算法细节：

首先我们有一个GA类，这个类里面有7个成员变量：  
a：一个浮点数，是fitness计算时用到的（aβ\*runtime + (1-a)price）

fr：failure rate

avg\_tm\_dict: 是一个dict类型，包含了所有的test以及对应的所有信息（key=(classname, methodname); value=[throttConf, outerRound, avgTime, failureRate, price]）。

setup\_tm\_dict：记录了运行该project，每个配置下需要的setup time。

pop\_size: popularity，为100

gene\_length: 机器数量

max\_iter: 迭代次数，为100

之后我们会调用init\_pop()方法初始化popularity，首先先将所有机器设置同一种配置的情况添加进去，再随机生成剩下的配置信息。之后调用maintain\_memo(machine\_list)将每个基因传进去。我维护了一个字典memo，这个字典的键为machine list(转化成tup类型)，在maintain\_memo()这个方法中，我会先检查是否用算法计算过在此配置情况下的最优解，如果已经计算过，就直接返回memo里的数据，反之，就会调用get\_alloc()方法进行计算。  
  
get\_alloc()需要传入5个参数：  
a, machine\_list(对应的配置信息)，fr, 以及avg\_tm\_dict和setup\_tm\_dict用于计算。get\_alloc()函数里首先会用random.random()生成一个0-1的随机数，当这个随机数小于等于a时，我们会调用scheduled\_algorithm()函数，反之，调用price\_priority\_algorithm()函数。这两个函数的传入参数和get\_alloc()一致。

scheduled\_algorithm()是一个贪心的最小运行时间算法。维护一个变量mach\_time\_dict, 用于记录每个机器目前分配的test的运行时间（setup time已经提前记录好）。遍历每一个test，在满足fr的前提下，将test分配在让并行运行时间最短的配置的机器上运行。由此可以得到test的分配情况，以及对应的并行时间，price等信息。

price\_priority\_algorithm()函数，是遍历test，在满足fr的前提下，选择将test分配到使得运行成本最低的配置的机器上。由此可以得到test的分配情况，以及对应的并行时间，price等信息。

得到信息之后可以利用a值计算fitness，得到相应的score值。由此完成了popularity的初始化。之后对score进行排序，由于score越小，machine list的设置越优，我们选取前20%作为新一代popularity的parents。然后进行crossover()，对parents进行交叉变异生成children。之后再对children进行突变，遍历children，每次由random.random()生成一个随机数，当随机数小于1/gene\_length时，就对当前的gene进行随机位置的突变（将某个machine的配置替换）。上述每生成一个gene都会调用maintain\_memo(machine\_list)。

就这样迭代100次，取最后一代popularity的score最小的machine list, 作为最后的最优结果。