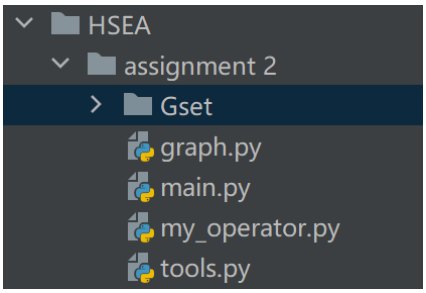
Assignment-02

演化算法求解子模优化问题-Max Cut

201300086史浩男

一、项目结构



- graph.py：生成图的函数
- main.py：SGA模型与寻参函数
- my_operator.py：演化算子函数
- tools.py：画图、计时、debug等工具性函数

测试说明

默认参数在`main.get_args()`中指定，如graph选项，问题规模，迭代轮数

SGA算法参数需要在调用`main.binary_string_group()`时指定，如`binary_string_group(args, miu=5, lamda=2,)`

运行和测试都在`main.main()`中

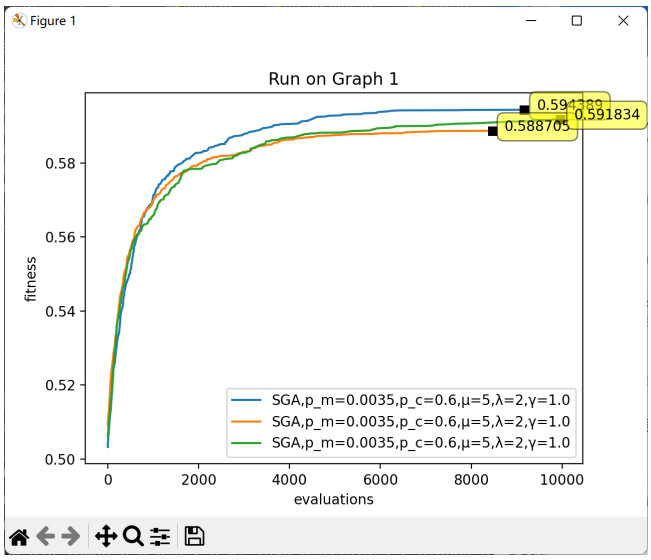
二、最优结果展示

1、Gset-graph：0.651

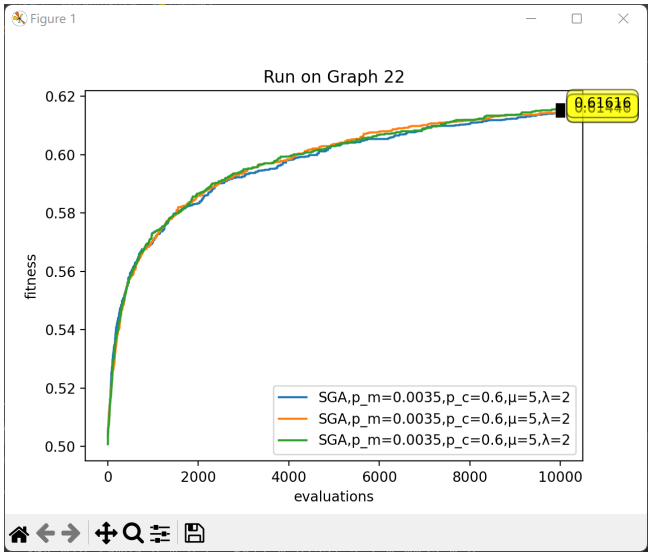
经过了不断调参优化，算法在不同Gset图上运行3次，都达到了接近0.6，甚至在部分graph上超过0.65的最优fitness:

- T迭代轮数：默认的10000

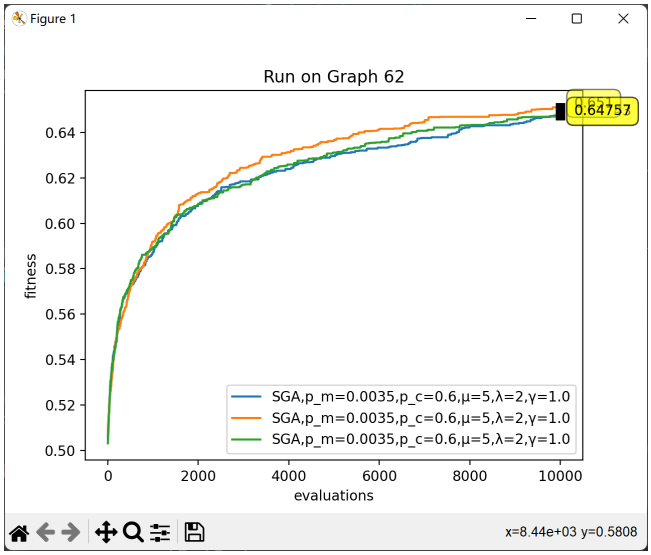
```
time= 246.59872589999577
```
`best fitness=0.59439`




```
time= 316.3431347000005 best fitness=0.61616
```




```
time= 462.0136388999963 best fitness=0.651
```



2、regular-graph：0.69984

任务规模为：

- n-nodes节点数：5000
- n-d节点度：5
- T迭代轮数：左图1w，中图4w，右图10w

图1：达到0.65的跑分，可以在5分钟很快跑完1w轮，得到结果

```
time= 342.55748519999906 best fitness=0.65144
```

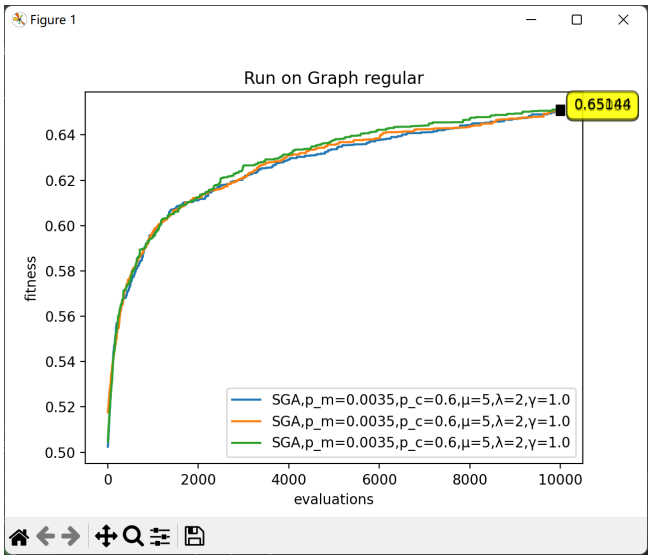


图2：达到0.684的跑分。运行时间96min，停止条件为2000轮内无性能提升，共4w次迭代

```
time= 5785.686057600004 best fitness=0.68464
```

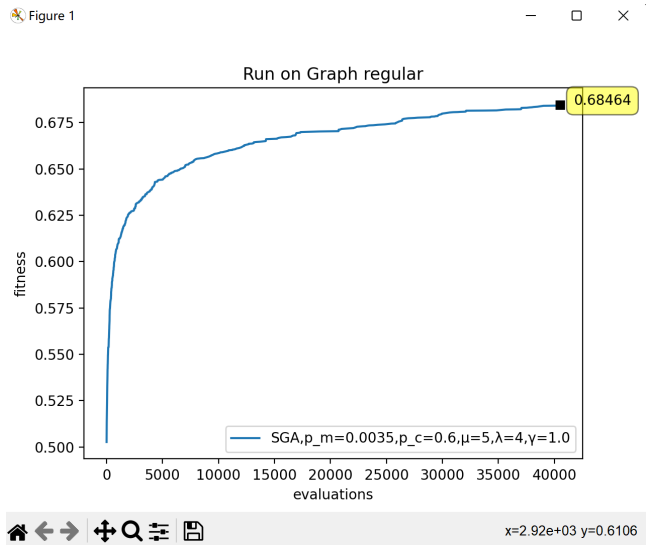
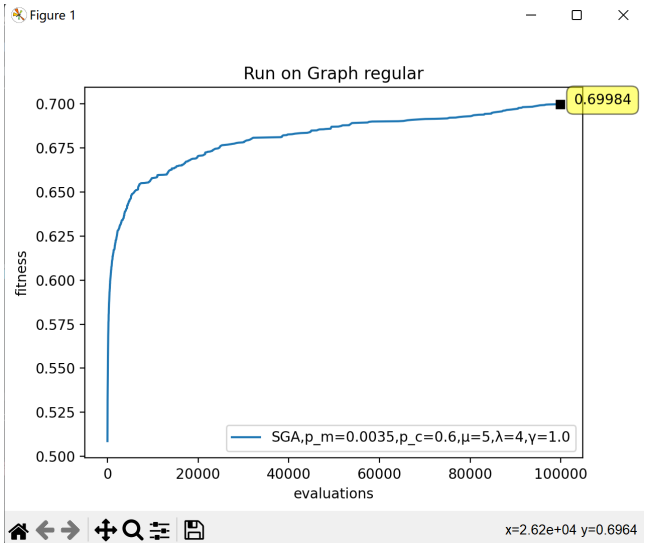


图3：达到高达0.699的跑分！！！。但运行时间226min，总共10w次迭代

```
time= 13584.065033499995 best fitness=0.69984
```



三、两个任务

任务一：设计演化算法

1、算法设计

我采用的方法是simple GA，任务一中没有进行调参和优化

（1）定义解和种群

用01串表示cut情况

```
#tools.py
def generate_binary(n):
    seed = "01"
    sa = []
    for i in range(n):
        sa.append(random.choice(seed))
    salt=''.join(sa)
    return np.array(list(map(int,salt)))
```

设置种群大小为$\mu = 4$

（2）交叉变异算子

我比较了`one_bit_mutation`和`bit_wise_mutation`，发现明显后者性能更好

```
#my_operators.py
def one_bit_mutation(x):
    bit=random.randint(0,len(x)-1)
    x[bit]=x[bit]*-1
    return x

def bit_wise_mutation(x,p):
    for num,i in enumerate(x):
        seed=random.randint(1,len(x))
        if seed<int(len(x)*p):
            x[num]*=-1
    return x
```

交叉算子使用适合SGA的`one_point_crossover`

```
#my_operators.py
def one_point_crossover(x,y,p):
    seed=random.randint(0,len(x)-1)
    if seed<int(len(x)*p):
        x1=np.hstack((x[:seed],y[seed:]))
        y1 = np.hstack((y[:seed], x[seed:]))
        return x1,y1
    return x,y
```

(3) 父代选择--FPS

将所有父代的fitness值减去最小父代fitness的1/2，再归一化以fitness占比作为抽中的概率

```
#my_operators.py
def takeSecond(elem):
    return elem[1]

def fitness_propotional_selection(group, lamda):
    """
    :param group: parent
    :param lamda: 用于交配父代个数，也是即将产生子代个数
    :return: 选出的parent集合
    """
    g=sorted(group,key=takeSecond)#按fitness升序
    pro = np.array(list(map(lambda x: x[1], g)))
    pro=pro-pro[0]/2
    pro=pro/pro.sum()
    new_group=[]
    for i in range(lamda):
        index = np.random.choice(np.arange(len(g)), p=pro)
        new_group.append(g[index])
    return new_group
```

（4）生存者选择--fitness-based

选择fitness最佳的μ个作为survival

```
#my_operators.py
def survival_best_miu(newgroup,miu):
    g=sorted(newgroup,key=takeSecond)  # 按fitness升序
    return g[-miu:]
```

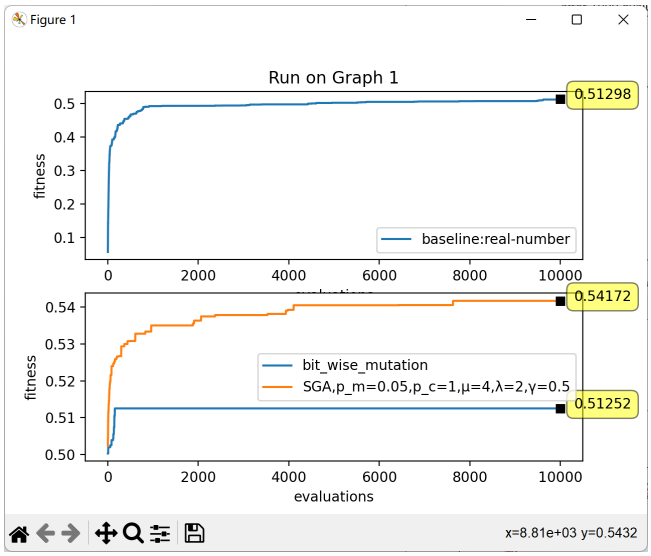
2、问题规模与效果

- graph采用已提供的Gset
- T迭代轮数：默认的10000

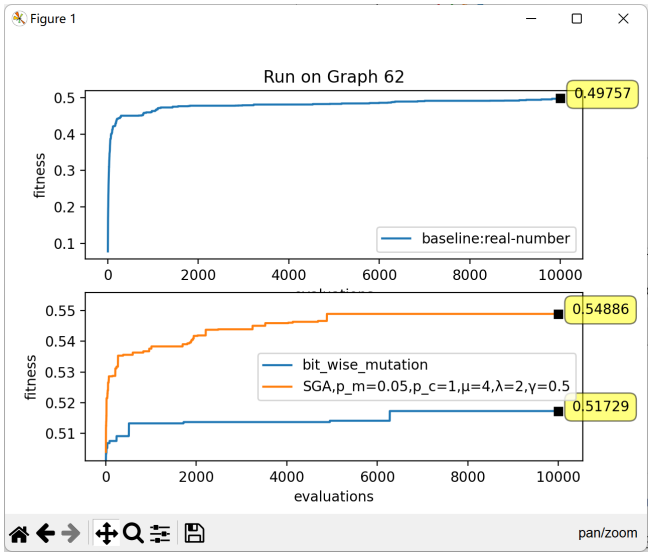
在图 G1，G62上运行的效果如下：

其中，baseline是框架代码中用实数来进行解表示的算法，同时我比较了只使用bit-wise mutation和完整SGA的效果

```
time= 284.784278799998
```




```
time= 520.357285099999
```



任务二：演化算法改进

在任务二中，保持问题规模与任务一相同，且都是在Gset 1上进行调参测试

针对演化算子和参数的改进，我先后进行了如下尝试：

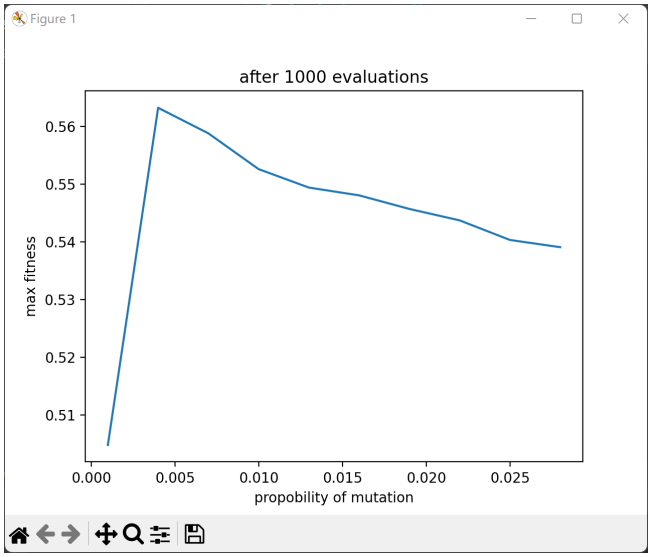
- mutation概率p
- 种群大小μ和父代挑选数λ
- 繁殖方式
- crossover概率p_c
- 减小FPS选择压力--调参γ

其中效果最显著的是mutation概率p，直接使 best fitness从0.53提升到0.59

详细尝试过程如下：

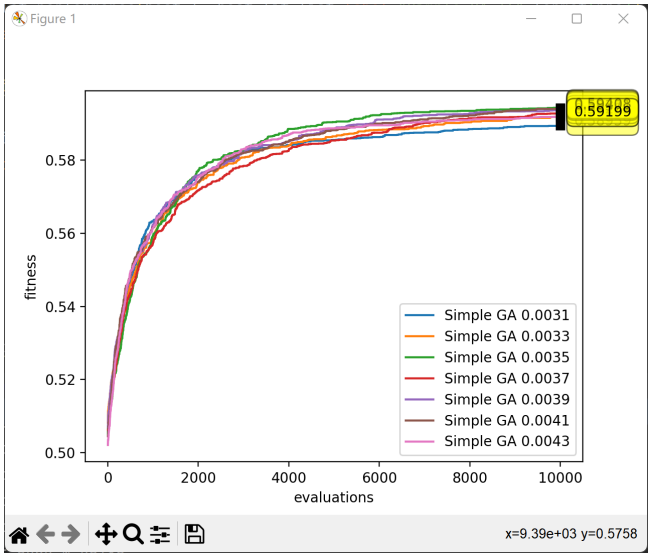
1、改进mutation概率p

针对目前效果最好的SGA算法，我先是遍历了`p=(0,0.025)`，以迭代1000次后最优fitness为指标，作图如下：



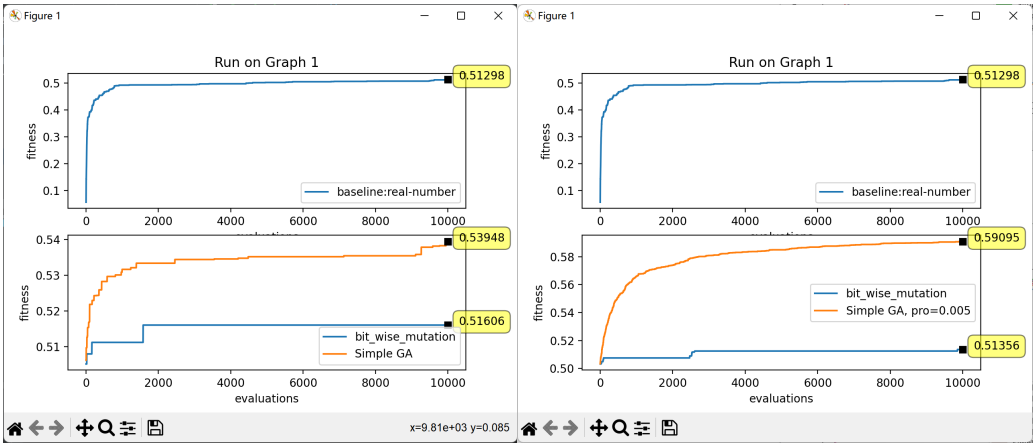
发现p较小时无法得到有效优化，最优p值大概在`p=(0.031,0.043)`之间

继续缩小p的范围，增大迭代次数到1w，作图如下：



于是得出结论，在迭代10000次时，**最优参数p约为**`0.0035`

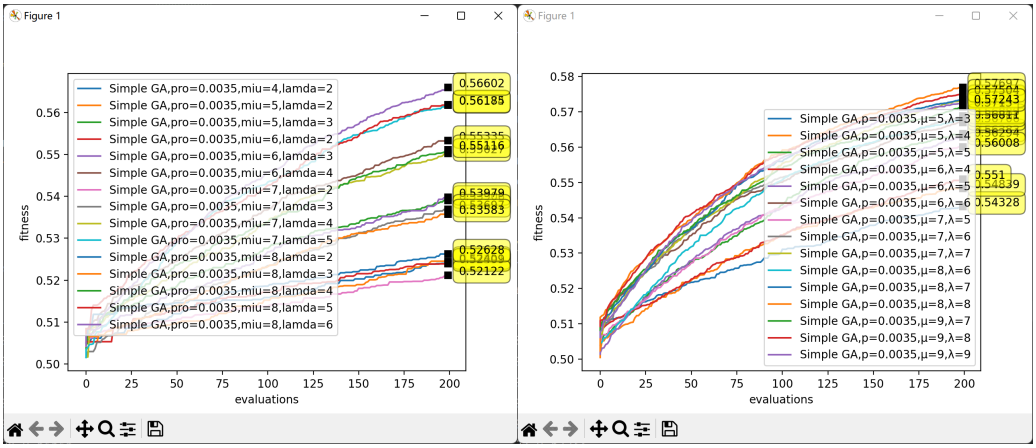
改进后的运行效果对比图如下（左图改进前p=0.05，右图改进后p=0.0035）



2、改进种群大小μ和父代挑选数λ

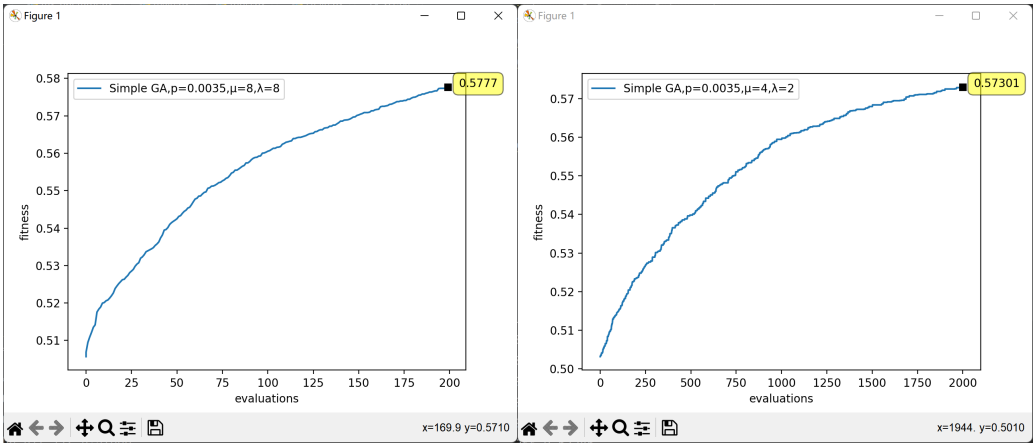
- μ：维持的种群大小
- λ：从种群μ中挑选出进行crossover和mutation的父代个数

保持`p=0.0035`不变，改变μ和λ的大小，迭代200次，性能如下两图：



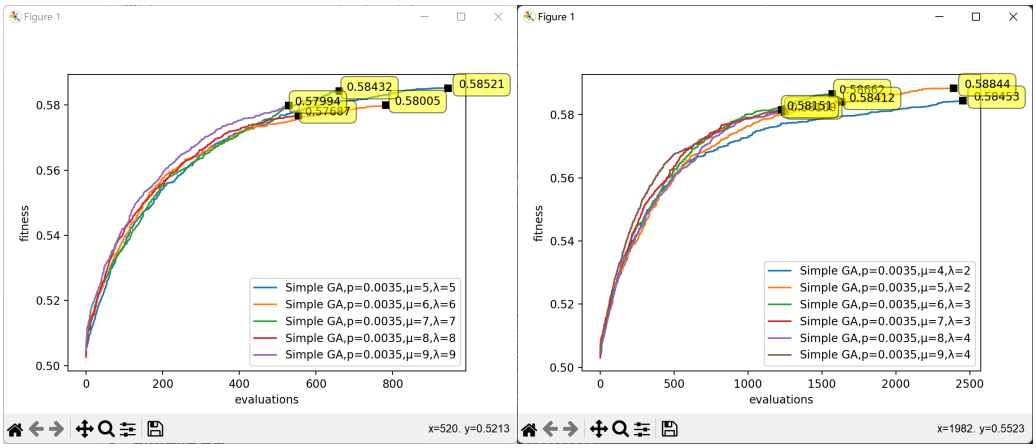
观察发现：**λ越大，性能越好。μ越大，性能越好。μ-λ越小，性能越好**

增大μ和λ后，只需要迭代200次，就可以达到之前迭代2000次才能达到的fitness值。（但每次迭代时间开销增大）



考虑到算法运行时间，我将每组参数运行时间设为1min，迭代次数上限设为1000次，运行结果如左图：

为节省时间，减小μ和λ，将每组参数运行时间设为1min，迭代次数上限设为10000次，运行结果如右图：



发现**μ和λ较大时虽然fitness上升需要的轮数少，但是时间开销更大，得不偿失。**

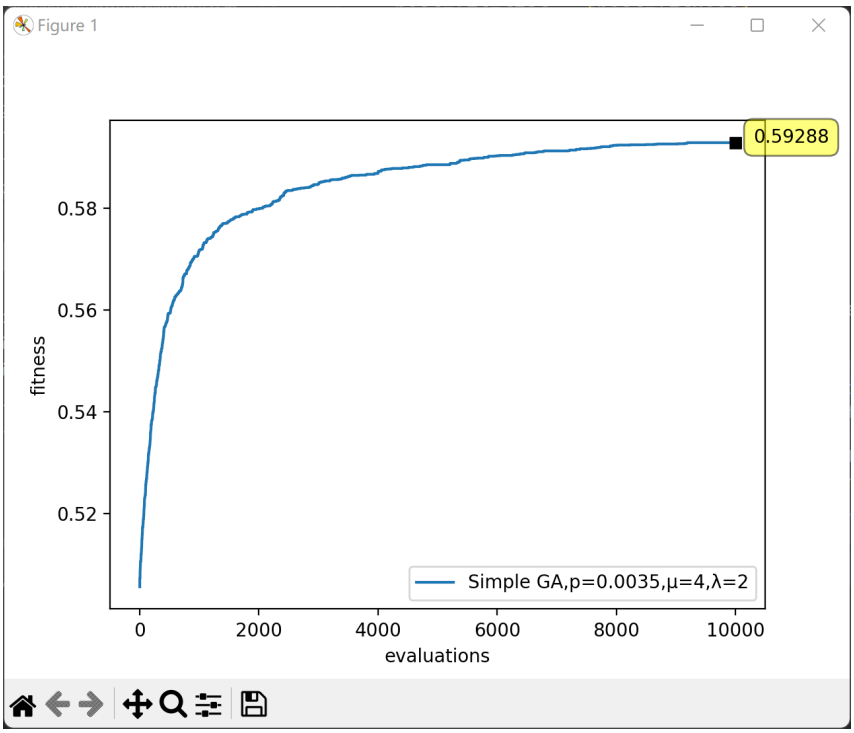
结论：最优（μ，λ）组合为（5，2）

3、改进繁殖方式

之前办法：在选出的λ个父代中两两交配（执行mutation和crossover），再在当前（$\mu+\frac{\lambda(\lambda+1)}{2}$）个个体中选出最优的$\mu$个

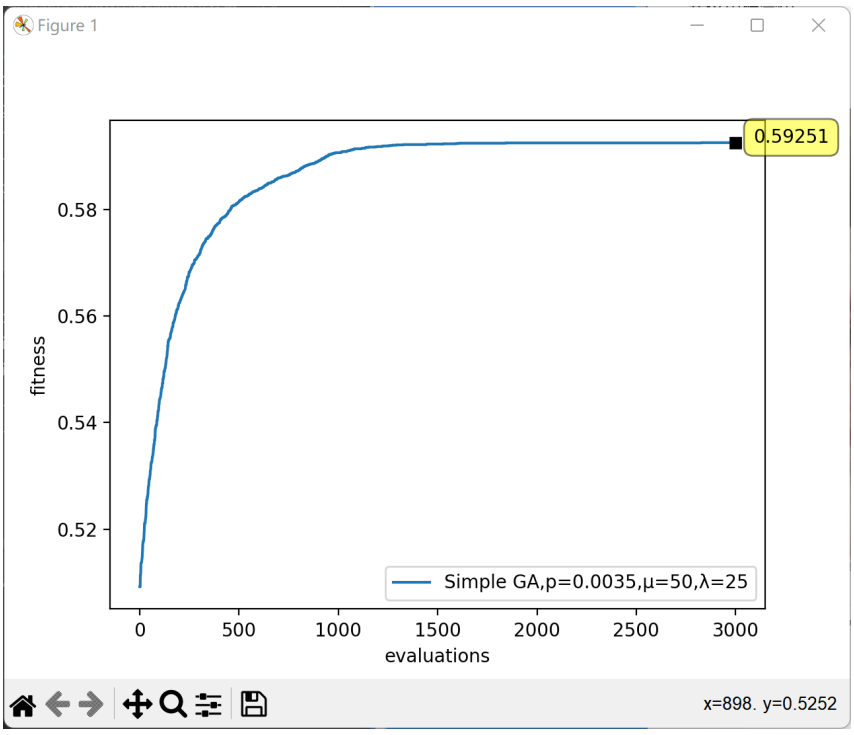
现改为：在选出的λ个父代中相邻交配，再在当前（$\mu+2\lambda$）个个体中选出最优的μ个，改进后计算速度提升明显：

```
time= 246.59872589999577
```



如果扩大种群规模，也可以得到较好结果，但时间开销大幅度上升

```
time= 905.9333375999995
```

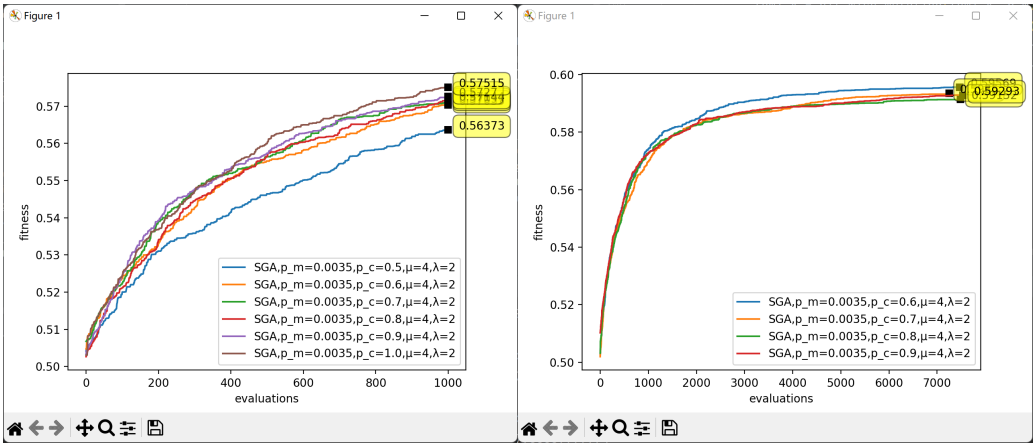


4、改进crossover概率p_c

控制其他参数不变进行最优p_c搜索：

左图：先迭代1000次，观察不同p_c大致情况

右图：控制运行时间不变，遍历$p_c = (0.6, 1)$，时间设为200s，迭代上限10000次，比较fitness值

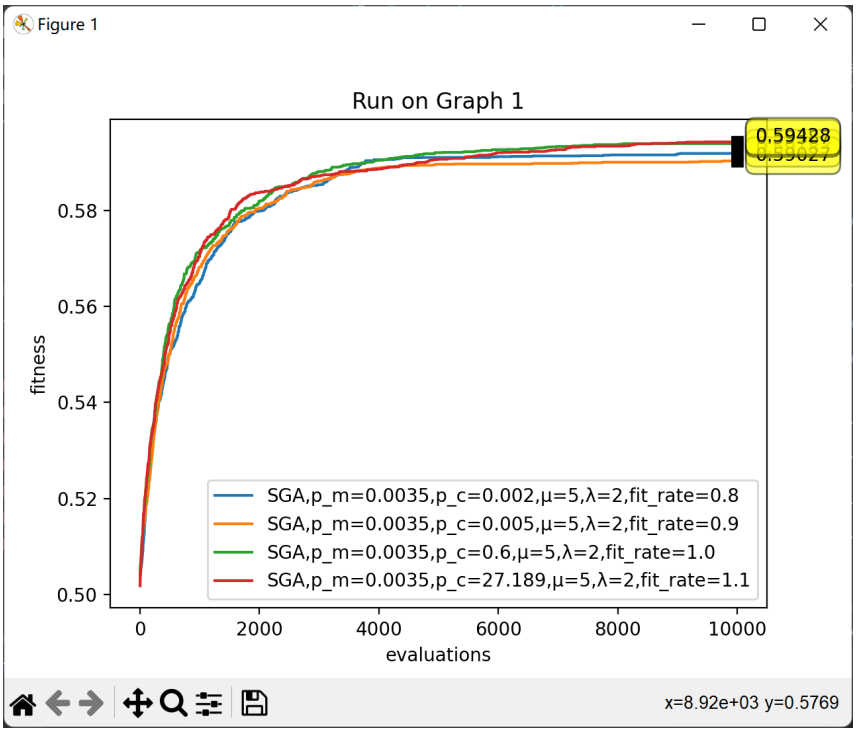


发现在训练初期p_c越大，fitness提升效果越好。但随着迭代次数的增加，较小的p_c会发生**性能反超**

尝试crossover概率p_c的动态变化

思路来源：针对max cut问题，搜索可能在最优解附近跳动，因此探究在已经搜索到较好解时，如果连续100次迭代都没有得到更好的解，则适当以一定比例改变p_c

设置100次迭代无性能提升后的p_c变化比例为`fit_rate`，在`fit_rate=(0.8,1.1)`间搜索：



因此并不需要`fit_rate`

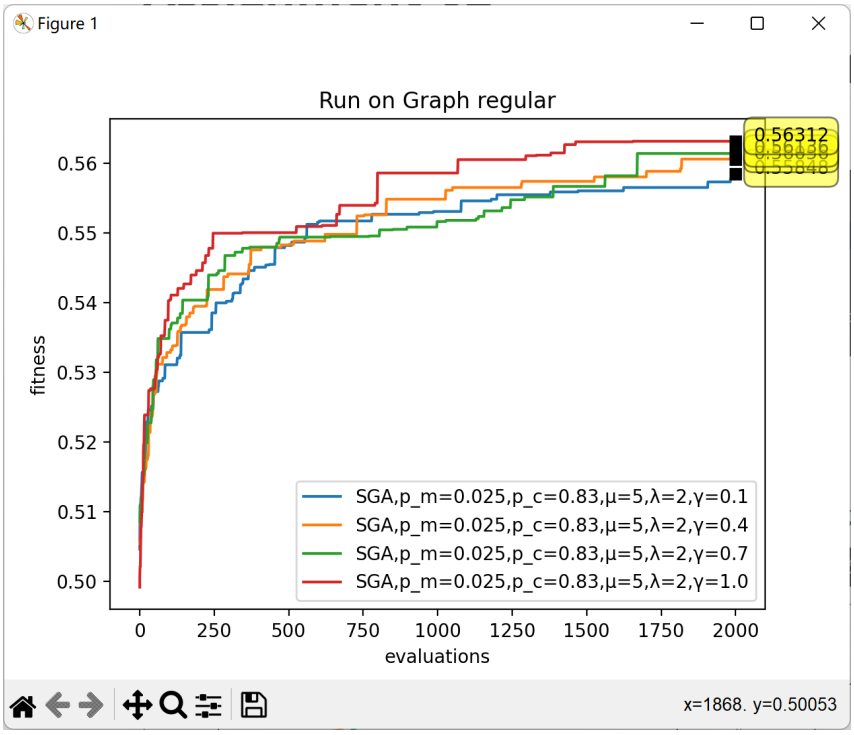
结论：优化crossover概率为0.6

5、减小FPS选择压力--调参γ

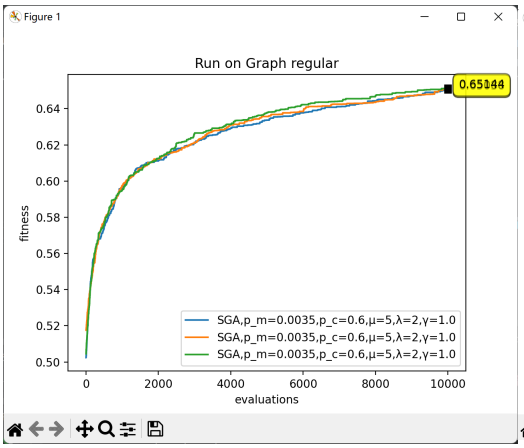
γ的意义是，在确定所有父代解被选中概率时，要减去最小父代fitness的多少倍

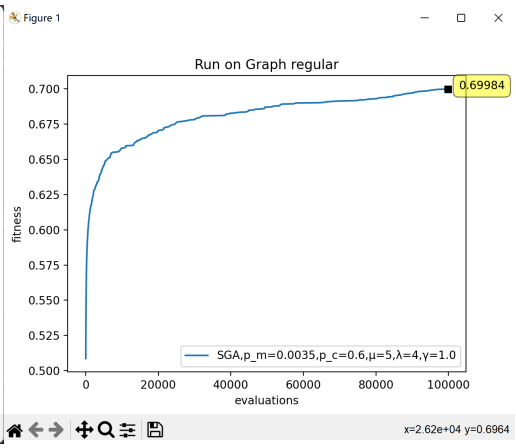
γ=1即所有父代都减去最小父代的fitness后再进行概率归一运算

```
g=sorted(group,key=takeSecond)#按fitness升序
pro = np.array(list(map(lambda x: x[1], g)))
pro=pro-pro[0]*gama
pro=pro/pro.sum()
```



如图所示，γ=1有显著的优势，因此确定将γ固定为1





四、感悟与收获

我非常喜欢探索更好演化算法的这一过程，花了整整一天多的时间调各种各样的参，但乐在其中

- SGA相比只用mutation的效果提升很大，因为可以将较好的解保存在种群中，最好的解可以一直留在种群中，而不是下一轮就被替换掉，我想这也就是"遗传"的意义
- 对效果影响最大的参数的mutation的概率
- 增大种群数量可以在达到相同fitness的情况下减少轮数，但如果限制同样的运行时间，发现还是小种群性价比最高，所以最后我选择了$\mu = 5$的小种群
- 不同的问题规模，最适合的参数也不一样。（平衡时间效率）
- 我很高兴在regular图上跑出了0.69984的高分，如果迭代轮数或种群规模进一步增加，我相信可以突破0.7。
- 算法当然还可以进一步优化，时间有限，就留给作业4的竞赛中再去探索吧！