《前沿智能算法及应用》
课 程 论 文

题目：遗传算法调试

学　　号	2212266	姓　　名	张恒硕
专业班级	智能科学与技术 2 班	手　　机	13940235405
出生日期	2004.2.11	提交日期	2024.9.17
成绩评定：　　分　　任课教师签名： 2024 年　月　日			

遗传算法调试

摘要：遗传算法（Genetic Algorithms，GA）是模拟生物进化过程的一种算法，体现了交配、变异等遗传学行为，以“自然选择”原理寻找问题的全局最优解。本文就其具体的代码展开分析，对其编码方式、交叉参数、变异参数对代码运行结果和运行效率的影响进行实验与分析。

关键词：遗传算法 编码方式 交叉参数 变异参数 运行效果

零、问题重述与初步分析：

问题如下：

$$f(x,y) = 21.5 + x\sin 4\pi x + y\sin 20\pi y$$

$$-3.0 \le x \le 12.1，4.1 \le y \le 5.8$$

求 max(f(x,y))。

下图给出了在问题区间，问题目标函数值的取值散点图，可以发现，该目标函数因为三角函数部分的存在，具有急剧的波动，不具有单调性。

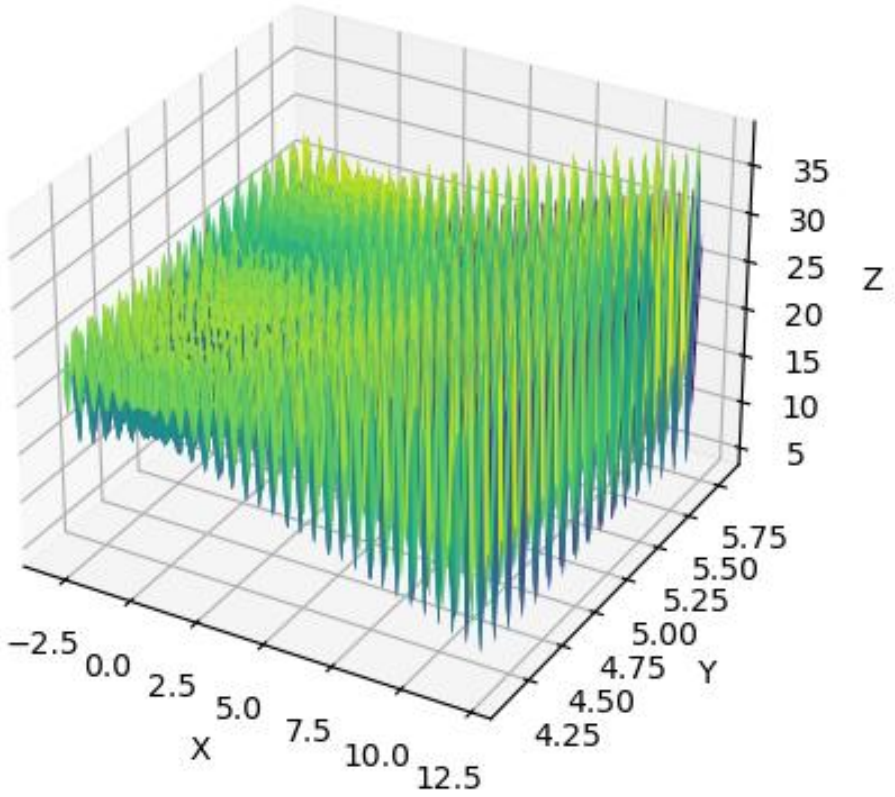

图 1 问题函数散点图

一、 代码展示与分析

以下先后给出了浮点数编码和二进制编码的遗传算法的代码，其都应用了单点交叉、单点变异、轮盘赌选择。

1.1 浮点数编码

```
import numpy as np

import pandas as pd

import time
```

```
n = 2               # 基因数目

population = 100  # 种群大小

generations = 2000# 迭代代数

X = [-3.0, 12.1]  # x 取值范围

Y = [4.1, 5.8]    # y 取值范围

bounds = [X, Y]   # 边界条件

repeat = 30        # 每组数据重复次数



# 种群初始化

def create_population(size, bounds):

    return np.random.rand(size, len(bounds)) * (np.array(bounds)[:, 1] -

np.array(bounds)[:, 0]) + np.array(bounds)[:,0]



# 适应度函数（函数一定非负，所以未进行保证非负的处理）

def get_fitness(pop):

    x = pop[:, 0]

    y = pop[:, 1]

    pred = 21.5 + x * np.sin(4 * np.pi * x) + y * np.sin(20 * np.pi * y)

    return pred



# 交叉：单点交叉

def crossover(pop, pc):
```

```
    child_pop = np.empty_like(pop, dtype=pop.dtype)
    for i in range(0, len(pop) - 1, 2):
        if np.random.rand() < pc:
            cross_point = np.random.randint(0, n)
            # 交换交叉点后的基因
            child_pop[i, :] = np.concatenate([pop[i, :cross_point], pop[i + 1, cross_point:]])
            child_pop[i + 1, :] = np.concatenate([pop[i + 1, :cross_point], pop[i,
cross_point:]])
        else:
            # 不进行交叉，直接复制
            child_pop[i, :] = pop[i, :]
            child_pop[i + 1, :] = pop[i + 1, :]
    return child_pop


# 变异：单点变异
def mutation(pop, pm, bounds):
    for i in range(len(pop)):
        for j in range(n):
            if np.random.rand() < pm:
                temp=np.random.uniform(bounds[j][0], bounds[j][1])
                pop[i, j] =temp
    return pop
```

```
# 选择：轮盘赌选择
def select(pop, fitness):
    idx = np.random.choice(np.arange(population), size=population, replace=True,
p=fitness / fitness.sum())
    return pop[idx]


start_time = time.time()
Pcs = np.linspace(0.1, 0.9, 9)
Pms = np.linspace(0.01, 0.09, 9)
fitness_list = np.zeros((9, 9))
for Pc in Pcs:
    for Pm in Pms:
        best_fitnesses = []
        for _ in range(repeat):
            pop = create_population(population, bounds)
            # best_idx = None
            for _ in range(generations):
                fitness = get_fitness(pop)
                # if best_idx is None or fitness[best_idx] < np.max(fitness):
                    # best_idx = np.argmax(fitness)
                pop = crossover(pop, Pc)
```

```
                pop = mutation(pop, Pm, bounds)
                pop = select(pop, fitness)
            # x = pop[best_idx, 0]
            # y = pop[best_idx, 1]
            # print("(%.5f,%.5f)" % (x, y))
            best_fitness = np.max(get_fitness(pop))
            best_fitnesses.append(best_fitness)
        mean_best_fitness = np.mean(best_fitnesses)
        fitness_list[int(Pc * 10) - 1, int(Pm * 100) - 1] = mean_best_fitness
        print("在 Pc=%.1f、Pm=%.2f 的条件下，多次遗传算法的均值为%.5f" % (Pc, Pm,
mean_best_fitness))
end_time = time.time()
print("浮点数编码代码运行时间：%.2f 秒" % (end_time - start_time))
df = pd.DataFrame(fitness_list, index=range(1, 10), columns=range(1, 10))
file_path = 'results.xlsx'
with pd.ExcelWriter(file_path, engine='openpyxl') as writer:
    df.to_excel(writer, sheet_name='float', index=True, header=True)
```

1.2 二进制编码

```
import numpy as np
import pandas as pd
import time
```

```python
n = 24              # 基因数目（二进制编码长度）
population = 100   # 种群大小
generations = 2000# 迭代代数
X = [-3.0, 12.1]   # x 取值范围
Y = [4.1, 5.8]     # y 取值范围
repeat = 30        # 每组数据重复次数


# 二进制与浮点数转化
def translateDNA(pop):
    x_pop = pop[:, 1::2]
    y_pop = pop[:, ::2]
    x = x_pop.dot(2 ** np.arange(n)[::-1]) / float(2 ** n - 1) * (X[1] - X[0]) + X[0]
    y = y_pop.dot(2 ** np.arange(n)[::-1]) / float(2 ** n - 1) * (Y[1] - Y[0]) + Y[0]
    return x, y


# 种群适应度
def get_fitness(pop):
    x, y = translateDNA(pop)
    pred = 21.5 + x * np.sin(4 * np.pi * x) + y * np.sin(20 * np.pi * y)
    return pred


# 交配、变异
```

```python
def crossover_and_mutation(pop):
    new_pop = []
    for father in pop:
        child = father.copy()
        # 交叉：单点交叉
        if np.random.rand() < Pc:
            mother = pop[np.random.randint(population)]
            cross_point = np.random.randint(low=0, high=2 * n)
            child[cross_point:] = mother[cross_point:]
        # 变异：单点变异
        for point in range(2*n):
            if np.random.rand() < Pm:
                child[point] = child[point] ^ 1
        new_pop.append(child)
    return new_pop


# 选择：轮盘赌
def select(pop, fitness):
    idx = np.random.choice(np.arange(population), size=population, replace=True,
p=fitness / fitness.sum())
    return pop[idx]
```

```
start_time = time.time()

Pcs = np.linspace(0.1, 0.9, 9)

Pms = np.linspace(0.01, 0.09, 9)

fitness_list = np.zeros((9, 9))

for Pc in Pcs:

    for Pm in Pms:

        best_fitnesses = []

        for _ in range(repeat):

            pop = np.random.randint(2, size=(population, n * 2))

            for _ in range(generations):

                x, y = translateDNA(pop)

                pop = np.array(crossover_and_mutation(pop))

                fitness = get_fitness(pop)

                pop = select(pop, fitness)

            best_fitness = np.max(get_fitness(pop))

            best_fitnesses.append(best_fitness)

            max_fitness_index = np.argmax(get_fitness(pop))

            # x, y = translateDNA(pop)

            # print("(%.5f,%.5f)" % (x, y))

        mean_best_fitness = np.mean(best_fitnesses)

        fitness_list[int(Pc * 10) - 1, int(Pm * 100) - 1] = mean_best_fitness

        print("在 Pc=%.1f、Pm=%.2f 的条件下，多次遗传算法的均值为%.5f" % (Pc, Pm,
```

```
mean_best_fitness))

end_time = time.time()

print("二进制编码代码运行时间：%.2f秒" % (end_time - start_time))

df = pd.DataFrame(fitness_list, index=range(1, 10), columns=range(1, 10))

file_path = 'results.xlsx'

with pd.ExcelWriter(file_path, engine='openpyxl') as writer:

    df.to_excel(writer, sheet_name='bin', index=True, header=True)
```

二、编码方式

针对两种编码方式的结果，可以简单地从以下两个方面进行比较。

- 运行时间：为了达到较高的精确度，针对两个变量的问题，二进制编码采用了 24 位基因进行编码，这带来了极大的运算量。

	浮点数编码	二进制编码
耗时（s）	1178.09	7869.89
以上时间是统一进行参数调配（81 组）、每组 30 次取均值的总耗时		

- 精确度：在综合各种参数条件的情况下，浮点数编码的精度明显高于二进制编码。

	浮点数编码	二进制编码
精确度	37.78290	36.88330
以上精确度是上述操作的 81 组均值数据的均值		

参考的数据为（所有结果中的最大值）：38.24754

三、交叉参数与变异参数

3.1 结果展示

以下给出两种编码方式各自的 81 种参数组合的结果情况，其中标蓝的是对应方式的最大值。

表 1 浮点数编码

	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.1	38.0678	38.13706	38.07381	38.02126	37.95089	38.04883	37.8967	38.10902	37.90274
0.2	37.94695	38.0347	38.08884	38.16102	38.09431	37.97408	38.01433	38.08647	37.80861
0.3	37.87805	37.98961	37.88685	38.16218	37.91215	37.9995	38.04786	37.80265	37.86702
0.4	37.65275	37.98558	37.88607	38.12574	37.89787	37.97976	37.91777	38.02588	37.82733
0.5	37.73943	37.63331	37.95310	37.98648	37.88236	38.02097	37.92012	37.89725	37.69008

0.6	37.36325	37.68684	37.88182	37.97271	37.81529	37.91783	37.62300	37.80670	37.65624
0.7	37.33243	37.56991	37.86155	37.75508	37.83433	37.76913	37.74802	37.47981	37.23127
0.8	37.21635	37.45596	37.57582	37.82805	37.35341	37.85097	37.71846	37.83366	37.50533
0.9	37.08981	37.07067	37.46489	37.54030	37.73556	37.41691	37.10365	37.46138	36.90314

表 2 二进制编码

	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.1	38.21218036	37.9768828	37.3831914	37.1000634	36.92141048	36.53909	36.26148919	35.82993593	36.31475188
0.2	37.98299494	37.89111693	37.40451657	37.21986061	36.8061303	36.61178	36.37042824	36.46458921	35.69454739

0.3	38.076 87615	37.669 76621	37.609 61137	37.189 74719	36.536 96584	36.7 2529	36.461 23373	36.345 39492	35.467 7378
0.4	38.247 54594	37.835 42334	37.540 6685	37.124 03791	37.263 95721	36.5 3824	36.492 71514	36.087 78798	36.002 07848
0.5	38.120 97109	37.715 65371	37.434 81058	37.310 57267	36.778 87899	36.3 2309	36.214 72765	36.220 52648	35.607 94205
0.6	38.168 85348	37.790 35216	37.522 43322	36.736 24698	36.853 44132	36.6 6930	36.085 98651	36.262 63535	35.814 79319
0.7	37.927 88396	37.836 40456	37.115 18022	37.096 97277	37.036 29869	36.1 8327	36.034 39115	36.332 23556	35.874 14504
0.8	37.944 83375	37.673 5861	37.444 38261	37.174 5134	36.414 74976	36.5 9759	36.151 73659	35.879 59393	35.637 4982

0.9	38.065 69269	37.733 03973	37.046 29771	37.110 96348	36.497 64832	36.4 7669	36.280 73576	36.255 66562	35.894 23045

3.2 交叉参数

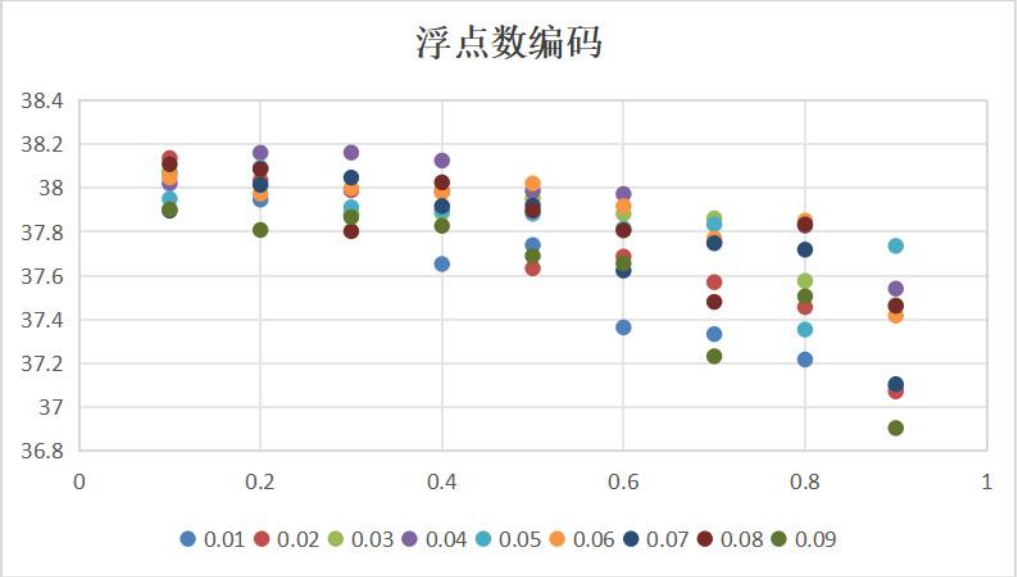

图 2 浮点数编码，交叉参数

由上图可知，对于浮点数编码而言，交叉概率较小的条件下，会得到较好的结果。这可能与自然界中大量繁殖带来的高交叉率的经验相违背。综合而言，选取 0.5~0.7 的交叉概率，能得到较优的结果，也能平衡多样性和优质性。

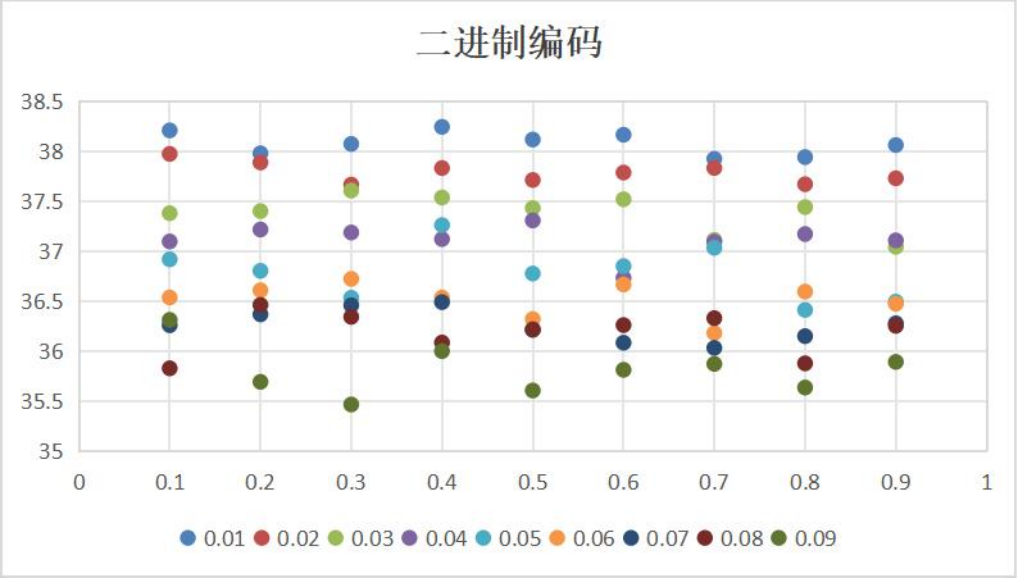

图 3 二进制编码，交叉参数

由上图可知，在容忍一定误差的情况下，使用二进制编码，不同的交叉参数带来的影响不大。这可能是由于较大的基因数量和较高的种群迭代次数，造成结果收敛性良好。

3.1 变异参数

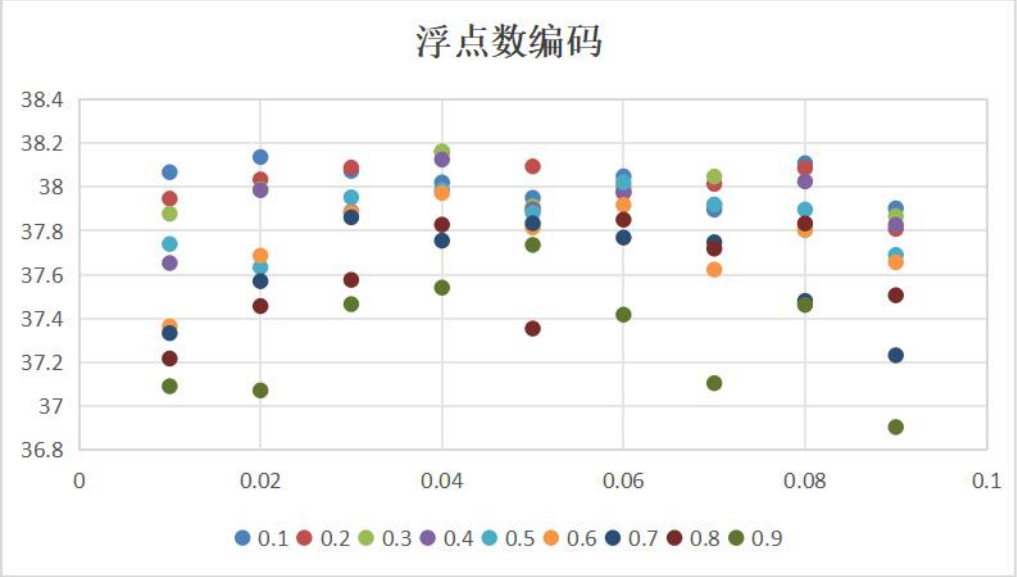

图 4 浮点数编码，变异参数

由上图可知，对于浮点数编码来说，在 0.01 数量级上，中间大小的变异参

数有较好的效果，可以取 0.05~0.07。在这个区间内，不同的交叉参数能普遍地获得较好的结果。

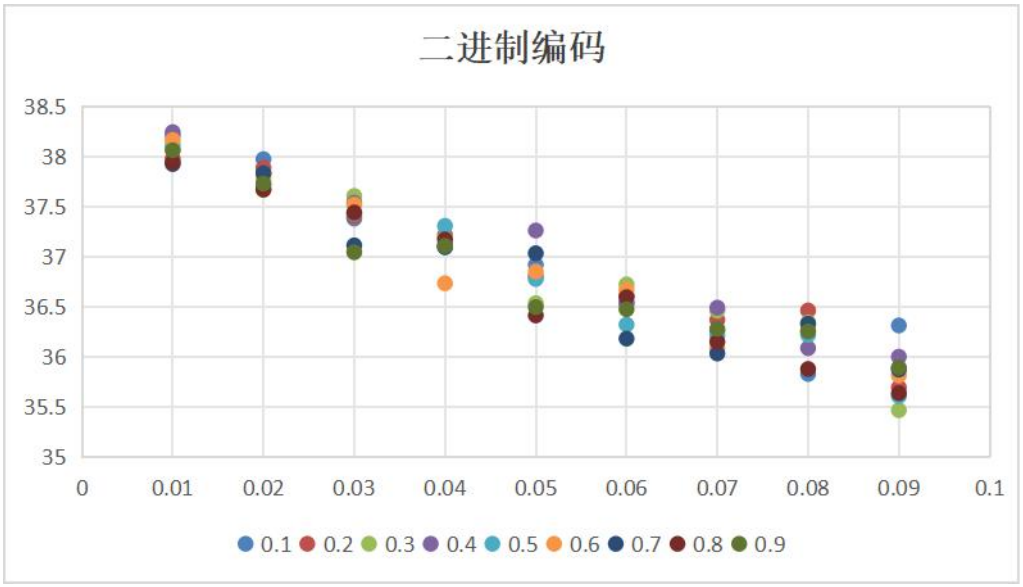

图 5 二进制编码，变异参数

由上图可知，对于二进制编码来说，越小的变异参数越为适宜。这是由于较大的基因数量对应极少的变量数目，使得一点点变异都会带来极大的改动。为了兼顾多样性，在选取较大的变异参数时，要有相应大的种群迭代次数。在较小的种群迭代次数下，如果变异参数较大，可能无法收敛。