一、使用启发式搜索算法求解下述数独问题。

			7		2			
1				4				7
6	5						9	4
4	7		8		1		6	2
5	8		2		9		1	3
8	6						7	5
9				6				8
			9		8			

用一个二维数组表示该数独表，其中空白的数字填 0。

```
a = [[0, 0, 0, 7, 0, 2, 0, 0, 0],
     [1, 0, 0, 0, 4, 0, 0, 0, 7],
     [6, 5, 0, 0, 0, 0, 0, 9, 4],
     [4, 7, 0, 8, 0, 1, 0, 6, 2],
     [0, 0, 0, 0, 0, 0, 0, 0, 0],
     [5, 8, 0, 2, 0, 9, 0, 1, 3],
     [8, 6, 0, 0, 0, 0, 0, 7, 5],
     [9, 0, 0, 0, 6, 0, 0, 0, 8],
     [0, 0, 0, 9, 0, 8, 0, 0, 0]]
```

首先定义估价函数：

$$f(n)=g(n)+h(n)$$

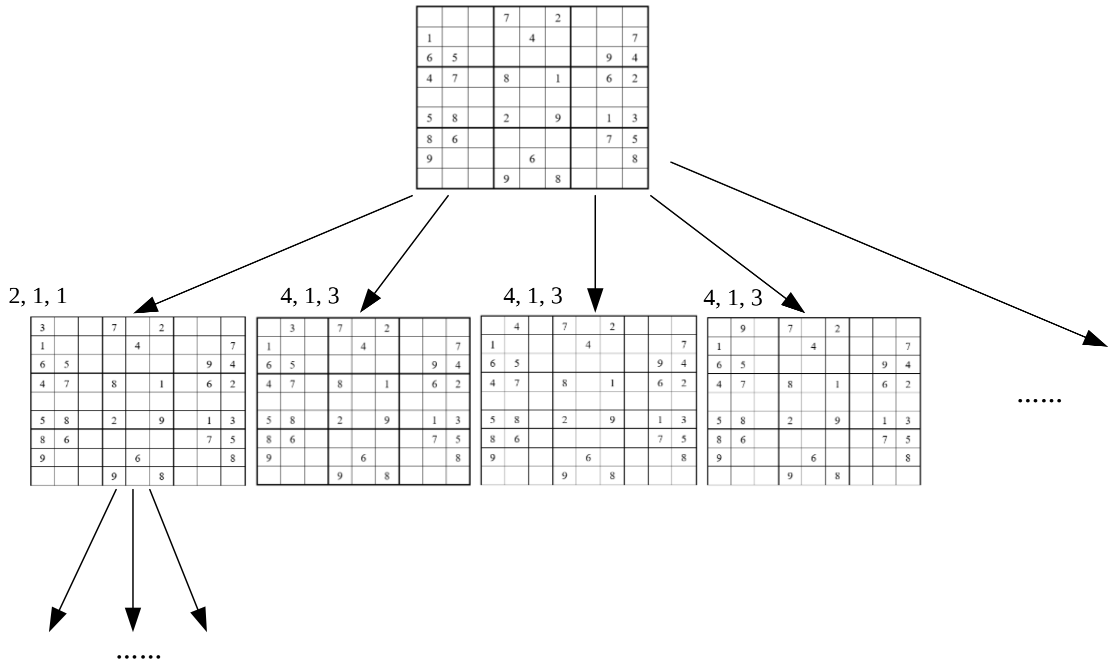
其中，$g(n)$为从初始节点到节点 n 已经付出的实际代价，即搜索的步次；$h(n)$为当前方格中可填入的不冲突的数字数，可填入的数字数越少，越接近目标值。

根据这一估价函数，每次对每一方格内的可填入数字进行计算，即 $h(n)$的值，程序如下：

```
def count(arr, row, col):
    n = 0
    num_list = []
    for num in range(1, 10):
        if (test_conflict(arr, num, row, col)):
            n = n + 1
            num_list.append(num)
    return n, num_list
```

```
def test_conflict(arr, num, row, col):
    for i in range(9):
        if i != row and num == arr[i][col]:
            return False
        if i != col and num == arr[row][i]:
            return False
    for i in range(row//3 * 3, row//3 * 3 + 3):
        for j in range(col//3 * 3, col//3 * 3 + 3):
            if((i != row or j != col) and num == arr[i][j]):
                return False
    return True
```

将每一方格内的可填入数字均作为一个节点，放入 OPEN 表中，根据 $f(n)$的值对 OPEN 表中的节点进行排序，选择 OPEN 表中 $f(n)$的值最小的节点再次进行扩展，按照全局启发式搜索的流程进行不断搜索（如下图所示）。

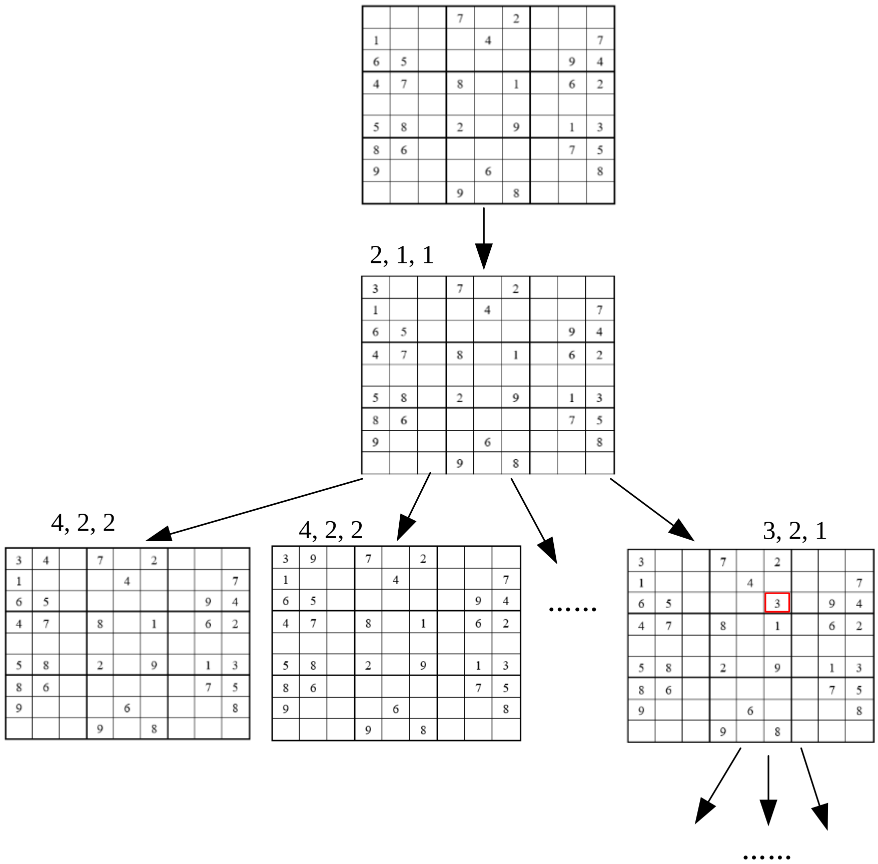


程序如下：

```python
def shudu(a, iter_num):
    if(complete(a)):
        print(a)
        return
    iter_num = iter_num + 1
    for row in range(9):
        for col in range(9):
            if (a[row][col] == 0):
                n, num_list = count(a, row, col)
                if(n == 0):
                    minIndex = index.index(min(index))
                    a = result[minIndex]
                    del(result[minIndex])
                    del(index[minIndex])
                    shudu(a, iter_num)
                else:
                   for num in num_list:
                        b = copy.deepcopy(a)
                        b[row][col] = num
                        result.append(b)
                        index.append(n+iter_num)
    minIndex = index.index(min(index))
    a = result[minIndex]
    del(result[minIndex])
    del(index[minIndex])
    shudu(a, iter_num)
    return
```

问题：搜索空间很大，有很多无用的搜索，程序跑不出来。

进一步增加先验知识：当某一空格的$f(n)$值最小时，仅将该空格所对应的节点添加进入 OPEN 表中，而非所有空格对应的节点，从而大大减小不必要的搜索。原理如下：



程序如下：

```python
def shudu(a, iter_num):
    if(complete(a)):
        print(a)
        return
    iter_num = iter_num + 1
    min_n = 10
    for row in range(9):
        for col in range(9):
            if (a[row][col] == 0):
                n, num_list = count(a, row, col)
                if(n == 0):
                    minIndex = index.index(min(index))
                    a = result[minIndex]
                    del(result[minIndex])
                    del(index[minIndex])
                    shudu(a, iter_num)
                    return
                elif n < min_n:
                        min_n = n
                        min_row = row
                        min_col = col
                        min_num_list = num_list.copy()
    for num in min_num_list:
        b = copy.deepcopy(a)
        b[min_row][min_col] = num
        result.append(b)
        index.append(min_n+iter_num)
    minIndex = index.index(min(index))
    a = result[minIndex]
    if(len(result) == 0):
        print('false')
        return 0
    del(result[minIndex])
    del(index[minIndex])
    shudu(a, iter_num)
    return
```

最终搜索结束条件为数独表中所有数字均已填上，即不存在 0 值，程序如下：

```python
def complete(a):
    for i in range(9):
        for j in range(9):
            if (a[i][j] == 0):
                return False
    return True
```

该题为的最终解如下：

3	4	8	7	9	2	6	5	1

1	9	2	6	4	5	8	3	7
6	5	7	1	8	3	2	9	4
4	7	9	8	3	1	5	6	2
2	1	3	4	5	6	7	8	9
5	8	6	2	7	9	4	1	3
8	6	1	3	2	4	9	7	5
9	3	4	5	6	7	1	2	8
7	2	5	9	1	8	3	4	6

观察一下搜索过程，由于该数独较为简单，每次都能找到唯一值的方格，直接可以搜索出结果，没有回溯的过程，因此选择了一个难度更高的数独如下：

8		9	7		5			6
		6				7		
				6	2			8
	7			2	4	8		
				8	3			
		5	9	7			3	
3				1				
		2				5		
9			2		6	3		1

采用本方法也可以将结果计算出来，如下所示。搜索的过程出现了回溯的过程。

8	3	9	7	4	5	1	2	6
5	2	6	1	9	8	7	4	3
7	1	4	3	6	2	9	5	8
6	7	3	5	2	4	8	1	9
2	9	1	6	8	3	4	7	5
4	8	5	9	7	1	6	3	2
3	5	8	4	1	9	2	6	7
1	6	2	8	3	7	5	9	4
9	4	7	2	5	6	3	8	1

二、分别使用 SA、GA 算法求解（解精度：.后 5 位）。

$$\max f(x) = x \cdot \sin(3x), \quad -1 \le x \le 30$$

1、SA 算法

1）目标函数的确定

因为模拟退火的原理是使能量最低，因此取目标函数为-$x$sin(3x)，最终使其达到最小。

2）算法参数的确定

令初始接受概率 P_0 = 0.9，因为搜索邻域大小为 6（在后面会介绍原因），所以

$$\Delta f = \Delta x \cdot \sin\left(3 \cdot \Delta x\right) \le \left|\Delta x\right| = 6$$

则由 Metropolis 准则可以得知，初始温度为：

$$T_0 = -\frac{\Delta f}{\ln P_0} = 56.9$$

则取初始温度 T_0 为 60.

令 Markov 链长度 L_k = 1000，停止准则为降温总次数不大于 K = 600，这两个参数可调，寻找合适的值。

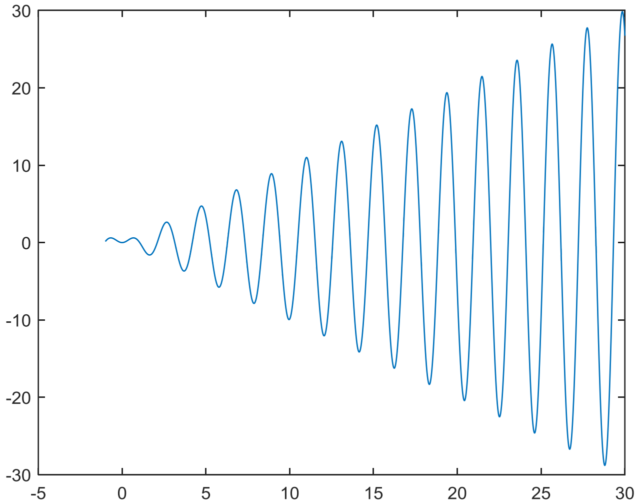
3）冷却进度表的确定

$$T_k = \alpha T_{k-1}$$

其中，α = 0.95.

4）领域搜索

x 的初始值为[-1, 30]内随机产生，考虑到函数 $x$sin(3x)的周期为 2π/3，则搜索邻域需大于 2π/3，才能确保不管当前 x 在哪，都能搜索到下一个高峰，如下图所示。



因此，在 $x\pm3$ 的邻域内随机搜索，同时需要保证 x 的值不超过[-1, 30]的范围，程序如下：

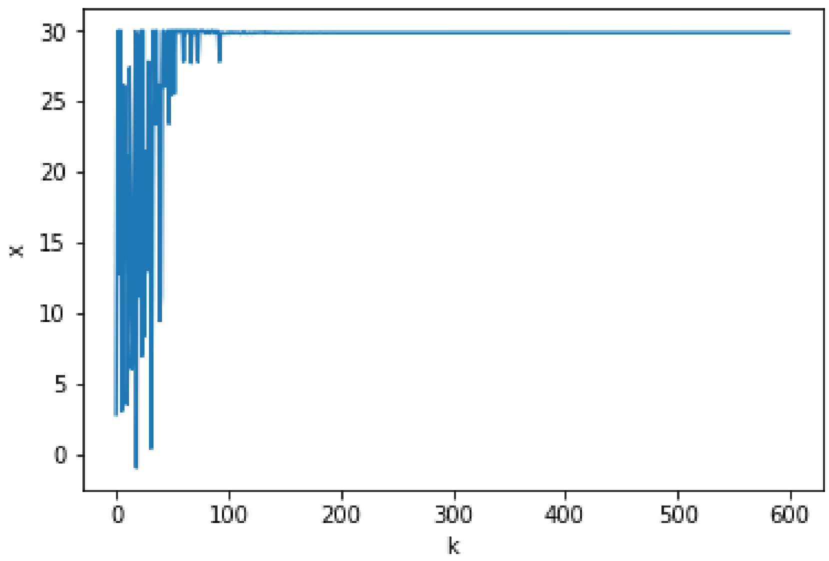
```
def find(x, min_x, max_x):
    new_x = x + random.uniform(-3, 3)
    if new_x < min_x:
        new_x = min_x
    elif new_x > max_x:
        new_x = max_x
    return new_x
```

5）模拟退火过程

分为内循环和外循环，内循环通过 L_k 次搜索到达热平衡，每次搜索若得到更优的目标函数值，则更新 x，否则依概率更新；外循环通过逐渐冷却温度，最终得到最优解，程序如下：

```
for k in range(K):
    temp = x[k]
    for l in range(L):
        f = func(temp)
        new_x = find(temp, min_x, max_x)
        new_f = func(new_x)
        if f > new_f:
            temp = new_x
        elif math.exp(-(new_f - f)/T[k]) > random.uniform(0, 1):
            temp = new_x
    x.append(temp)
    T.append(alpha * T[k])
```

搜索过程如下图所示，最优解为 29.84885，对应的最优值为 29.84792。



2、GA 算法

1）编码与初始种群确定

因为-1 ≤ x ≤ 30，精度为小数点后 5 位，所以将区间改为$[0, 31\times10^5]$，又因为 $2097152 = 2^{21} < 31\times10^5 < 2^{22} = 4194304$，所以编码长度为 22 位。

选取种群大小为 40，初始种群随机生成，代码如下：

```
def init(chromnum, chromlen):
    return random.randint(0, 2, size = (chromnum, chromlen))
```

根据编码求取 x 的原理为：

$$x' = \left(\sum_{i=0}^{21} b_i \times 2^i\right)_{10}$$

$$x = -1 + x' \times \frac{30-(-1)}{2^{22}-1}$$

从而保证编码与 x 一一对应，程序如下：

```python
def decode(pop):
    n, l = pop.shape
    result = zeros((1, n))
    for i in range(l):
        result = result + pow(2, l - 1 - i) * pop[:, i]
    return result.T * 31/(pow(2, 22) - 1) - 1
```

2）适应值求取

根据目标函数可求取每一编码的函数值：

```python
def func(pop):
    x = decode(pop)
    return multiply(x, sin(3 * x))
```

对目标函数值进行转换，因为适应值需大于等于 0，所以目标函数值大于等于 0 的染色体适应值取目标函数值，小于 0 的适应值取 0，代码如下：

```python
def fitval(val):
    fit = val.copy()
    fit[val < 0] = 0
    return fit
```

3）选择

根据适应值计算各染色体对应的存活率 $f_i/\sum f_i$，采用偏置轮盘选择方法，生成 40 个[0, 1]内的随机数作为选择概率，根据存活率选取 40 个染色体组成的新一代群体：

```python
def select(pop, fit):
    fit = cumsum(fit/sum(fit), axis = 0)
    n, l = pop.shape
    newpop = zeros((n, l))
    select_prob = random.random(size = (1, n))
    for i in range(n):
        for j in range(n):
            if select_prob[0, i] < fit[j]:
                newpop[i, :] = pop[j, :]
                break
    return newpop
```

4）交叉

设定交叉概率为 $p_c=0.8$，随机生成 20 个[0, 1]内的随机值与交叉

概率比较，交叉概率大于随机值才进行交叉，并随机生成交叉位置。考虑到需随机选择染色体进行交叉，因此先对 40 条染色体的顺序进行打乱，再按顺序两两选择。程序如下：

```
def cross(pop, pc):
    n, l = pop.shape
    newpop = zeros((n, l))
    cross_prob = random.random(size = (1, n//2))
    cross_pos = random.randint(1, l-1, size = (1, n//2))
    random.shuffle(pop)
    for i in range(0, n, 2):
        if cross_prob[0, i//2] < pc:
            newpop[i, 0:cross_pos[0, i//2]] = pop[i, 0:cross_pos[0, i//2]]
            newpop[i, cross_pos[0, i//2]:] = pop[i+1, cross_pos[0, i//2]:]
            newpop[i+1, 0:cross_pos[0, i//2]] = pop[i+1, 0:cross_pos[0, i//2]]
            newpop[i+1, cross_pos[0, i//2]:] = pop[i, cross_pos[0, i//2]:]
        else:
            newpop[i, :] = pop[i, :]
            newpop[i+1, :] = pop[i+1, :]
    return newpop
```

5）变异

设定变异概率为 $p_m = 0.1$，随机生成 40 个[0, 1]内的随机值与变异概率比较，变异概率大于随机值才进行变异，并随机生成变异位置，代码如下：

```
def variation(pop, pm):
    n, l = pop.shape
    newpop = pop.copy()
    variation_prob = random.random(size = (1, n))
    variation_pos = random.randint(0, l, size = (1, n))
    for i in range(n):
        if variation_prob[0, i] < pm:
            newpop[i, variation_pos[0, i]] = 1 - pop[i, variation_pos[0, i]]
    return newpop
```

6）终止

终止条件设定为进化代数为 $N=1000$，并在每一代计算最优染色体对应的 x 值和最优适应值，程序如下：

```python
def best(pop, fit):
    n, l = pop.shape
    index = argmax(fit)
    best_pop = zeros((1, l))
    best_pop[0, :] = pop[index, :]
    best_x = decode(best_pop)
    return best_x, fit[index]
```

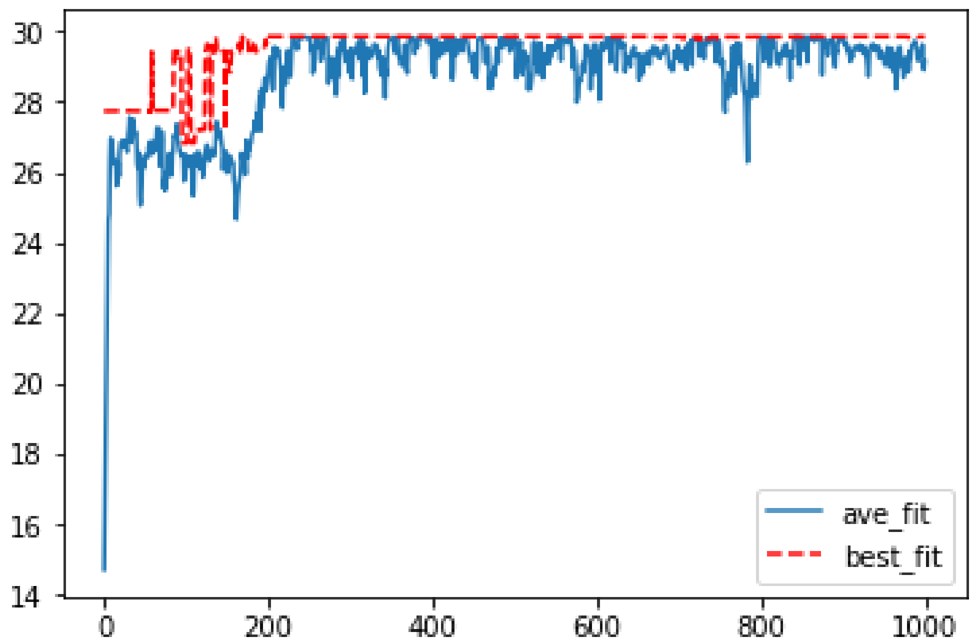
7）进化过程

进化过程依次进行选择、交叉、变异，程序如下：

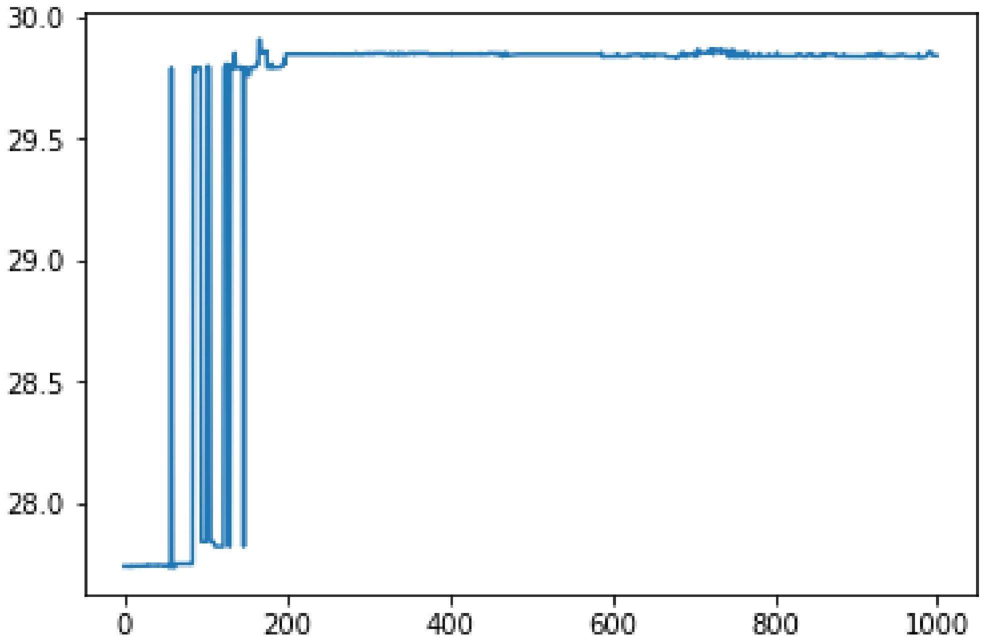
```python
for i in range(N):
    val = func(pop)
    fit = fitval(val)
    best_x[0, i], best_fit[0, i] = best(pop, fit)
    ave_fit[0, i] = sum(fit)/chromnum
    pop = select(pop, fit)
    pop = cross(pop, pc)
    pop = variation(pop, pm)
```

8）结果

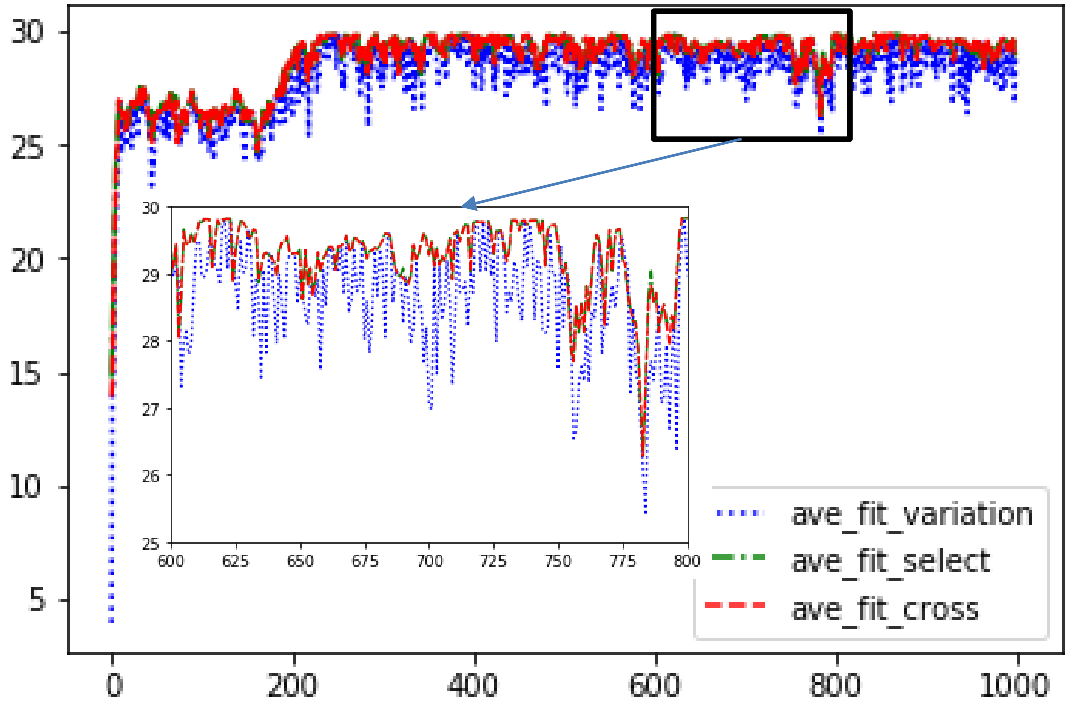
进化过程中最优适应值和平均适应值如下图所示：



x 的搜索过程如下图所示，最优解为 29.84305，对应的最优解为 29.84204：



进一步，我想看一下选择、交叉和变异对种群多样性的影响，因此做出三种操作后种群平均适应值如下图所示。从图中可以看出，交叉概率为 $p_c = 0.8$，交叉之后种群适应值改变较小，说明所产生的种群多样性较少；变异概率为 $p_m = 0.1$，变异之后种群适应值变化较大，说明所产生的种群多样性较大，但平均适应值也降低了；选择是适者生存的过程，大大减小了变异中产生的种群多样性，但同时也提高了种群的平均适应值。



30
25
20
15
10
5
0
200
400
600
800
1000
600
625
650
675
700
725
750
775
800
26
27
28
29
ave_fit_variation
ave_fit_select
ave_fit_cross