Лабораторная №6

Вариант №3 (1-6)

Выполнил студент группы P3212 Балин Артем

Точность разницы между средними значениями в поколении: 0.1

In [ ]:
```
from solution import solution

matrix = [
    [0, 4, 5, 3, 8],
    [4, 0, 7, 6, 8],
    [5, 7, 0, 7, 9],
    [3, 6, 7, 0, 9],
    [8, 8, 9, 9, 0],
]
probabilty = 0.01
n = 4  # number of population
e = 0.1 # accuracy
genetic_search = solution(matrix, probabilty)
population = [genetic_search.generate_random_gen_code() for i in range(n)]
func = [genetic_search.aim_function(i) for i in population]
print(" i |  код  | f | вероятность ")
print("---------------------------")

for i, (code, value) in enumerate(zip(population, func), start=1):
    code_str = "".join(map(str, code))
    print(f" {i} | {code_str} | {value}| {value/sum(func):^2f}")
print()
average = sum(func) / n
average_last = 10**9
generation = 2
while average_last - average > e:
    average_last = average
    new_population = []
    parents_pairs = genetic_search.parents()  # random pairs of parents
    i1, i2 = genetic_search.find_different_random_places()
    new_population.append(
        genetic_search.crossover(
            population[parents_pairs[0]], population[parents_pairs[1]], i1, i2
        )
    )
    new_population.append(
        genetic_search.crossover(
            population[parents_pairs[1]], population[parents_pairs[0]], i1, i2
        )
    )
    new_population.append(
        genetic_search.crossover(
            population[parents_pairs[2]], population[parents_pairs[3]], i1, i2
        )
    )
```

```
    new_population.append(
        genetic_search.crossover(
            population[parents_pairs[3]], population[parents_pairs[2]], i1, i2
        )
    )
    new_func = [genetic_search.aim_function(i) for i in new_population]
    average = sum(new_func) / n
    print("Потомки поколения №", generation - 1)
    print(" i |  код  | целевая функция |")
    print("---------------------------")
    for i, (code, value) in enumerate(zip(population, func), start=1):
        code_str = "".join(map(str, code))
        print(f" {i} | {code_str} | {value}")
    combined_population = population + new_population
    combined_func = [
        genetic_search.aim_function(individual) for individual in combined_popul
    ]
    sorted_population = sorted(
        zip(combined_population, combined_func), key=lambda x: x[1]
    )
    population = [individual for individual, value in sorted_population[:n]]
    func = [value for individual, value in sorted_population[:n]]
    average = sum(func) / n
    print(f"Поколение № {generation}")
    print(" i |  код  | f | вероятность ")
    print("---------------------------")
    for i, (code, value) in enumerate(zip(population, func), start=1):
        code_str = "".join(map(str, code))
        print(f" {i} | {code_str} | {value}| {value/sum(func):^2f}")
    print()
    generation += 1
# print minimum
min_value = min(func)
print("Минимальное значение целевой функции: ", min_value)
```

```
 i |  код  | f | вероятность
---------------------------
 1 | 53124 | 33| 0.255814
 2 | 42135 | 33| 0.255814
 3 | 12534 | 31| 0.240310
 4 | 32541 | 32| 0.248062

Parent 1: 3|25|41
Parent 2: 1|25|34
Child 1: 3|25|41

Parent 1: 1|25|34
Parent 2: 3|25|41
Child 1: 1|25|34

Parent 1: 4|21|35
Parent 2: 5|31|24
Child 1: 4|31|25

Parent 1: 5|31|24
Parent 2: 4|21|35
Child 1: 5|21|34

Потомки поколения № 1
 i |  код  | целевая функция |
---------------------------
 1 | 53124 | 33
 2 | 42135 | 33
 3 | 12534 | 31
 4 | 32541 | 32
Поколение № 2
 i |  код  | f | вероятность
---------------------------
 1 | 12534 | 31| 0.246032
 2 | 12534 | 31| 0.246032
 3 | 32541 | 32| 0.253968
 4 | 32541 | 32| 0.253968

Parent 1: 3|254|1
Parent 2: 1|253|4
Child 1: 3|254|1

Parent 1: 1|253|4
Parent 2: 3|254|1
Child 1: 1|254|3

Parent 1: 1|253|4
Parent 2: 3|254|1
Child 1: 1|254|3

Parent 1: 3|254|1
Parent 2: 1|253|4
Child 1: 3|254|1

Потомки поколения № 2
 i |  код  | целевая функция |
---------------------------
 1 | 12534 | 31
 2 | 12534 | 31
 3 | 32541 | 32
```

```
 4 | 32541 | 32
Поколение № 3
 i |  код  | f | вероятность
--------------------------
 1 | 12534 | 31| 0.246032
 2 | 12534 | 31| 0.246032
 3 | 32541 | 32| 0.253968
 4 | 32541 | 32| 0.253968

Минимальное значение целевой функции:  31
```

Код

```
    from random import randint, random, shuffle


    class solution:
        def __init__(self, matrix, p):
            self.matrix = matrix
            self.n = len(matrix)
            self.p = p  # probability of mutation

        def get_len(self, x, y):
            return self.matrix[int(x) - 1][int(y) - 1]

        def aim_function(self, s):
            result = 0
            for i in range(-1, self.n - 1):
                result += self.get_len(s[i], s[i + 1])
            return result

        def find_different_random_places(self):
            i1 = randint(1, self.n - 1)
            i2 = randint(1, self.n - 1)
            if i1 == i2:
                return self.find_different_random_places()
            return min(i1, i2), max(i1, i2)

        def is_mutaded(self):
            return random() <= self.p

        def mutation(self, s):
            i1, i2 = self.find_different_random_places()
            print(f"Mutation: {s} -> {s[:i1] + s[i2] + s[i1 + 1 :
    i2] + s[i1] + s[i2 + 1 :]}")
            return s[:i1] + s[i2] + s[i1 + 1 : i2] + s[i1] + s[i2 +
    1 :]

        def crossover(self, s1, s2, i1, i2):
            res1 = s1[:i1]
            for x in s2[i1:i2]:
                if x not in res1:
                    res1 += x
            for x in s1:
```

```
            if x not in res1:
                res1 += x
        if self.is_mutaded():
            res1 = self.mutation(res1)
        print(f"Parent 1: {s1[:i1]}|{s1[i1:i2]}|{s1[i2:]}")
        print(f"Parent 2: {s2[:i1]}|{s2[i1:i2]}|{s2[i2:]}")
        print(f"Child 1: {res1[:i1]}|{res1[i1:i2]}|
{res1[i2:]}\n")
        return res1

    def generate_random_gen_code(self):
        res = [str(i) for i in range(1, self.n + 1)]
        shuffle(res)
        return ''.join(res)
    @staticmethod
    def parents():
        n = 4  # hard-code
        res = [i for i in range(0, n)]
        shuffle(res)
        return res
```