbeA␣
↪== farbeC):
            fitness_indi += 1
        #beides gleich: +0
        if(farbeB != farbeC and farbeB != farbeD):
            fitness_indi += 2
        if(farbeB == farbeC and farbeB != farbeD or farbeB != farbeC and farbeB␣
↪== farbeD):
            fitness_indi += 1
        #beides gleich: +0
        if(farbeC != farbeD):
            fitness_indi += 1
        if(farbeD != farbeB and farbeD != farbeE):
            fitness_indi += 2
        if(farbeD == farbeB and farbeD != farbeE or farbeD != farbeB and farbeD␣
↪== farbeE):
            fitness_indi += 1

        #anzahl farben durch länge set (einzigartige elemente)
        individuum = population[index_pop]
        anzahl_farben = len(set(individuum))

        if(anzahl_farben < max_farben):
            fitness_indi += 5
            max_farben = anzahl_farben

        fitness_jed_indi[index_pop] = fitness_indi
    return fitness_jed_indi


def berechne_wahrscheinlichkeit(fitness_werte, fitness_gesamt):
    wahrscheinlichkeiten = {}
    intervalle = {}
    summand = 0

    for index in range(len(fitness_werte)):
        wahrscheinlichkeit = fitness_werte[index]/fitness_gesamt
```

```
        wahrscheinlichkeiten[index] = wahrscheinlichkeit
        #summiere die wahrscheinlichkeiten um intervallgrenzen zu berechnen
        grenze = wahrscheinlichkeit + summand
        intervalle[index] = grenze
        summand = grenze
    #print("Wahrsch:", wahrscheinlichkeiten)
    #print("Intervall", intervalle)
    return intervalle


def selektiere(intervalle):
    gewaehltes_individuum = 0
    zufallszahl = random.random()
    #print("Zufall:", zufallszahl)
    for index in range(len(intervalle)):
        if(zufallszahl < intervalle[index]):
            gewaehltes_individuum = index
            break
    return gewaehltes_individuum


def rekombiniere(mutter, vater):
    global wahrscheinlichkeit_rekombinieren
    zufallszahl = random.random()
    if(wahrscheinlichkeit_rekombinieren < zufallszahl):
        return [mutter, vater]

    kind1 = mutter.copy()
    kind2 = vater.copy()

    zufallszahl = random.randint(1, len(mutter)-2) #sollte zw 1 und 4 sein
 ↪länge mutter = 6
    kind1[zufallszahl:] = vater[zufallszahl:]
    kind2[zufallszahl:] = mutter[zufallszahl:]

    return[kind1, kind2]


def mutiere(kinder, individuum_size):
    global wahrscheinlichkeit_mutieren
    global umwandlung
    global rueckumwandlung
    kinder_binaer = []

    #zahlen in binär umwandeln
    for index_kinder in range(len(kinder)):
        kind_binaer = []
```

```
        for index_kind in range(individuum_size):
            zahl = kinder[index_kinder][index_kind]

            for index in range(3):
                kind_binaer.append(umwandlung[zahl][index])

        kinder_binaer.append(kind_binaer)

    for index_kinder in range(len(kinder_binaer)):
        kind_binaer = kinder_binaer[index_kinder]
        for index_bit in range(len(kind_binaer)):
            zufallszahl = random.random()
            if(zufallszahl < wahrscheinlichkeit_mutieren):
                if(kind_binaer[index_bit] == 0):
                    kind_binaer[index_bit] = 1
                elif(kind_binaer[index_bit] == 1):
                    kind_binaer[index_bit] = 0

        kinder_binaer[index_kinder] = kind_binaer

    #zurückübersetzen in zahlen
    for index in range(len(kinder_binaer)):
        kind_binaer = kinder_binaer[index]
        kind_zahlen = []
        for i in range(0, len(kind_binaer), 3):
            binaerzahl = []
            binaerzahl = kind_binaer[i:i+3]
            if(binaerzahl == [1, 1, 1] or binaerzahl == [0, 0, 0] or binaerzahl␣
 ↪== [1, 1, 0]):
                zufallszahl = random.choice([1, 2, 3, 4, 5])
                kind_zahlen.append(zufallszahl)
                continue

            kind_zahlen.append(rueckumwandlung[tuple(binaerzahl)])
        kinder_binaer[index] = kind_zahlen
    return kinder_binaer



def run(generation_size, population_size,individuum_size = 6):
    population = generiere_start_population(population_size, individuum_size)
    #for individuum in population:
        #print(individuum)

    for generationen in range(generation_size):
        population_neu = []
```

```
        #1x pro Generation
        fitness_werte = berechne_Fitness_individuum(population,␣
↪population_size, individuum_size)
        #Dictionary, um Indi einen Wert zuzuordnen
        fitness_gesamt = sum(fitness_werte.values())
        #print(fitness_werte)
        #print(fitness_gesamt)
        intervalle = berechne_wahrscheinlichkeit(fitness_werte, fitness_gesamt)

        #neue Generation erstellen
        for number in range(population_size):

            mutter_index = selektiere(intervalle)
            vater_index = selektiere(intervalle)
            while(mutter_index == vater_index):
                vater_index = selektiere(intervalle)
            #print("gewählt:", mutter_index, vater_index)

            mutter = population[mutter_index]
            vater = population[vater_index]
            kinder = []
            kinder = rekombiniere(mutter, vater)
            #print(mutter, vater)
            kinder = mutiere(kinder, individuum_size)
            #print(kinder)
            population_neu.append(kinder[0])
            population_neu.append(kinder[1])



        population = population_neu
run(100, 50)
```

[ ]:

[192]:
```
#4 Damen Problem
wahrscheinlichkeit_rekombinieren = 0.6
wahrscheinlichkeit_mutieren = 0.01
umwandlung = {
    1: [0, 0],
    2: [0, 1],
    3: [1, 0],
    4: [1, 1],

}
rueckumwandlung = {
    (0, 0):1,
```

```
    (0, 1):2,
    (1, 0):3,
    (1, 1):4,
}
#ich musste nur die gezogenen zahlen anpassen 1-4 statt 1-5
def generiere_start_population(population_size, individuum_size):
    population = []  # leere Liste Pop
    for index_pop in range(population_size):
        individuum = []  # leere Liste Indi
        for index_ind in range(individuum_size):
            ziffer = random.choice([1, 2, 3, 4])
            individuum.append(ziffer)
        population.append(individuum) #append wg leerer Liste
    return population

def berechne_Fitness_individuum(population, population_size, individuum_size):
    fitness_jed_indi = {}
    for index_pop in range(population_size):
        #Konflikte werden abgezogen, der Score bleibt positiv
        fitness_indi = 81
        fehler = 1
        individuum = population[index_pop]
        #aktuelle Dame
        for xKoordinate in range(len(individuum)):
            yKoordinate = individuum[xKoordinate]
            #andere Damen (inklusive aktuelle)
            for xKoordinate2 in range(len(individuum)):
                yKoordinate2 = individuum[xKoordinate2]

                if(xKoordinate == xKoordinate2):
                    continue
                #andere Dame befindet sich in selber Zeile wie aktuelle Dame
                if(yKoordinate == yKoordinate2):
                    fehler += 5

                #die Diagonalen, sind die Seiten des Steigungsdreiecks gleich␣
 ↪lang, liegen die Damen auf einer Diagonalen
                xVersatz = abs(xKoordinate - xKoordinate2)
                yVersatz = abs(yKoordinate - yKoordinate2)

                if(xVersatz == yVersatz):
                    fehler += 5

        fitness_indi -=(fehler)
        fitness_jed_indi[index_pop] = fitness_indi
    return fitness_jed_indi
```

```
def berechne_wahrscheinlichkeit(fitness_werte, fitness_gesamt):
    wahrscheinlichkeiten = {}
    intervalle = {}
    summand = 0

    for index in range(len(fitness_werte)):
        wahrscheinlichkeit = fitness_werte[index]/fitness_gesamt
        #summiere die wahrscheinlichkeiten um intervallgrenzen zu berechnen
        grenze = wahrscheinlichkeit + summand
        intervalle[index] = grenze
        summand = grenze
    #print("Wahrsch:", wahrscheinlichkeiten)
    #print("Intervall", intervalle)
    return intervalle

def selektiere(intervalle):
    gewaehltes_individuum = 0
    zufallszahl = random.random()
    #print("Zufall:", zufallszahl)
    for index in range(len(intervalle)):
        if(zufallszahl < intervalle[index]):
            gewaehltes_individuum = index
            break
    return gewaehltes_individuum

def rekombiniere(mutter, vater):
    global wahrscheinlichkeit_rekombinieren
    zufallszahl = random.random()
    if(wahrscheinlichkeit_rekombinieren < zufallszahl):
        return [mutter, vater]

    kind1 = mutter.copy()
    kind2 = vater.copy()

    zufallszahl = random.randint(1, len(mutter)-1) #sollte zw 1 und 4 sein␣
↪länge mutter = 4
    kind1[zufallszahl:] = vater[zufallszahl:]
    kind2[zufallszahl:] = mutter[zufallszahl:]

    return[kind1, kind2]

def mutiere(kinder, individuum_size):
    global wahrscheinlichkeit_mutieren
    global umwandlung
    global rueckumwandlung
    kinder_binaer = []
```

```
    #zahlen in binär umwandeln
    for index_kinder in range(len(kinder)):
        kind_binaer = []

        for index_kind in range(individuum_size):
            zahl = kinder[index_kinder][index_kind]

            for index in range(2):
                kind_binaer.append(umwandlung[zahl][index])

        kinder_binaer.append(kind_binaer)

    for index_kinder in range(len(kinder_binaer)):
        kind_binaer = kinder_binaer[index_kinder]
        for index_bit in range(len(kind_binaer)):
            zufallszahl = random.random()
            if(zufallszahl < wahrscheinlichkeit_mutieren):
                if(kind_binaer[index_bit] == 0):
                    kind_binaer[index_bit] = 1
                elif(kind_binaer[index_bit] == 1):
                    kind_binaer[index_bit] = 0

        kinder_binaer[index_kinder] = kind_binaer

    #zurückübersetzen in zahlen
    for index in range(len(kinder_binaer)):
        kind_binaer = kinder_binaer[index]
        kind_zahlen = []
        for i in range(0, len(kind_binaer), 2):
            binaerzahl = []
            binaerzahl = kind_binaer[i:i+2]

            kind_zahlen.append(rueckumwandlung[tuple(binaerzahl)])
        kinder_binaer[index] = kind_zahlen
    return kinder_binaer


def run(generation_size, population_size,individuum_size = 4):
    population = generiere_start_population(population_size, individuum_size)

    best = None
    bestFit = 0

    #setup Generation: Fitness berechnen, wahrscheinlichkeit aufstellen
    for generationen in range(generation_size):
        population_neu = []
        #for individuum in population:
```

```
            #print(individuum)
        fitness_werte = berechne_Fitness_individuum(population,␣
↪population_size, individuum_size)
        print(fitness_werte)
        fitness_gesamt = sum(fitness_werte.values())
        intervalle = berechne_wahrscheinlichkeit(fitness_werte, fitness_gesamt)

        for i in range(len(fitness_werte)):
            fit = fitness_werte[i]
            if(fit>bestFit):
                bestFit = fit
                best = population[i]

        #neue Generation aus alter erzeugen
        for number in range(population_size):


            mutter_index = selektiere(intervalle)
            vater_index = selektiere(intervalle)
            while(mutter_index == vater_index):
                vater_index = selektiere(intervalle)
            mutter = population[mutter_index]
            vater = population[vater_index]
            #print(mutter, vater)
            kinder = []
            kinder = rekombiniere(mutter, vater)
            kinder = mutiere(kinder, individuum_size)
            #print("Kind", kinder)
            population_neu.append(kinder[0])
            population_neu.append(kinder[1])
        population = population_neu

        print(best, bestFit)
run(50,10)
```

```
{0: 50, 1: 50, 2: 40, 3: 40, 4: 40, 5: 40, 6: 30, 7: 40, 8: 50, 9: 60}
[3, 1, 1, 4] 60
{0: 40, 1: 40, 2: 60, 3: 30, 4: 50, 5: 40, 6: 60, 7: 40, 8: 50, 9: 30}
[3, 1, 1, 4] 60
{0: 40, 1: 60, 2: 50, 3: 60, 4: 50, 5: 40, 6: 40, 7: 60, 8: 50, 9: 40}
[3, 1, 1, 4] 60
{0: 60, 1: 40, 2: 40, 3: 40, 4: 40, 5: 40, 6: 40, 7: 60, 8: 70, 9: 40}
[2, 3, 1, 4] 70
{0: 40, 1: 60, 2: 40, 3: 50, 4: 60, 5: 40, 6: 40, 7: 40, 8: 40, 9: 40}
[2, 3, 1, 4] 70
{0: 40, 1: 60, 2: 50, 3: 40, 4: 50, 5: 40, 6: 50, 7: 40, 8: 60, 9: 40}
[2, 3, 1, 4] 70
{0: 40, 1: 40, 2: 50, 3: 40, 4: 50, 5: 50, 6: 40, 7: 50, 8: 60, 9: 50}
```

```
[2, 3, 1, 4] 70
{0: 60, 1: 50, 2: 50, 3: 50, 4: 50, 5: 50, 6: 40, 7: 50, 8: 40, 9: 40}
[2, 3, 1, 4] 70
{0: 40, 1: 50, 2: 40, 3: 50, 4: 60, 5: 50, 6: 50, 7: 50, 8: 40, 9: 50}
[2, 3, 1, 4] 70
{0: 50, 1: 40, 2: 60, 3: 60, 4: 50, 5: 50, 6: 70, 7: 50, 8: 50, 9: 50}
[2, 3, 1, 4] 70
{0: 50, 1: 50, 2: 50, 3: 40, 4: 50, 5: 70, 6: 50, 7: 50, 8: 50, 9: 60}
[2, 3, 1, 4] 70
{0: 40, 1: 50, 2: 50, 3: 60, 4: 50, 5: 50, 6: 50, 7: 50, 8: 50, 9: 70}
[2, 3, 1, 4] 70
{0: 50, 1: 50, 2: 70, 3: 60, 4: 50, 5: 50, 6: 40, 7: 50, 8: 50, 9: 60}
[2, 3, 1, 4] 70
{0: 40, 1: 70, 2: 40, 3: 50, 4: 40, 5: 50, 6: 50, 7: 80, 8: 50, 9: 50}
[3, 1, 4, 2] 80
{0: 70, 1: 40, 2: 40, 3: 50, 4: 50, 5: 50, 6: 50, 7: 50, 8: 80, 9: 50}
[3, 1, 4, 2] 80
{0: 50, 1: 50, 2: 60, 3: 50, 4: 50, 5: 50, 6: 40, 7: 50, 8: 50, 9: 80}
[3, 1, 4, 2] 80
{0: 50, 1: 50, 2: 50, 3: 70, 4: 50, 5: 50, 6: 80, 7: 40, 8: 60, 9: 50}
[3, 1, 4, 2] 80
{0: 40, 1: 70, 2: 40, 3: 50, 4: 50, 5: 50, 6: 70, 7: 50, 8: 50, 9: 50}
[3, 1, 4, 2] 80
{0: 50, 1: 50, 2: 70, 3: 50, 4: 70, 5: 50, 6: 50, 7: 70, 8: 50, 9: 50}
[3, 1, 4, 2] 80
{0: 70, 1: 50, 2: 70, 3: 50, 4: 70, 5: 70, 6: 50, 7: 50, 8: 50, 9: 70}
[3, 1, 4, 2] 80
{0: 60, 1: 70, 2: 70, 3: 50, 4: 50, 5: 70, 6: 60, 7: 70, 8: 50, 9: 50}
[3, 1, 4, 2] 80
{0: 50, 1: 50, 2: 80, 3: 70, 4: 70, 5: 50, 6: 60, 7: 70, 8: 70, 9: 50}
[3, 1, 4, 2] 80
{0: 80, 1: 60, 2: 70, 3: 50, 4: 50, 5: 70, 6: 70, 7: 50, 8: 50, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 50, 3: 70, 4: 60, 5: 80, 6: 60, 7: 50, 8: 70, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 60, 2: 70, 3: 70, 4: 80, 5: 70, 6: 70, 7: 60, 8: 50, 9: 50}
[3, 1, 4, 2] 80
{0: 70, 1: 80, 2: 80, 3: 70, 4: 70, 5: 70, 6: 70, 7: 70, 8: 60, 9: 60}
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 70, 3: 70, 4: 70, 5: 70, 6: 70, 7: 60, 8: 70, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 70, 3: 70, 4: 70, 5: 70, 6: 70, 7: 70, 8: 70, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 70, 3: 70, 4: 70, 5: 70, 6: 70, 7: 70, 8: 70, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 70, 3: 70, 4: 70, 5: 70, 6: 70, 7: 70, 8: 70, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 70, 3: 70, 4: 50, 5: 70, 6: 70, 7: 70, 8: 70, 9: 70}
```

```
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 70, 3: 70, 4: 70, 5: 70, 6: 70, 7: 70, 8: 70, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 70, 3: 70, 4: 70, 5: 70, 6: 70, 7: 70, 8: 70, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 80, 3: 70, 4: 70, 5: 70, 6: 70, 7: 70, 8: 70, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 70, 2: 50, 3: 70, 4: 70, 5: 70, 6: 50, 7: 70, 8: 80, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 80, 2: 70, 3: 70, 4: 70, 5: 70, 6: 70, 7: 70, 8: 50, 9: 80}
[3, 1, 4, 2] 80
{0: 80, 1: 50, 2: 50, 3: 70, 4: 70, 5: 70, 6: 70, 7: 70, 8: 50, 9: 70}
[3, 1, 4, 2] 80
{0: 70, 1: 50, 2: 50, 3: 70, 4: 50, 5: 80, 6: 70, 7: 70, 8: 70, 9: 50}
[3, 1, 4, 2] 80
{0: 80, 1: 70, 2: 50, 3: 70, 4: 70, 5: 50, 6: 50, 7: 70, 8: 50, 9: 80}
[3, 1, 4, 2] 80
{0: 50, 1: 70, 2: 50, 3: 70, 4: 50, 5: 50, 6: 50, 7: 70, 8: 50, 9: 70}
[3, 1, 4, 2] 80
{0: 50, 1: 70, 2: 50, 3: 50, 4: 50, 5: 50, 6: 50, 7: 60, 8: 50, 9: 50}
[3, 1, 4, 2] 80
{0: 50, 1: 50, 2: 50, 3: 50, 4: 70, 5: 50, 6: 50, 7: 70, 8: 60, 9: 50}
[3, 1, 4, 2] 80
{0: 50, 1: 50, 2: 70, 3: 50, 4: 80, 5: 50, 6: 80, 7: 50, 8: 50, 9: 70}
[3, 1, 4, 2] 80
{0: 80, 1: 80, 2: 50, 3: 70, 4: 60, 5: 70, 6: 80, 7: 80, 8: 80, 9: 80}
[3, 1, 4, 2] 80
{0: 80, 1: 50, 2: 80, 3: 80, 4: 60, 5: 80, 6: 80, 7: 80, 8: 80, 9: 50}
[3, 1, 4, 2] 80
{0: 80, 1: 80, 2: 70, 3: 80, 4: 80, 5: 80, 6: 50, 7: 80, 8: 80, 9: 80}
[3, 1, 4, 2] 80
{0: 50, 1: 80, 2: 80, 3: 80, 4: 70, 5: 80, 6: 80, 7: 70, 8: 80, 9: 80}
[3, 1, 4, 2] 80
{0: 80, 1: 70, 2: 50, 3: 70, 4: 80, 5: 80, 6: 80, 7: 80, 8: 80, 9: 80}
[3, 1, 4, 2] 80
{0: 70, 1: 80, 2: 80, 3: 80, 4: 80, 5: 80, 6: 80, 7: 80, 8: 80, 9: 80}
[3, 1, 4, 2] 80
{0: 80, 1: 80, 2: 80, 3: 80, 4: 80, 5: 80, 6: 80, 7: 80, 8: 80, 9: 70}
[3, 1, 4, 2] 80
```