报告题目：Maximum Cut

王崧睿 221502011 detect0530@gmail.com

1 引言

演化算法可以解决很多优化问题，其中最大割问题是一个NP难问题，本次作业使用了遗传算法来解决最大割问题。

盼来的演化算法实践！同时更加期待第四次HW！

2 任务一 基本演化算法

本阶段，我将介绍code的基本结构，以及如何运行。

2.1 演化算子

```
def Get_fitness(graph,x,n_edges,threshold=0):
    g1=np.where(x==0)[0]
    g2=np.where(x==1)[0]
    fitness = round(nx.cut_size(graph,g1,g2)/n_edges,5)
    return fitness
```

计算fitness的函数，这里使用了nx的cut_size函数，计算割集的大小，除以总边数，得到fitness。

```
def One_bit_mutation(x):
    x_new=copy.deepcopy(x)
    idx=random.randint(0,len(x_new)-1)
    x_new[idx]=x_new[idx]^1
    return x_new
```

单点变异，随机选择一个点，取反。

```
def One_point_crossover(x,y,p):
    if random.random()<p:
        idx=random.randint(0,len(x)-1)
        x_new=np.concatenate((x[:idx],y[idx:]))
        y_new=np.concatenate((y[:idx],x[idx:]))
        return x_new,y_new
    else:
        return x,y
```

基本的单点交叉，随机选择一个点，交换两个个体的片段。

```
def Cmp_fit(x):
    return x[1]

def Selection_fitness(group, number, game=0.5):
    #g_rk = sorted(group, key=Cmp_fit, reverse=True)
    g_rk=group
    new_group = []
    prob = np.array(list(map(Cmp_fit, g_rk)))
    prob = prob - prob.min()*2/3
    prob = prob / prob.sum()
    for i in range(number):
        index=np.random.choice(np.arange(0,len(group),1),p=prob)
        new_group.append(g_rk[index])
    return new_group
```

我选择按照fitness选择交叉的子代，并且按照fitness的大小设置选择的概率，同时为了防止fitness差别不大的情况，我将fitness减去最小值的2/3，然后归一化，这样可以放大fitness差别进而影响概率。

```
def Survival_with_fitness(group,number):
    g_rk = sorted(group, key=Cmp_fit, reverse=True)
    return g_rk[:number]
```

生存策略，我选择了按照fitness排序，选择前number个个体。

2.2 迭代过程

```
def SEA(args=get_args()):
    graph, n_nodes, n_edge = graph_generator(args)
    group = []
    max_fitness = 0
    Init_P(n_nodes)
    for i in range(args.size):
        x = Generate_random_binary_string(n_nodes)
        fitness = Get_fitness(graph, x, n_edge)
        group.append((x, fitness))

    iterations = []
    best_results = []
    for _ in range(args.T):
        old_group = Selection_fitness(group, args.size)
        new_group = copy.deepcopy(group)
```

```
        for i in range(args.size):
            idx, idy = np.random.choice(args.size, 2)
            x, y = old_group[idx][0], old_group[idy][0]
            x, y = Two_point_crossover(x, y, args.prob_c)
            x = Bit_wise_mutation(x, args.prob_m, graph, n_edge,old_group[idx][1])
            y = Bit_wise_mutation(y, args.prob_m, graph, n_edge,old_group[idy][1])
            fitness_x = Get_fitness(graph, x, n_edge)
            fitness_y = Get_fitness(graph, y, n_edge)
            max_fitness = max(max_fitness, fitness_x, fitness_y)
            new_group.append((x, fitness_x))
            new_group.append((y, fitness_y))

        Group = []
        for i in new_group:
            x= Heavy_tailed_mutation(i[0],args.prob_m,graph,n_edge,i[1])
            new_fitness=Get_fitness(graph,x,n_edge)
            max_fitness=max(max_fitness,new_fitness)
            Group.append((x,new_fitness))
        group = Survival_with_fitness(Group, args.size)
        iterations.append(_)
        best_results.append(max_fitness)
        if(_ % 100 == 0):
            print(_, max_fitness)
    return iterations, best_results
```

简而言之，先生成图，而后

1. 选择初始种群
2. 进入迭代
3. 选择杂交父代
4. 交叉遗传
5. 变异
6. 生存选择
7. 下一轮迭代
8. 迭代结束后保存数据

2.3 运行结果

参数选择：

- 种群size 1
- 单点变异p=0.001
- 交叉变异p=0.6
- $G1 \rightarrow G10$

可以看到，fitness不稳定，且未收敛，意味着当前算子和参数的设置还有进步空间。

3 任务二 改进变异算子

在tool_box.py中实现：

```
P=[]
def Init_P(n):
    P.clear()
    B=1.5
    sum=0
    for i in range(n//2):
        sum =sum + (i+1)**(-B)
    P.append(0)
    for i in range(n//2):
        P.append(((i+1)**(-B))/sum)
    print(P)

def Get_P():
    return P
```

这样在main中，对每一个图，只用初始化一次P，然后在变异算子中，使用调用Get_P函数来快速生成变异概率。

```python
def Heavy_tailed_mutation(x,p,graph,n_edges,ori_fitness):
    x_new=copy.deepcopy(x)
    p=Get_P()
    Len = len(x_new)
    idx = np.random.choice(np.arange(0, Len//2+1, 1), p=p)
    for num, _ in enumerate(x_new):
        if (random.random() * Len < idx):
            x_new[num] = x_new[num] ^ 1
    fitness_new = Get_fitness(graph, x_new, n_edges)
    if (ori_fitness > fitness_new):
        return x
    return x_new
```

可以发现这个函数对比之前改进了很多，首先采用Heavy Tailed分布，其次，若变异后反而更弱了，则不变异。

参数设置：

- 种群size 1
- 单点变异p= distribution with Heavy Tailed
- 交叉变异p=0.6

可以看出，优化速度和2000it收敛位置有提升，但不大，优化瓶颈不在这里。

4 任务三 算法改进

4.1 交叉算子改进

```
def One_point_crossover(x,y,p):
    if random.random()<p:
        idx=random.randint(0,len(x)-1)
        x_new=np.concatenate((x[:idx],y[idx:]))
        y_new=np.concatenate((y[:idx],x[idx:]))
        return x_new,y_new
    else:
        return x,y

def Two_point_crossover(x,y,p):
    if random.random()<p:
        idx1=random.randint(0,len(x)-1)
        #idx2=random.randint(0,len(x)-1)
        idx2=np.random.choice(np.arange(max(0,idx1-100),min(len(x),idx1+100),1))
        if idx1>idx2:
            idx1,idx2=idx2,idx1
        x_new=np.concatenate((x[:idx1],y[idx1:idx2],x[idx2:]))
        y_new=np.concatenate((y[:idx1],x[idx1:idx2],y[idx2:]))
        return x_new,y_new
    else:
        return x,y
```

我将单点交叉改进为双点交叉，这样可以增加交叉的可能性，同时，将交叉的片段长度限制在100以内，这样可以增加交叉的可能性，同时减少交叉的片段长度，防止交叉后的个体过于混乱。

4.2 变异算子改进

```
def One_bit_mutation(x):
    x_new=copy.deepcopy(x)
    idx=random.randint(0,len(x_new)-1)
    x_new[idx]=x_new[idx]^1
    return x_new
def Bit_wise_mutation(x,p,graph,n_edges,ori_fitness):
    x_new=copy.deepcopy(x)
    for num,_ in enumerate(x_new):
        if(random.random()<p):
            x_new[num]=x_new[num]^1
    fitness_new=Get_fitness(graph,x_new,n_edges)
    if(ori_fitness>fitness_new):
        return x
    return x_new

def Heavy_tailed_mutation(x,p,graph,n_edges,ori_fitness):
    x_new=copy.deepcopy(x)
    p=Get_P()
    Len = len(x_new)
    idx = np.random.choice(np.arange(0, Len//2+1, 1), p=p)
```

```
    for num, _ in enumerate(x_new):
        if (random.random() * Len < idx):
            x_new[num] = x_new[num] ^ 1
    fitness_new = Get_fitness(graph, x_new, n_edges)
    if (ori_fitness > fitness_new):
        return x
    return x_new
```

写了三种不同的变异，经过测试Heavy Tailed分布最佳。(虽然收纳值相对不那么稳定)

4.3 参数优化

对于参数来说，一般的规律是太小太大都不好，也就是说参数的 取值-fitness 画出图来大概是个凸函数。

这就启发我使用三分确定参数合理的取值。（节省时间和计算资源

对于每一个参数

```
# 三分确定合理的参数取值
l=0.0,r=1.0
while(r-l>1e-5):
    ll = (r-l)/3 + l
    rr = (r-l)/3*2 + l
    if(Performance(ll)>Performance(rr)):
        l=ll
    else:
        r=rr
args.ans=ll
# then, goto Run_with_different_Graph Function
........

def Run_with_different_graph(args=get_args()):
    param_sets = []
    for i in range(1, 11):
        param_sets.append("G"+str(i))
    results = []
    for i in range(1, 11):
        args.gset_id = i
        iterations, best_results = SEA(args)
        results.append({'params': param_sets[i-1], 'iterations': iterations,
'best_results': best_results})

    for result in results:
        plt.plot(result['iterations'], result['best_results'],
label=str(result['params']))

    plt.legend(title='Parameter Sets', loc='lower right')
    plt.title('Evolution of Best Results Over Iterations')
    plt.xlabel('Iterations')
    plt.ylabel('Best Results')
    plt.grid(True)
```

```
    plt.show()
```

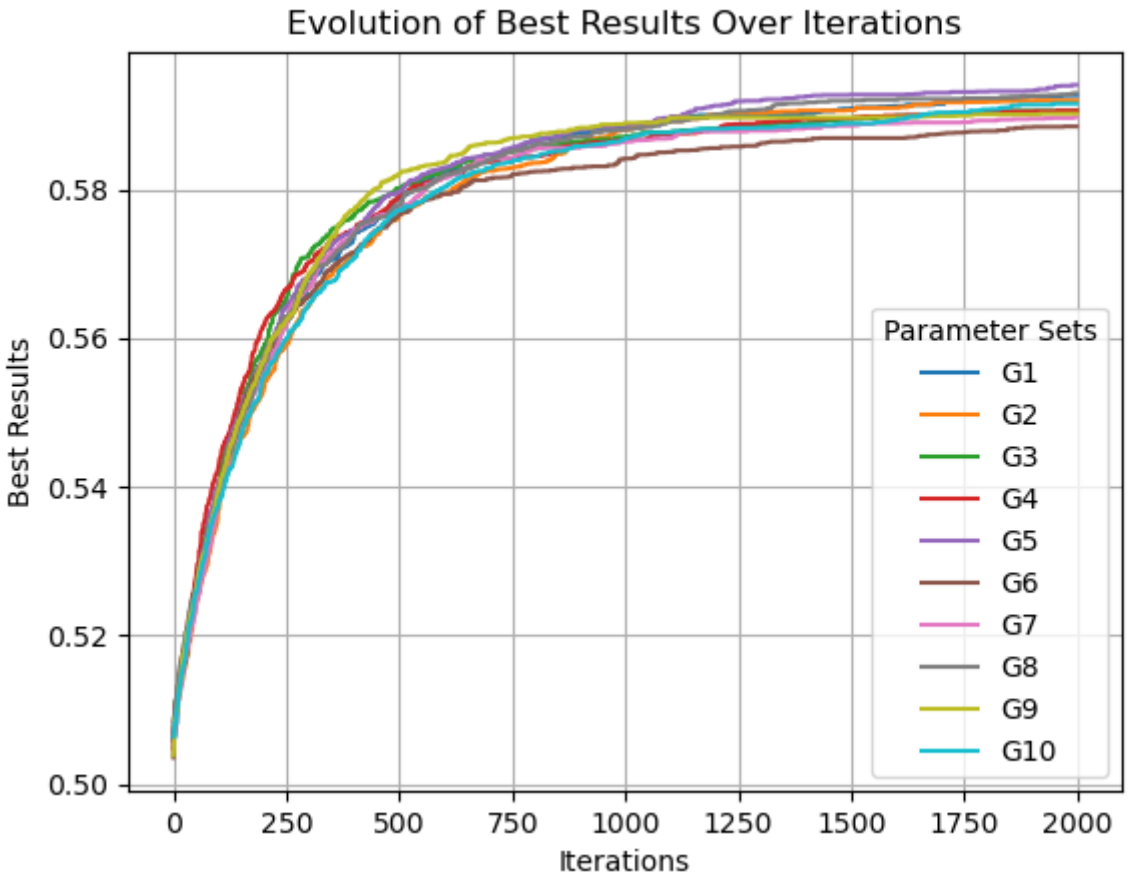
这样通过对比$G1 \rightarrow G10$的曲线图，就可以确定参数的合理值了。

同时，参数的设定还要考虑时间因素，比如种群size太大的话，跑不下来

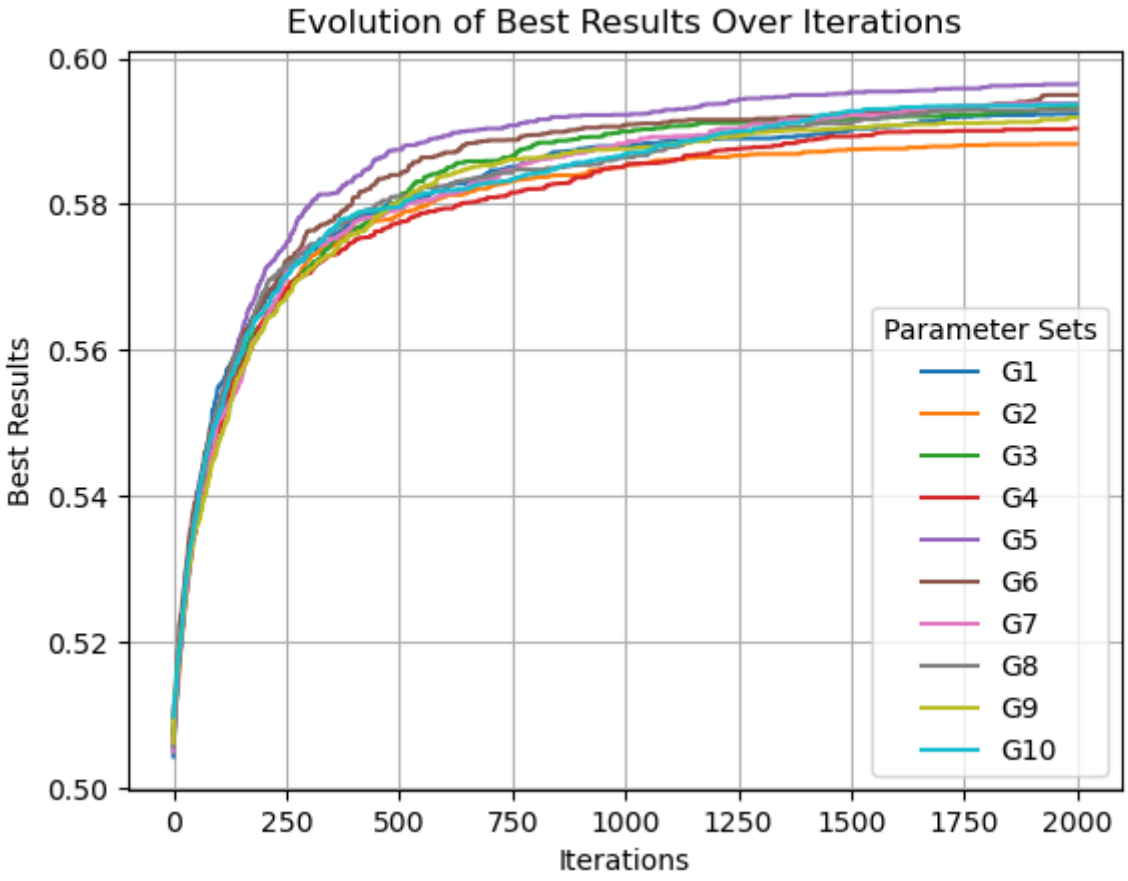
最终确定关键参数如下：

- 种群size 8
- 单点变异p=0.003
- 交叉变异p=0.7

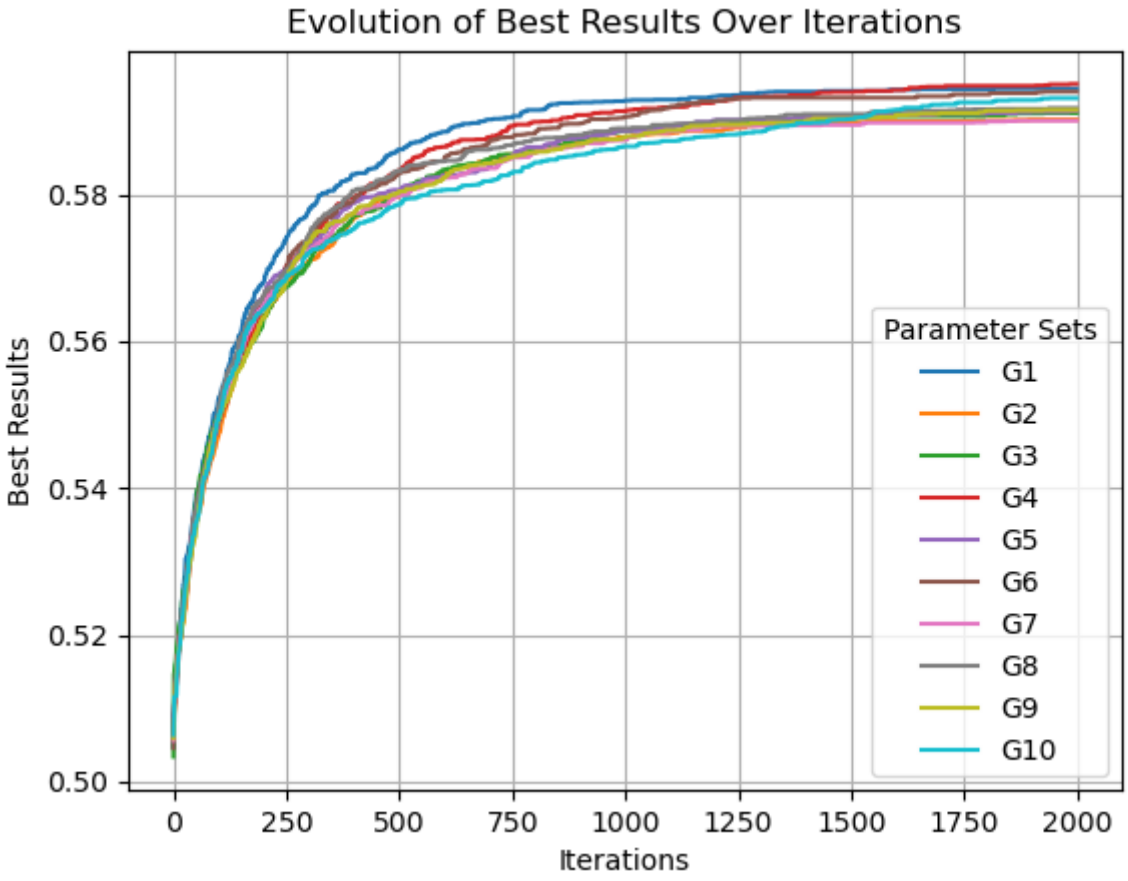
4.4 结果展示和说明



这是一开始瞎设参数跑的结果，可能是因为种群size开的很大（20），结果看着还行，但是跑了很久很久。

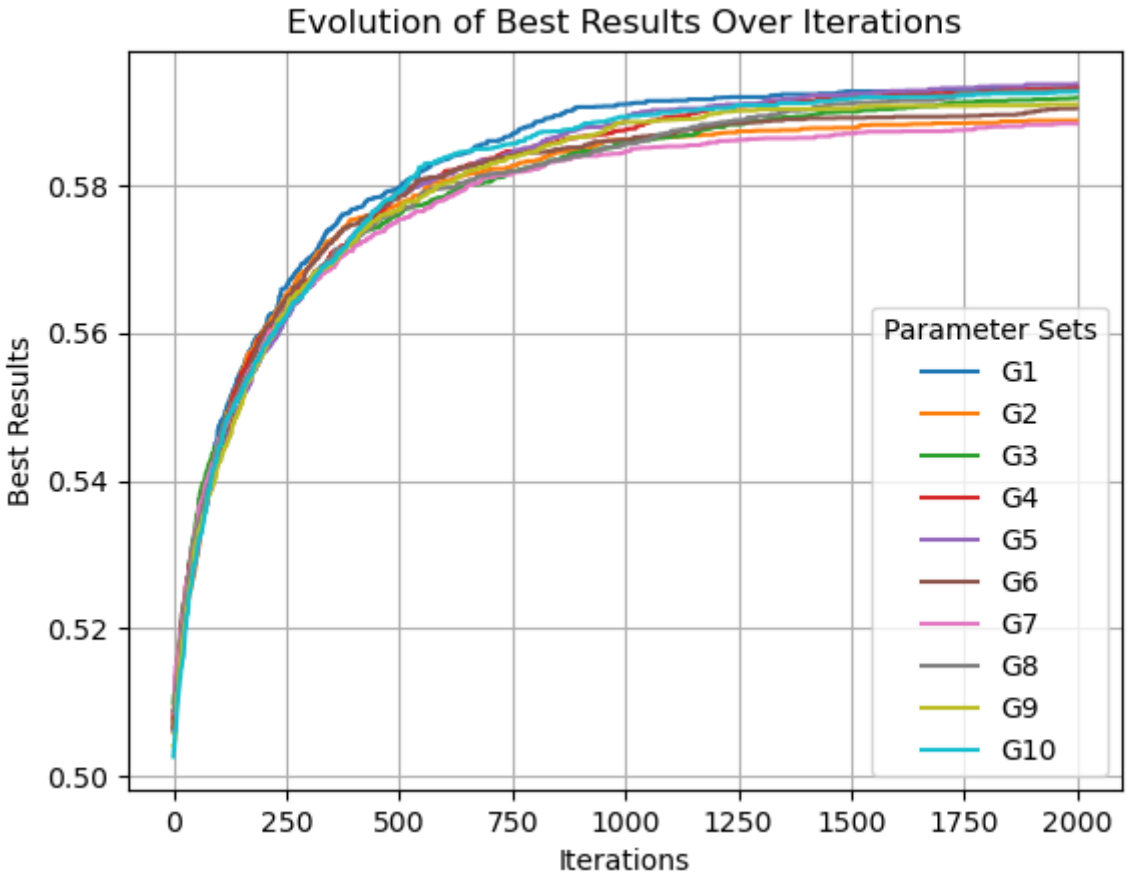


这是采用了Heavy Tailed单点变异后的结果，可以看到，收敛速度有了很大提升，比如同样是500次迭代，使用前稳定在0.58下面，采用后稳定在0.58上面，但是相对的2000次收纳结果采用Heavy Tailed mutation后反而不太稳定，猜测可能与其分布有较大变异能力导致结果差异性大。



经过三分调整参数后，可以看到前500次跌打，fitness提升速度又有了提升，同时最后收敛值也有了提升，通过参数的调整，使得算法更加合理。

以上是正面优化例子，但在实践其他优化时，也遇到了一些问题，比如：

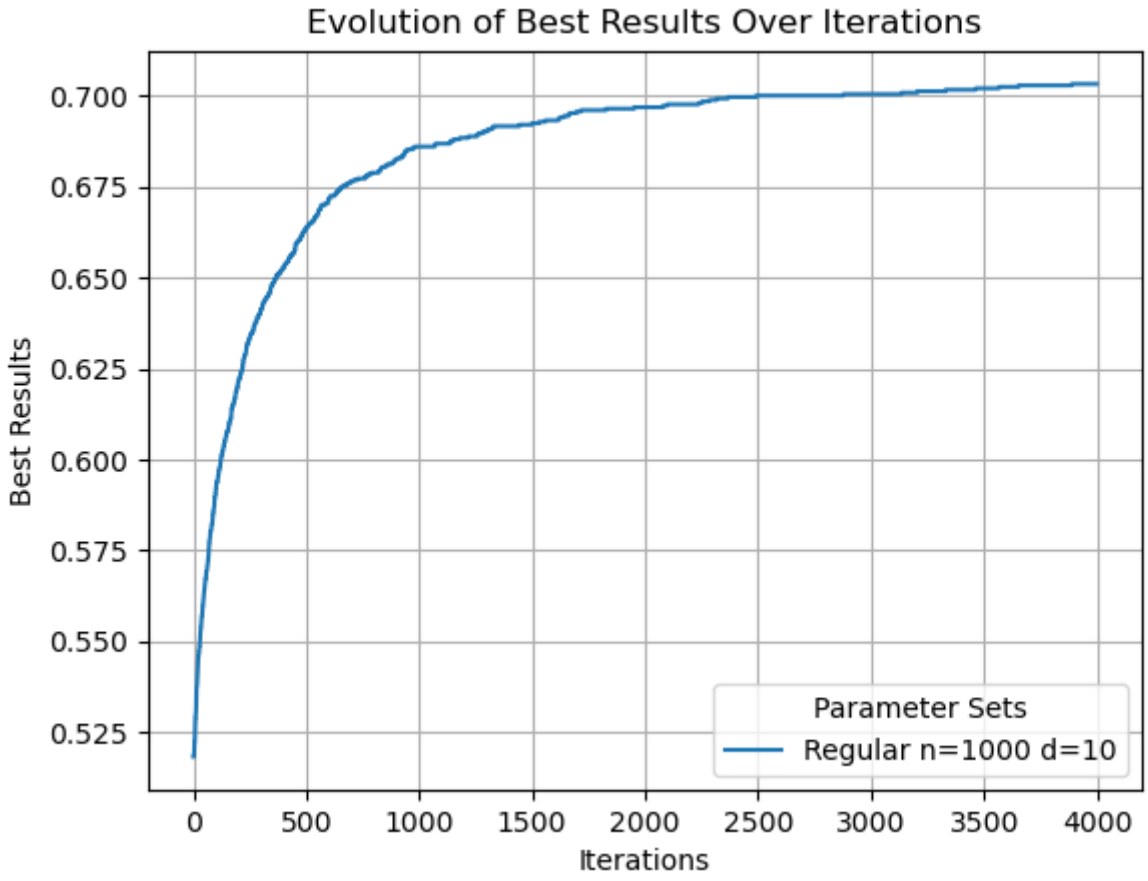


这是采用Two_point_crossover后的结果，采用了双点交叉后，收敛速度反而变慢了，这与我们期望的双点交叉降低混乱度的猜想相悖。效果甚至有略微的负提升。

```
def Two_point_crossover(x,y,p):
    if random.random()<p:
        idx1=random.randint(0,len(x)-1)
        #idx2=random.randint(0,len(x)-1)
        idx2=np.random.choice(np.arange(max(0,idx1-100),min(len(x),idx1+100),1))
        if idx1>idx2:
            idx1,idx2=idx2,idx1
        x_new=np.concatenate((x[:idx1],y[idx1:idx2],x[idx2:]))
        y_new=np.concatenate((y[:idx1],x[idx1:idx2],y[idx2:]))
        return x_new,y_new
    else:
        return x,y
```

其他图的表现

在给定的Gset - G61中，相对优秀的一支曲线，可以到0.75的fitness。



这是在随机正则图（n=1000，d=10）中的运行结果，可以看到，收敛速度很快，并且收纳值在0.7以上，这是一个很好的结果。

总之，与任务一的原始算法比较，通过优化遗传算子，调整参数，收敛速度和收敛值都得到了巨大的进步。证明了算法的有效性。

5 结语

很高兴能动手实践演化算法，之前遇到np问题总是头大，但是通过本课程的学习，对于演化算法有了更深的理解，也对于np问题有了更深的认识，希望能在之后的课程中继续学习演化算法，也希望能在之后的课程中学习到更多的np问题的解决方法。

期待HW4的实验！！！

以上