p024_blatt05-Aufgabe 1

June 6, 2024

Blatt 05 - Praktische Optimierung - Adrian Lentz, Robert

Lösungen und Erklärungen für Blatt 05.

Adrian Lentz - Matrikelnummer: 258882

Robert Schönewald - Matrikelnummer: 188252

Aufgabe 5.1

[1]:
```
import numpy as np
import timeit
import matplotlib.pyplot as plt
import scipy
import statsmodels.api as sm
import statsmodels.distributions.empirical_distribution as edf
from scipy.stats import multivariate_normal
```

[2]:
```
#Funktion definieren
def f_a(x):
    return 1.5*x[0]**2 + x[1]**2 + 21*np.sin(x[0])*np.cos(x[1]) + 0.
↪5*((abs(x[0]))**2 + (abs(x[1]))**2)
```

[3]:
```
#Implementiere Threshold Accepting
def TA(f,d,X):
    Z=multivariate_normal(np.zeros(d), np.identity(d))
    T0=1
    for x in range(20):
        Y=X+Z.rvs()
        fY=f(Y)
        fX=f(X)
        if fY<fX+T0:
            X=Y
        T0=T0*0.5
    return X
```

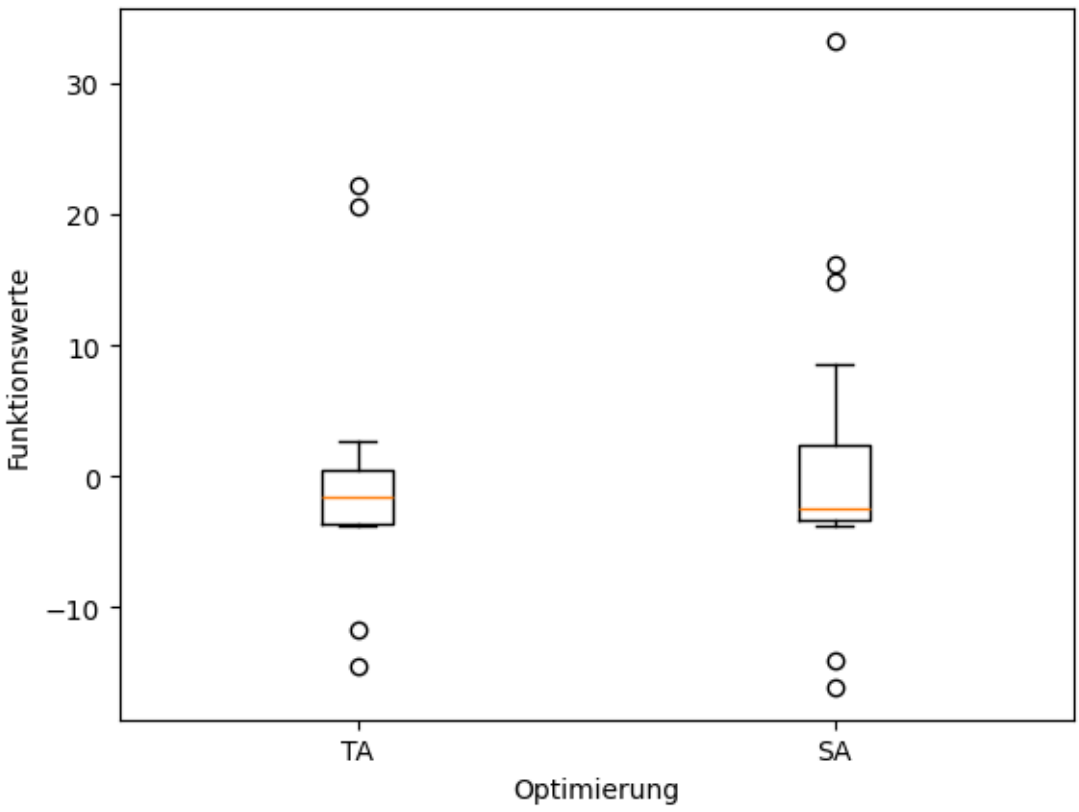
[4]:
```
#Implementiere Simulated Annealing
def SA(f,d,X):
    Z=multivariate_normal(np.zeros(d), np.identity(d))
    T0=1
    for x in range(20):
        Y=X+Z.rvs()
        fY=f(Y)
        fX=f(X)
        if fY<fX:
            X=Y
        if fY>fX:
            if np.random.uniform(0,1) < np.exp((fX-fY)/T0):
                X=Y
        T0=T0*0.5
    return X
```

[5]:
```
TAtraj=[]
TAval=[]
for x in range(20):
    result=TA(f_a,2,[5,5])
    TAtraj.append(result)     #Fülle Liste mit Ergebnissen
    TAval.append(f_a(result)) #Fülle Liste mit allen Funktionswerten
#np.median(TAtraj)
#np.mean(TAtraj)
#np.std(TAtraj)
```

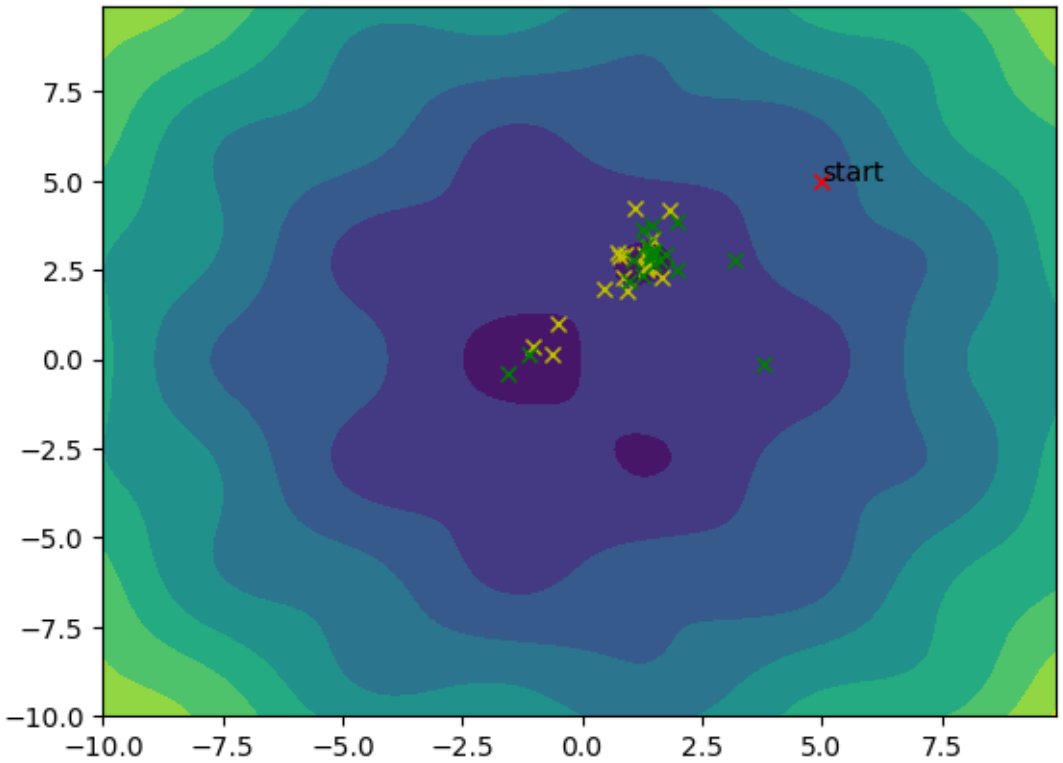
[6]:
```
SAtraj=[]
SAval=[]
for x in range(20):
    result=SA(f_a,2,[5,5])
    SAtraj.append(result)     #Fülle Liste mit Ergebnissen
    SAval.append(f_a(result)) #Fülle Liste mit allen Funktionswerten
#np.median(SAtraj)
#np.mean(SAtraj)
#np.std(SAtraj)
```

[7]:
```
fig, ax = plt.subplots()
ax.boxplot([TAval,SAval])
ax.set_xticklabels(['TA','SA'])
ax.set(xlabel='Optimierung', ylabel='Funktionswerte')
```

[7]:
```
[Text(0.5, 0, 'Optimierung'), Text(0, 0.5, 'Funktionswerte')]
```



[8]:
```
x, y = np.mgrid[-10:10:.1, -10:10:.1]
pos = [x, y]
fig2 = plt.figure()
ax2 = fig2.add_subplot(111)
ax2.contourf(x, y, f_a(pos))
ax2.plot(5,5,"rx")
ax2.text(5,5,"start")
for x in TAtraj:
    ax2.plot(x[0],x[1],"yx")              #TA Werte werden in Gelb markiert
for x in SAtraj:
    ax2.plot(x[0],x[1],"gx")              #SA Werte werden in Grün markiert
```



Wie man in dem obigen Bild und dem Boxplot sieht, sind SA und TA ungefähr gleich variant. Beide Varianten finden meistens den lokalen Tiefpunkt in der Nähe des Startpunkts. Sowohl SA und TA produzieren Ausreißer, die dem globalen Optimum nahe kommen, Jedoch hat SA nach oben die stärkeren Ausreißer. Insgesamt sind beide Varianten sehr ähnlich, und beide Möglichkeiten, lokale Optima zu überwinden scheinen zu funktionieren. Um wirklich große Unterschiede feststellen zu können müsste die Stichprobengröße wahrscheinlich deutlich höher sein. Dies ist bei häufigerer Ausführung des Codes aufgefallen, da teilweise SA bessere Ergebnisse geliefert hatte und teilweise TA, jedoch ohne ein klares Schema feststellen zu können.

Aufgabe 5.2

[1]:
```
import numpy as np
import random
import statistics
```

[2]:
```
class Mutationen:
    def __init__(self):
        pass

    def two_swap_mutation(self, individual, K):
        size = len(individual)
        if size < 2:
```

```
            return individual # Keine Mutation möglich, wenn die Länge weniger␣
↪als 2 ist

        for _ in range(K):  # Wähle zwei verschiedene Positionen zufällig aus
            pos1, pos2 = random.sample(range(size), 2)
            individual[pos1], individual[pos2] = individual[pos2],␣
↪individual[pos1] # Tausch der Werte an den beiden Positionen

        return individual

    def one_translocation_mutation(self, individual, K):
        size = len(individual)
        for _ in range(K):  # Wähle eine zufällige Position für das Gen, das␣
↪verschoben werden soll
            pos1 = random.randint(0, size - 1)
            gene = individual.pop(pos1)
            pos2 = random.randint(0, size - 1) # Wähle eine neue zufällige␣
↪Position, um das Gen wieder einzufügen
            individual.insert(pos2, gene)
        return individual

    def mutate(self, individual, method, K):
        if method == '2-Swap':
            return self.two_swap_mutation(individual, K)
        elif method == '1-Translocation':
            return self.one_translocation_mutation(individual, K)
        else:
            raise ValueError("Invalid mutation method. Choose '2-Swap' or␣
↪'1-Translocation'.")
```

[3]:
```
class Rekombination:
    def __init__(self):
        pass

    def order_based_crossover(self, parent1, parent2):
        size = len(parent1)
        start, end = sorted(random.sample(range(size), 2))  # Wähle zwei␣
↪zufällige Schnittpunkte
        #Initailisieren der Kinder
        child1 = [-1] * size
        child1[start:end] = parent1[start:end]
        #Aktuelle Indexposition
        child1_pointer = end
        parent2_pointer = end
        #OBX für 1.Kind
        while -1 in child1:
```

```
            if parent2[parent2_pointer % size] not in child1: #Index Bsp [5 %␣
↪9]: Modulo Operator, hier 5<9 --> parent2[5] mit Kind verglichen
                child1[child1_pointer % size] = parent2[parent2_pointer % size]␣
↪#Indexpositon des Kindes wird mit entsprechnden Eintrag des Eltern aufgefüllt
                child1_pointer += 1
            parent2_pointer += 1

        return child1

    def partially_mapped_crossover(self, parent1, parent2):
        size = len(parent1)
        start, end = sorted(random.sample(range(size), 2))
        child1 = [-1] * size
        child1[start:end] = parent1[start:end]

        mapping = {}
        for i in range(start, end):
            mapping[parent1[i]] = parent2[i]
            mapping[parent2[i]] = parent1[i]

        def fill_child(child, parent, start, end, mapping):
            for i in range(size):
                if i < start or i >= end:
                    gene = parent[i]
                    while gene in child[start:end]:
                        gene = mapping[gene]
                    child[i] = gene

        fill_child(child1, parent2, start, end, mapping)
        return child1

    def keine_rekombination(self, parent1, parent2): #Keine Rekombination -->␣
↪Kinder sind gleich zu Eltern
        return parent1

    def recombine(self, parent1, parent2, method):
        if method == 'OBX':
            return self.order_based_crossover(parent1, parent2)
        elif method == 'PMX':
            return self.partially_mapped_crossover(parent1, parent2)
        elif method == 'NIX':
            return self.keine_rekombination(parent1, parent2)
        else:
            raise ValueError("Invalid recombination method. Choose 'OBX',␣
↪'PMX', or 'NIX'.")
```

[4]:
```python
class Zielfunktionen:
    def __init__(self):
        pass

    def ziel_1(self, kandidat): #Maximum - Minimum
        sums = self.evaluate(kandidat)
        return max(sums) - min(sums)

    def ziel_2(self, kandidat): #Summe der absoluten Differenz von allen Paaren
        sums = self.evaluate(kandidat)
        total_difference = 0
        for i in range(len(sums)):  #Sums enthält alle möglichen s1 bis s8␣
↪Elemente
            for j in range(i + 1, len(sums)):  #Durchläuft nachfolgendes␣
↪Element nach i
                total_difference += abs(sums[i] - sums[j])
        return total_difference

    def ziel_3(self, kandidat):  #Summe der absoluten Differenz von s1 und␣
↪allen andern s2 bis s8
        sums = self.evaluate(kandidat)
        s1 = sums[0]
        total_difference = 0
        for i in range(1, len(sums)):
            total_difference += abs(s1 - sums[i])
        return total_difference

    def evaluate(self, kandidat):
        s1 = sum(kandidat[0:3])
        s2 = sum(kandidat[3:6])
        s3 = sum(kandidat[6:9])
        s4 = sum(kandidat[0:9:3])
        s5 = sum(kandidat[1:9:3])
        s6 = sum(kandidat[2:9:3])
        s7 = kandidat[0] + kandidat[4] + kandidat[8]
        s8 = kandidat[2] + kandidat[4] + kandidat[6]
        return [s1, s2, s3, s4, s5, s6, s7, s8]
```

[31]:
```python
class EvolutionaryAlgorithm:
    def __init__(self, mu, lamb, mutation_k, mutation_method,␣
↪recombination_method, zielfunktion,max_evaluations=10000):  #Weil Probleme,␣
↪versuch mit maximaler evaluations-Anzahl
        self.mu = mu #Eltern
        self.lamb = lamb #Kinder
        self.mutation_k = mutation_k #Für Mutationen k=2 oder k=4
        self.mutation_method = mutation_method #Mutationen
        self.mutationen = Mutationen()
```

```
        self.rekombination_method = recombination_method #Rekombinationen
        self.rekombination = Rekombination()
        self.zielfunktion = zielfunktion #Zielfunktion
        self.max_evaluations = max_evaluations  # Max Evaluations festgelegt
        self.population = [random.sample(range(1, 10), 9) for _ in range(mu)]␣
↪#Erstellung von (mu)-Individuen (zufällige Zahl von 1 bis 9) einer␣
↪zufälliger Population


    def run(self):
        auswertungen = 0 #Anzahl der Auswertungen der Zielfunktion
        while auswertungen < self.max_evaluations:
            offspring = []
            for _ in range(self.lamb): #Schleife für Kinder
                parent1, parent2 = random.sample(self.population, 2)
                child = self.rekombination.recombine(parent1, parent2, self.
↪rekombination_method) #Rekombination
                child = self.mutationen.mutate(child, self.mutation_method,␣
↪self.mutation_k) #Mutation
                offspring.append(child)
                auswertungen += 1
            self.population += offspring
            self.population.sort(key=self.zielfunktion) #Sortierung nach␣
↪Zielfunktion
            self.population = self.population[:self.mu]   #Auswahl bester␣
↪Individuen

            if self.is_magic_square(self.population[0]):
                return auswertungen

    def is_magic_square(self, kandidat): #Überprüfen ob magischer Würfel passt
        return self.zielfunktion(kandidat) == 0 #Zielfunktion ist gleich 0 für␣
↪magischen Würfel

if __name__ == "__main__":
    mu_values = [5,10,20] #5,10,20
    lambda_values = [1,5,10] #1,5,10
    mutation_operators = ['2-Swap', '1-Translocation'] #'2-Swap',␣
↪'1-Translocation'
    mutation_k_werte = [2,4] #2,4
    rekombination_operators = ['OBX', 'PMX', 'NIX']#'OBX', 'PMX', 'NIX'
    zielfunktionen = Zielfunktionen()
    zielfunktion_operators = [zielfunktionen.ziel_1, zielfunktionen.ziel_2,␣
↪zielfunktionen.ziel_3]

    max_evaluations = 10000
```

```
    results = []

    for mu in mu_values: #Durch Eltern
        for lamb in lambda_values: #Durch Kinder
            for mutation_operator in mutation_operators: #Durch Mutationen
                for mutation_k in mutation_k_werte: #Mutationswert k=2 oder k=4
                    for rekombination_operator in rekombination_operators:␣
↪#Durch Rekombinationen
                        for zielfunktion_operator in zielfunktion_operators:␣
↪#Durch Zielfunktionen
                            evals_for_combination = []
                            for _ in range(10): #100-Wiederholungen
                                random.seed()
                                ea = EvolutionaryAlgorithm(mu, lamb,␣
↪mutation_k, mutation_operator, rekombination_operator,␣
↪zielfunktion_operator, max_evaluations)
                                evaluations = ea.run()
                                if evaluations is not None:
                                    evals_for_combination.append(evaluations)
                            if evals_for_combination:
                                median_evaluations = statistics.
↪median(evals_for_combination)
                                results.append((median_evaluations, mu, lamb,␣
↪mutation_operator, mutation_k, rekombination_operator, zielfunktion_operator.
↪__name__))
                                print(f"Median Evaluations:␣
↪{median_evaluations}, Mu: {mu}, Lambda: {lamb}, Mutation:␣
↪{mutation_operator}, Mutation K: {mutation_k}, Rekombination:␣
↪{rekombination_operator}, Zielfunktion: {zielfunktion_operator.__name__}")

    # Sortiere die Ergebnisse nach der Anzahl der Evaluierungen (Median) und␣
↪gib die besten und schlechtesten Parameterkombinationen aus
    results.sort()

    print("\nDie 5 besten Parameterkombinationen:")
    for result in results[:5]:
        print(result)

    print("\nDie 5 schlechtesten Parameterkombinationen:")
    for result in results[-5:]:
        print(result)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[31], line 57
     55 random.seed()
```

```
     56 ea = EvolutionaryAlgorithm(mu, lamb, mutation_k, mutation_operator,␣
 ↪rekombination_operator, zielfunktion_operator, max_evaluations)
---> 57 evaluations = ea.run()
     58 if evaluations is not None:
     59     evals_for_combination.append(evaluations)

Cell In[31], line 21, in EvolutionaryAlgorithm.run(self)
     19 for _ in range(self.lamb): #Schleife für Kinder
     20     parent1, parent2 = random.sample(self.population, 2)
---> 21     child =␣
 ↪self.rekombination.recombine(parent1, parent2, self.rekombination_method)␣
 ↪#Rekombination
     22     child = self.mutationen.mutate(child, self.mutation_method, self.
 ↪mutation_k) #Mutation
     23     offspring.append(child)

Cell In[3], line 52, in Rekombination.recombine(self, parent1, parent2, method)
     50     return self.order_based_crossover(parent1, parent2)
     51 elif method == 'PMX':
---> 52     return self.partially_mapped_crossover(parent1, parent2)
     53 elif method == 'NIX':
     54     return self.keine_rekombination(parent1, parent2)

Cell In[3], line 42, in Rekombination.partially_mapped_crossover(self, parent1,
 ↪parent2)
     39                 gene = mapping[gene]
     40             child[i] = gene
---> 42 fill_child(child1, parent2, start, end, mapping)
     43 return child1

Cell In[3], line 38, in Rekombination.partially_mapped_crossover.<locals>.
 ↪fill_child(child, parent, start, end, mapping)
     36 if i < start or i >= end:
     37     gene = parent[i]
---> 38     while gene in child[start:end]:
     39         gene = mapping[gene]
     40     child[i] = gene

KeyboardInterrupt:
```

Leider kommt der EA nicht über eine bestimmte Kombinationen hinaus von Parametern, ich weiß leider nicht warum.

Versuch deswegen eine maximale evaluation festzulegen (10000) und die Anzahl der Wiederholungen zu verringern hat auch nicht geholfen.

Selbst nach sehr langer Rechenzeit , kommt der EA leider nicht aus einer einzigen Kombination von Paramteren raus, weswegen hier ein paar manulle Paramter Kombinationen festgelegt wur-

den, um einen Vergleich zu ermöglichen. Zusätzlich erscheint die Rekombination “PMX” nicht zu funktionieren, leider.

[ ]:
```
'Bsp-Output:'

'''
Evaluations: 784, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2,␣
 ↪Rekombination: OBX, Zielfunktion: ziel_1
Evaluations: 1364, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2,␣
 ↪Rekombination: OBX, Zielfunktion: ziel_1
Evaluations: 1903, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2,␣
 ↪Rekombination: OBX, Zielfunktion: ziel_1
Evaluations: 1642, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2,␣
 ↪Rekombination: OBX, Zielfunktion: ziel_1
Evaluations: 4661, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2,␣
 ↪Rekombination: OBX, Zielfunktion: ziel_1
Evaluations: 2212, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2,␣
 ↪Rekombination: OBX, Zielfunktion: ziel_1
Evaluations: 3474, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2,␣
 ↪Rekombination: OBX, Zielfunktion: ziel_1
....... etc.
'''
```

Alle Parameter-Kombinationen für μ=5 (manuell mit EA berechnet):

Median Evaluations: 1847, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 1209.0, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 1467.0, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 5351.0, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2, Rekombination: NIX, Zielfunktion: ziel_1

Median Evaluations: 4139, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2, Rekombination: NIX, Zielfunktion: ziel_2

Median Evaluations: 1945, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2, Rekombination: NIX, Zielfunktion: ziel_3

Median Evaluations: 5996.0, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 7065, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 6984.0, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 6890, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_1

Median Evaluations: 8724.0, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_2

Median Evaluations: 6543.5, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_3

Median Evaluations: 1987.0, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 2873.5, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 1222, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 4678.0, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 3879, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 5053, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 3304.0, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 2, Rekombination: NIX, Zielfunktion: ziel_2

Median Evaluations: 2621, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 2, Rekombination: NIX, Zielfunktion: ziel_3

Median Evaluations: 1910.5, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_1

Median Evaluations: 770.0, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_2

Median Evaluations: 8444.0, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_3

Median Evaluations: 1920, Mu: 5, Lambda: 5, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 1035, Mu: 5, Lambda: 5, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 1460.0, Mu: 5, Lambda: 5, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 3945, Mu: 5, Lambda: 5, Mutation: 2-Swap, Mutation K: 2, Rekombination: NIX, Zielfunktion: ziel_3

Median Evaluations: 3002.5, Mu: 5, Lambda: 5, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 2365, Mu: 5, Lambda: 5, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 2815.0, Mu: 5, Lambda: 5, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 2510, Mu: 5, Lambda: 5, Mutation: 2-Swap, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_1

Median Evaluations: 4055, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 1865.0, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 2445, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 920, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 2, Rekombination: NIX, Zielfunktion: ziel_2

Median Evaluations: 6290.0, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 7725.0, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 3435, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 4610.0, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_1

Median Evaluations: 2180, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_2

Median Evaluations: 920, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 1130, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 1990, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 6520, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 2, Rekombination: NIX, Zielfunktion: ziel_2

Median Evaluations: 5460.0, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 4480.0, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 6340.0, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_3

Median Evaluations: 990, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_3

Median Evaluations: 1170, Mu: 5, Lambda: 10, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 1340, Mu: 5, Lambda: 10, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 2580, Mu: 5, Lambda: 10, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_3

-

Median Evaluations: 5750, Mu: 5, Lambda: 10, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1

Median Evaluations: 3550.0, Mu: 5, Lambda: 10, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_2

Median Evaluations: 7415.0, Mu: 5, Lambda: 10, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_3

Für Ziel1: Maximum - Minium: (nur für mu=5)

Beste Parameter-Kombination:

1. Median Evaluations: 920, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1
2. Median Evaluations: 1170, Mu: 5, Lambda: 10, Mutation: 1-Translocation, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1
3. Median Evaluations: 1847, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1
4. Median Evaluations: 1910.5, Mu: 5, Lambda: 1, Mutation: 1-Translocation, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_1
5. Median Evaluations: 1920, Mu: 5, Lambda: 5, Mutation: 2-Swap, Mutation K: 2, Rekombination: OBX, Zielfunktion: ziel_1

Schlechste Parameter-Kombination:

6. Median Evaluations: 6890, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 4, Rekombination: NIX, Zielfunktion: ziel_1
7. Median Evaluations: 6290.0, Mu: 5, Lambda: 5, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1
8. Median Evaluations: 5996.0, Mu: 5, Lambda: 1, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1
9. Median Evaluations: 5750, Mu: 5, Lambda: 10, Mutation: 1-Translocation, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1
10. Median Evaluations: 5460.0, Mu: 5, Lambda: 10, Mutation: 2-Swap, Mutation K: 4, Rekombination: OBX, Zielfunktion: ziel_1

[ ]: