覆盖表生成的遗传算法配置参数优化

梁亚澜 聂长海
(南京大学计算机软件新技术国家重点实验室 南京 210093)

摘 要 覆盖表生成是组合测试的关键问题,很多数学方法、贪心算法以及演化搜索方法等被应用于生成各种覆盖表.针对演化搜索方法的性能受到方法本身配置参数影响很大这一实际问题,文中以二维覆盖表生成为实例,系统地对典型的演化搜索方法——遗传算法的种群规模、进化代数、交叉概率、变异概率以及遗传算法的变种算法等因素进行探索,设计了 pair-wise 法、Base choice 法和爬山法 3 条实验路线探索遗传算法的这些配置参数及其相互作用对算法生成二维覆盖表效果的影响,并回答两个问题:对于特定二维覆盖表生成问题,是否存在遗传算法的最优参数配置;对于一般的二维覆盖表生成问题,是否存在通用的遗传算法最优参数配置.

关键词 二维覆盖表;遗传算法;配置参数优化;组合测试;测试用例生成
中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2012.01522

The Optimization of Configurable Genetic Algorithm for Covering Arrays Generation

LIANG Ya-Lan NIE Chang-Hai
(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093)

Abstract Covering array generation is one of the key issues in combinatorial testing. Many mathematical methods, greedy algorithms and evolutionary search methods have been applied in this field. Since the performance of evolutionary search methods is significantly impacted by their configurable parameters, we take genetic algorithm, one of the typical evolutionary search methods, as an example to discuss the different influences of its five configurable parameters (population size, evolution generation, crossover probability, mutation probability, variants of the algorithm) on the performance of 2-way covering array generation. Meanwhile we design three classes of experiments to systemically analyze the influences of each of the configurable parameters and the interactions among them. Our contributions are to answer the following questions: whether there exists an optimal configuration of genetic algorithm for a particular 2-way covering array generation and whether there exists a common optimal configuration for all 2-way covering arrays generation.

Keywords 2-way covering array; genetic algorithm; optimal configuration; combinatorial testing; test case generation

1 引 言

软件作为一个复杂的逻辑体,其最终的运行效果受到软件自身及外部许多因素的影响.如何以尽可能小的代价检测出软件系统中各种因素及其相互作用所引发的各类故障是一个非常重要的问题.

组合测试方法通过生成少量测试用例[1],以较

收稿日期:2011-01-06;最终修改稿收到日期:2011-06-14.本课题得到国家自然科学基金(60773104,61021062)、国家"八六三"高技术研究发展计划项目基金(2008AA01Z143)、江苏省自然科学基金(BK2010372)资助.**梁亚澜**,男,1988 年生,硕士研究生,主要从事软件测试领域的科研工作.**聂长海**(通信作者),男,1971 年生,教授,博士生导师,主要从事软件测试领域的科研和教学工作. E-mail: changhainie@nju.edu.cn.

小代价最大限度地对软件系统各因素间的相互作用关系进行覆盖，可以有效检测系统中各种因素相互作用所触发的软件故障. 根据 Kuhn 等人[2]的研究发现，测试二维交互关系能够找出 70％的错误，因此全盘考虑测试的成本、覆盖表的生成难度以及各种组合覆盖程度对覆盖表规模的影响等因素. 两两组合测试是人们最常使用的方法[1].

目前国内外对于二维覆盖表生成方法的研究已有不少成果，人们提出了很多数学方法[3]、贪心算法[4-6]以及演化搜索算法[7-8]等. 遗传算法是一种典型的演化搜索方法，该方法被广泛地应用于解决复杂的优化问题. 人们也已将其应用于覆盖表生成[9-11]，但现有的文献未能考虑到演化搜索方法的性能受到方法本身配置参数影响很大这一实际问题. 本文针对该问题，以二维覆盖表生成为实例，系统地探索遗传算法各配置参数及其相互作用对算法性能的影响.

本文重点关注于遗传算法生成覆盖表的效果与遗传算法本身配置参数变化的关系，从而更好地发掘遗传算法生成覆盖表的潜力. 我们通过实验对遗传算法的种群规模、进化机制、交叉概率、变异概率及其变种算法这 5 个因素进行取值组合，设计了 pair-wise 法、Base choice 法[12-13]和爬山法 3 条不同的实验路线，系统地探索遗传算法的 5 个因素及其相互作用对遗传算法覆盖表生成效果的影响程度和影响性质，并以生成覆盖表规模和消耗时间为分析依据寻找出最佳配置，为今后进一步研究覆盖表生成的遗传算法打下理论和实践基础.

本文第 2 节介绍相关研究，并指出存在的问题；第 3 节给出覆盖表的定义及相关概念；第 4 节对遗传算法 4 个参数配置及其变种算法的概念进行简要介绍；第 5 节是具体的实验设计；第 6 节分析实验数据并得出结论；第 7 节是总结和展望.

2　相关工作及存在的问题

Ghazi 等人[9]在 2003 年最先将遗传算法应用于覆盖表生成，他们通过实验得出在参数个数为 4，每个参数取值个数都为 3 的待测系统中，利用遗传算法可以得到包含 11 条测试用例的二维覆盖表. 文章只表明使用遗传算法生成覆盖表的可行性，实验只用了一个例子，实验设计过于简单且没有进行深入的研究.

Shiba 等人[10]在 2005 年介绍覆盖表生成的人工智能技术的文章中，对遗传算法如何确定染色体结构、适应度函数、选择交叉变异策略等进行了讨论，并对其中一组配置进行了实验，文章中并没有对遗传算法生成覆盖表的性能进行分析.

McCaffrey[11]在 2010 年提出遗传算法生成覆盖表的性能会受到其自身配置参数的影响，并提取出种群规模、染色体结构、进化选择策略等因素进行了讨论. 但是 McCaffrey 只通过实验验证了在其中一种参数配置下，遗传算法生成覆盖表的效果比较理想，并没有系统地探索遗传算法自身的这些因素及其相互作用对算法性能的影响特点和影响程度.

针对已有工作未能系统探索遗传算法生成覆盖表的性能，本文对遗传算法的配置参数进行了更为深入的探索，重点关注种群规模、进化代数、交叉概率、变异概率及遗传算法的变种算法这 5 个重要配置参数及其相互关系对遗传算法生成覆盖表性能的影响程度和影响性质，通过科学的实验设计，对这 5 个配置参数的取值组合进行优化以找出覆盖表生成遗传算法的最优参数配置，从而进一步发掘遗传算法在覆盖表生成方面的潜力.

3　覆盖表

软件质量受到其自身及外部许多因素的影响，针对这一实际问题，要求一个完备的检测方法必须能够检测出这些因素及其相互作用关系所引发的各类故障. 覆盖表，特别是二维覆盖表可以用最少的测试代价，系统地覆盖待测试系统中的各种参数组合. 以下我们先给出测试用例的定义，再通过一个例子介绍什么是覆盖表，然后给出覆盖表的一般定义.

定义 1.　设待测系统 SUT 有 n 个参数，每个参数有 $v_i(1\leq i\leq n)$ 个取值，形成集合 V_i. n 元组 $\langle a_1,a_2,\cdots,a_n\rangle$ 称为 SUT 的一个测试用例($a_i\in V_i$).

如表 1 所示，一个待测系统 SUT 存在 4 个可能影响其性能的参数分别为 C_1、C_2、C_3、C_4. 每个参数各有 3 个不同的取值{0、1、2}. 如果以穷举法对 4 个参数的所有取值组合进行检测，我们需要 $3^4=81$ 个测试用例才可以检测出各因素相互之间的关系所引发的故障，测试代价相对较大. 这个缺点在系统因素及其相互作用更复杂的待测软件中更为明显. 而且测试数据表明，并不是所有参数取值之间都存在相互作用，因此穷举法会产生大量冗余的测试用例.

表 1 SUT

C_1	C_2	C_3	C_4
0	0	0	0
1	1	1	1
2	2	2	2

组合测试方法所生成的测试用例集——覆盖表[1]，通过对软件系统各因素间的相互作用关系进行有选择的针对性覆盖，大大减少了冗余的测试用例，同时在保证测试效果的前提下有效地缩减了测试用例集的规模. 在表 1 的例子中，如需要覆盖任意两个参数间的相互关系，则只需要表 2 中的 9 条测试用例，从该表中可以检查它覆盖了任意两个参数之间的 9 个组合，在实际检测中可以检测出 SUT 中所有因为两两参数之间的相互作用而引发的软件故障.

表 2 SUT 的覆盖表

C_1	C_2	C_3	C_4
0	0	0	0
1	0	2	1
2	1	2	0
2	0	1	2
1	1	0	2
0	1	1	1
2	2	0	1
1	2	1	0
0	2	2	2

覆盖表作为组合测试的测试用例集，其定义如下.

定义 2. 假设影响一个待测系统的参数共 n 个，每个参数有 v 种可能取值. 则覆盖表 $\mathrm{CA}(N;t,n,v)$ 是一个 $N\times n$ 的数组，每行对应着一条测试用例，每列对应一个参数的所有取值. 任意 t 个参数形成的子数组 $N\times t$ 覆盖了该 t 个参数所有取值的组合，即每个 t 元组至少出现 1 次. 其中 t 为组合覆盖测试的强度，本文中提到的两两组合测试指的是 $t=2$ 的情况，生成的即为二维覆盖表[1].

在真实的软件系统中，影响系统的参数取值数目可能不尽相同. 于是人们将覆盖表扩展，形成了混合覆盖表 $\mathrm{MCA}(N;t,n,(v_1,v_2,\cdots,v_n))$. 表中第 i 列对应第 i 个参数，该参数取值数目记为 v_i，任意 t 个参数形成的 $N\times t$ 的子数组中包含了该 t 个参数的所有 t 元组. 当 $v_1=v_2=\cdots=v_n=v$ 时，$\mathrm{MCA}(N;t,n,(v_1,v_2,\cdots,v_n))$ 即为 $\mathrm{CA}(N;t,n,v)$[1].

覆盖表的生成是组合测试的关键环节，目前国内外对组合测试的很多研究主要集中在该领域. 除了数学方法[3]、启发式贪心算法[4-6]等，演化搜索算法是其中一类重要的方法，这一类方法利用模拟退火[7]、遗传算法[9-11]、蚁群算法和粒子群算法等演化计算技术生成覆盖表，是已有贪心算法和数学方法的重要补充，具有很强的灵活性和有效性，但其性能受其自身配置参数影响很大，本文针对这一实际问题，以遗传算法这一典型的演化搜索算法为研究对象，系统地探索算法中各配置参数对算法生成覆盖表的最终效果的影响性质和影响程度.

4 覆盖表生成的遗传算法

遗传算法[9-11,14]是一种通过模拟自然世界生物进化过程探索问题最优解的演化搜索算法，已被广泛地用于复杂的优化问题. 在遗传算法中，问题的一组候选解以染色体的形式构成算法的初始种群，算法对每一代种群中的染色体进行交叉变异，然后通过适应度函数值这一评价标准淘汰适应值较差的染色体，保留适应值较好染色体进入下一代，直到某一代种群中出现适应值满足问题要求的染色体，该染色体即为问题的最优解.

在本文中，我们采用待测系统 SUT 中参数 C_1，$C_2,\cdots,C_n$ 的一个取值组合 $\langle a_1,a_2,\cdots,a_n\rangle$ 作为染色体，即候选测试用例，以逐条生成测试用例的方式构造二维覆盖表. 算法流程图如图 1 所示，其具体步骤如下：

1. 初始化测试用例集 TG（TG 可设空集，也可放入所需的初始用例）；初始化待测系统 SUT 中需要被覆盖的二元组集合 S；随机生成初始的种群，种群规模为 m；初始化进化代数 T；初始化种群个体的适应值数组 fitness.

2. 计算种群中各染色体的适应度函数值，适应值即为该染色体所包含的未被 TG 覆盖的二元组合对个数.

3. 对种群中的染色体进行选择、交叉（交叉概率为 Pc）、变异（变异概率为 Pm）以产生新的种群. 对新种群中的个体进行适应值评价，若新的种群中存在最优染色体 c，则将其作为覆盖表中的一条新的用例，跳至步 5；否则重复该步过程，直到迭代次数达到 T.

4. 在当前种群中选择适应值最高的染色体 c，作为覆盖表中的一条新的用例.

5. 将 c 加入 TG 中，并从 S 中删去 c 所包含的二元组合对，若 S 为空集，即当前 TG 中的用例包含了所有二元组合对，则算法结束，TG 即为所求覆盖表；否则，跳至步 3.

由于遗传算法的效果受其自身配置参数的影响，为了进一步提高算法生成覆盖表的性能，以下我们从中提取出种群规模 m、进化代数 T、交叉概率 Pc、变异概率 Pm 以及遗传算法变种算法这 5 个对其性能影响较大的重要因素进行进一步讨论.

```
/*第1步*/
Initialize (TG)                                    //初始化测试用例TG(TG可以设空集，也可放入一些指定的初始实例)
Initialize S;                                      //初始化未被TG覆盖的二元组集合S
Initialize (group[m][n]);                          //初始化种群，种群规模为m，每条染色体为一条测试用例，n为测试用例参数个数
Initialize (T)                                     //初始化最终进化次数T
Initialize (fitnesss [m])                          //初始化种群个体的适应值数组
/*第2步*/
Evaluate (group [m][n], fitness [m]);              //对种群适应值进行评价
while (There exist uncovered pairs in S)
    {
    /*第3步*/
    t=0;                                           //重置当前进化次数t
    while (t<T)                                    //当前进化代数t没有达到T
    {
        Select (temp [m][n], group [m][n]);        //利用轮盘赌算法从group中选取染色体进入temp
        Crossover (temp[m][n]);                    //对temp中的染色体进行交叉操作
        Mutate (temp [m][n]);                      //对temp中的染色体进行变异操作
        Transit (group [m][n], temp[m][n]);        //将temp中的染色体放回group中
        Evaluate (group[m][n], fitness[m]);        //对种群适应值进行评价
        if (There exists a best chromosome "c" in group [m][n])
        {
            break;
        }
        t++;
    }
    /*第4步*/
    if (There exists no best chromosome "c" in group [m][n])
    {
            Choose a chromosome "c" in group [m][n] possessing the highest fitness;
    }
    /*第5步*/
    TG=TGAc                                        //将最优解放入测试用例集TG中
    S=S-{c};                                       //从S中删除c所包含的二元组合对
}
```

图 1　覆盖表生成的遗传算法流程图

4.1　种群规模 m

遗传算法中种群是由一定数量的染色体组成，种群中染色体的数量称为种群规模 m. 通常种群规模太小则不能提供足够的采样点，造成算法过早收敛；种群太大虽然可以增加优化信息，阻止早熟收敛的发生，但无疑会增加计算量，造成收敛时间太长.

为了准确找到最合适的种群规模取值，在正式实验前，我们对不同种群规模下遗传算法生成覆盖表的性能进行了初步的测试工作，结果显示：种群规模在6000以上时，算法生成覆盖表的规模下降程度不明显，算法速度却大大降低；种群规模在100以下时，算法生成覆盖表的规模有增加趋势且性能很不稳定.

综合考虑实验的准确度和复杂度等因素，本文种群规模 m 的取值集合为{100,2100,4100,6100}，该取值集合既保证实验能够击中最合适的种群规模取值，又保证实验复杂度在可操作范围内.

4.2　进化代数 T

通过进化代数 T 限定算法每轮的迭代次数，迭代次数达到进化代数 T 则算法终止. 与种群规模取值相似，终止进化代数过小，则生成的解无法接近最优解，过大，则运行消耗代价过大. 因此，在实际问题解决时，需要权衡各方面因素，力图找到一个平衡点.

在正式实验前，根据对不同进化代数取值下遗传算法生成覆盖表性能的初步测试我们发现：进化代数在100以下时，算法生成覆盖表的规模有增大趋势；进化代数在1000以上时，算法生成覆盖表的规模没有明显减小趋势且算法的速度大大降低. 综合考虑实验的准确度和复杂度等因素，本文 T 的取值集合为{100,600,1100}.

4.3　交叉概率 Pc

对种群中的染色体进行交叉操作以产生新的染色体，实质上是对问题解空间的广度搜索. 交叉概率太大，则种群中个体更新很快，容易破坏已有的高适应值的个体；概率太小，则交叉操作很少进行，会使搜索停滞不前，造成算法的不收敛.

通过测试工作，我们发现交叉概率 Pc 对遗传算法性能的影响没有明显的趋向性，为了保证实验结果的准确，对其采取在合法取值空间中均匀取值的方法，Pc 的取值集合为{1,0.8,0.6,0.4,0.2}.

4.4　变异概率 Pm

变异操作是对种群模式的扰动，有利于增加种群的多样性，实质上是对问题解空间的深度搜索. 同样，变异概率太小则很难产生新模式；变异概率太大则会使遗传算法成为随机搜索算法，失去遗传算法的特性和优点.

与交叉概率 Pc 相同，变异概率 Pm 对算法性能的影响也没有明显的趋向性，为了保证能够击中最合适的 Pm 取值，本文 Pm 的取值集合为{1,0.8,0.6,0.4,0.2}.

4.5　遗传算法的变种算法

本文在对传统遗传算法4个配置参数进行探索的同时，对遗传算法进行一定程度的改动，以发掘遗

传算法生成覆盖表的潜力.本文所探索的算法分别为:

GA:即遗传算法.GA 采用逐条生成测试用例的方法;在种群进化过程中,父代的最优个体 c 直接进入子代,其余染色体的选择策略 select 为轮盘赌选择法;适应值 f 为该条染色体所包含的未被覆盖对个数;交叉策略为多点交叉;变异策略是单点变异.

GA-:在种群进化过程中,GA-将 GA 中适应值 f 改为该条染色体所包含的已被覆盖对的个数,使 GA 的优胜劣汰制变为优汰劣胜制(注:优汰劣胜仅为种群进化中 select 的标准,而个体进入覆盖表的标准仍是选取包含未被覆盖对个数最多的个体).

GAr:在种群进化过程中,GAr 将 GA 染色体选择策略 select 改为随机选择,即父代中的个体随机选择进入子代.

GA climb:在 GA 的基础上采用爬山法对每一代种群中最优个体进行处理:GA climb 算法中每一代的最优个体 c 只可能被当前适应值更高的个体替换,否则直接进入下一代而不参加其它操作.算法保证了最优个体的优良特性能够在不被交叉变异破坏的基础上得到不断优化.

GA- climb:在 GA-的基础上改变对每一代种群中最优个体的处理,处理过程同 GA climb.

GAr climb:在 GAr 的基础上改变对每一代种群中最优个体的处理,处理过程同 GA climb.

5 实验设计

实验表明,覆盖表生成的遗传算法的效果会受到其自身 5 个因素(配置参数)的影响.如何确定相应的配置参数取值以求算法达到最好效果,即如何对有序集〈算法,种群规模,进化次数,交叉概率,变异概率〉进行取值,这是本文需要解决的问题.本文通过对形如〈GA-,2100,600,0.21,0.41〉的不同参数配置下的遗传算法的性能进行测试和比较,主要回答以下问题:(1)遗传算法这 5 个因素对于算法的性能的影响程度和性质如何.(2)是否存在一组最优参数配置,使得二维覆盖表的遗传算法对于某个特定问题,总是能发挥较优的性能.(3)该组最优参数配置是否具有通用性,即对于其它覆盖表生成问题也能产生近优解.

本文对不同参数配置下遗传算法生成覆盖表性能的评判标准为:首先考虑算法生成覆盖表规模,生成覆盖表规模最小的为最优参数配置;若不同参数配置下生成覆盖表规模相同,则选取生成覆盖表过程消耗时间最少的为最优参数配置.

遗传算法 5 个配置参数在相应取值范围内产生的值组合(即参数配置)规模十分庞大,在表 3 中的参数取值情况下,共产生 $6\times4\times3\times5\times5=1800$ 种不同的参数配置,利用每一种配置对应的遗传算法进行覆盖表生成并选优是不可行的.

表 3 遗传算法 5 个配置参数的取值集合

算法	种群规模	进化代数	交叉概率	变异概率
GA	100	100	1	1
GA-	2100	600	0.8	0.8
GAr	4100	1100	0.6	0.6
GA climb	6100		0.4	0.4
GA- climb			0.2	0.2
GAr climb				

为了更高效地解决问题,本实验采用 3 条实验路线对最优参数配置进行搜索,利用 3 个不同的参数配置集,在确保击中最优参数配置的前提下大大减少了所需测试的配置个数(为了在配置集中方便记录各参数的取值,本文将其对应为有序的自然数,如表 3 中"种群规模"中有 4 个不同取值,我们将其与 0,1,2,3 这 4 个自然数相对应):

(1) pair-wise 参数配置集

假设表 3 中各参数两两之间存在相互作用,我们对其进行二维组合覆盖,生成的二维覆盖表作为 pair-wise 配置集.为了概率更高地包含较优配置,本文没有采用规模最小的覆盖表,而是加入了适量冗余.如表 4 所示,该覆盖表共有 34 组不同的参数配置,这 34 条参数配置覆盖了所有参数之间的二维组合.比较该组参数配置集下遗传算法的待测实例覆盖表生成效果,从中得到最优参数配置 P_x($1\leq x\leq34$).

表 4 pair-wise 配置集

测试号	a_1	a_2	a_3	a_4	a_5	测试号	a_1	a_2	a_3	a_4	a_5
P_1	5	3	1	0	1	P_{18}	3	1	0	0	4
P_2	4	1	2	0	3	P_{19}	2	3	1	2	2
P_3	2	2	0	4	4	P_{20}	5	2	2	4	3
P_4	3	0	1	1	0	P_{21}	2	2	1	0	0
P_5	0	0	0	2	1	P_{22}	3	3	0	3	1
P_6	5	1	0	3	0	P_{23}	1	0	0	3	4
P_7	3	2	2	3	2	P_{24}	4	2	1	4	1
P_8	1	3	0	1	3	P_{25}	0	1	1	2	3
P_9	0	3	2	4	0	P_{26}	0	1	1	1	2
P_{10}	1	1	1	4	2	P_{27}	3	0	2	4	3
P_{11}	4	3	1	2	4	P_{28}	4	3	1	3	0
P_{12}	2	0	1	3	3	P_{29}	0	2	0	0	4
P_{13}	**5**	**0**	**2**	**1**	**4**	P_{30}	5	3	0	2	2
P_{14}	4	0	0	0	2	P_{31}	1	0	2	0	1
P_{15}	1	2	2	2	0	P_{32}	0	2	1	3	1
P_{16}	0	2	1	1	1	P_{33}	4	3	2	1	1
P_{17}	2	1	2	1	1	P_{34}	3	3	1	2	0

(2) Base choice 参数配置集

Base choice 是一种组合设计方法[12-13]. 该方法先选取一个参数配置作为基准参数配置 B(在本实验中该基准配置 B 即为 pair-wise 法所得最优参数配置 P_x),在此基础上改变其中某一个因素的取值并保持其余因素取值不变,从而产生一组参数配置集. 采用上述方法对遗传算法所有因素取值进行覆盖,得到的参数配置集即为 Base choice 参数配置集. 例如表 5 是以表 4 中的 P_{13} 为基准参数配置所对应的 Base choice 配置集.

表 5　Base choice 配置集

测试号	a_1	a_2	a_3	a_4	a_5	测试号	a_1	a_2	a_3	a_4	a_5
B_1	**5**	**0**	**2**	**1**	**4**	B_{11}	5	0	**1**	1	4
B_2	**0**	0	2	1	4	B_{12}	5	0	2	**0**	4
B_3	**1**	0	2	1	4	B_{13}	5	0	2	**2**	4
B_4	**2**	0	2	1	4	B_{14}	5	0	2	**3**	4
B_5	**3**	0	2	1	4	B_{15}	5	0	2	**4**	4
B_6	**4**	0	2	1	4	B_{16}	5	0	2	1	**0**
B_7	5	**1**	2	1	4	B_{17}	5	0	2	1	**1**
B_8	5	**2**	2	1	4	B_{18}	5	0	2	1	**2**
B_9	5	**3**	2	1	4	B_{19}	5	0	2	1	**3**
B_{10}	5	0	**0**	1	4						

(3) 爬山法参数配置集

爬山法首先选取一个参数配置作为基准参数配置 C_0,例如假设 C_0 为〈5,0,2,1,4〉(在本实验中该基准配置 C_0 即为 pair-wise 法所得最优参数配置 P_x),从遗传算法第 1 个配置参数变种算法开始,依次改变其在 C_0 中的取值并保持其余参数取值不变,从而产生 6 个参数配置集. 比较该组参数配置下遗传算法的覆盖表生成效果,选取生成效果最好的参数配置 C_1 作为新的选定配置,例如变种算法取值为 2 时效果最好,则 C_1=〈2,0,2,1,4〉. 然后在 C_1 的基础上改变遗传算法第 2 个配置参数种群规模的取值,得到 4 个参数配置集并产生 C_2,例如种群规模取值为 1 时生成效果最好,则 C_2=〈2,1,2,1,4〉. 依此类推,直到改变第 5 个配置参数的取值得到 C_5,C_5 即为爬山法所得的最优参数配置. 上述过程说明爬山法参数配置集是在实验中一步步动态生成的.

为满足实验需求,我们设计了可配置的覆盖表生成的遗传算法工具 COGA(Configurable and Optimized Genetic Algorithm). COGA 可自动以上文所描述的 pair-wise 法、Base choice 法和爬山法这 3 条实验路线对遗传算法的 5 个因素进行取值集合空间的搜索和取值组合的优化,得到待测实例的覆盖表生成的最优参数配置. 其输入为:(1) 待测实例描述,如 3^{13}(13 个参数,每个参数都有 3 个取值);(2) 遗传算法 5 个配置参数各自的参数取值集合. 输出为:(1) 运行 3 条实验路线过程中覆盖表的生成规模和消耗时间等数据记录;(2) 能为待测实例生成最优覆盖表的遗传算法配置.

6　实验结果及分析

根据实验设计我们对不同规模的待测实例进行覆盖表生成,得到各待测实例在不同配置集下遗传算法生成的覆盖表规模和消耗时间. 考虑到每一个待测实例的参数配置优化过程都需要对 34+23+23=80 个不同参数配置下的待测实例进行覆盖表生成实验,且为了保证数据的准确性,本文对 Base choice 配置集和爬山配置集中的每一个配置都进行 5 次实验并取最优解为生成结果,消耗时间较多,所以只选取 15 个具有代表性的待测实例进行实验(在 24 GB 内存,4 核 CPU(Intel(R)Xeon(R) E7450,主频 2.40 GHz)的刀片机远程终端上运行,实验周期为 1 个月),其中待测实例的选取兼顾下列几点:(1) 选取 4^{10}、3^{13} 和 6^4 等较为常见的待测实例. (2) 有选择地加入覆盖表生成规模和消耗时间相差较大的待测实例,以增加实验用例选择空间的广度. (3) 包含参数取值不一致的待测实例,以保证实验结论更加准确可靠.

下面我们分别对 3 个配置集下的实验数据进行整理分析.

6.1　pair-wise 实验

实验所得覆盖表的生成规模和消耗时间分别如表 6、表 7 所示,表中第 1 列列出 pair-wise 配置集中 34 个不同配置{$P1,P2,\cdots,P34$};第 1 行是各个待测实例,例如 4^{10} 表示该待测实例有 10 个参数,每个参数取值个数均为 4;$7^6\,6^7\,5^6$ 表示该待测实例有 19 个参数,其中 6 个参数有 7 个取值,7 个参数有 6 个取值,还有 6 个参数有 5 个取值.

通过不同配置下的覆盖表生成结果对比可以发现,利用遗传算法生成覆盖表的最终效果受算法自身配置参数的影响很大,例如在实例 6^{20} 中,不同参数配置下的覆盖表生成规模最多时相差 42 个,差距十分明显. 综合覆盖表的生成规模和消耗时间这两个标准,我们得到 pair-wise 配置集中各待测实例的最优参数配置 P_x 并在表中用加粗标记.

表 6 pair-wise 配置集生成覆盖表的规模

	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
P_1	30	19	62	39	106	23	87	46	106	173	92	**71**	39	45	23
P_2	30	20	61	37	106	22	84	45	102	167	89	73	39	44	21
P_3	31	20	70	44	125	24	101	53	121	207	103	80	41	**42**	22
P_4	31	19	66	41	116	24	96	50	113	190	100	76	39	43	**20**
P_5	36	21	79	49	151	28	120	60	139	245	121	95	45	45	25
P_6	30	19	64	42	113	23	93	48	112	184	96	75	39	44	21
P_7	31	19	60	38	104	21	86	45	104	170	91	74	40	44	22
P_8	32	19	71	44	124	24	102	52	121	205	105	79	40	45	22
P_9	31	20	70	44	125	24	101	52	122	205	104	78	44	43	25
P_{10}	32	20	72	45	128	25	105	54	126	214	108	81	42	44	22
P_{11}	30	19	62	37	102	22	79	**41**	96	162	85	74	40	43	24
P_{12}	37	23	82	51	148	27	121	61	141	246	126	92	47	47	25
P_{13}	30	**18**	64	**35**	**100**	23	**77**	42	**90**	**157**	**82**	75	39	45	21
P_{14}	31	20	70	43	120	24	99	51	118	198	100	77	39	45	23
P_{15}	32	20	72	44	125	23	101	52	123	207	107	79	42	44	23
P_{16}	30	20	70	44	124	24	100	53	122	204	106	79	41	42	21
P_{17}	32	19	72	45	130	25	105	55	124	213	110	81	41	43	24
P_{18}	30	19	60	38	103	22	84	45	101	167	90	75	41	43	23
P_{19}	33	20	69	43	123	25	101	53	121	205	104	79	40	45	23
P_{20}	29	19	60	37	102	22	84	44	103	165	87	74	**37**	42	23
P_{21}	31	20	70	44	126	24	102	52	121	208	105	80	42	43	22
P_{22}	30	19	63	39	107	22	90	48	109	178	93	75	38	45	22
P_{23}	37	22	86	49	150	28	119	58	140	250	122	94	44	47	24
P_{24}	29	19	62	39	105	22	88	47	107	174	93	74	38	42	22
P_{25}	31	20	68	44	122	24	100	51	120	202	102	80	42	43	22
P_{26}	31	21	71	44	127	24	102	52	122	206	105	79	42	44	22
P_{27}	**28**	19	**59**	37	101	**21**	82	44	99	162	86	74	39	44	22
P_{28}	31	19	62	40	111	22	86	48	105	172	91	72	40	44	23
P_{29}	29	19	64	42	112	24	93	50	114	188	98	77	39	43	22
P_{30}	29	21	64	41	109	22	91	47	109	178	93	73	39	43	21
P_{31}	38	23	82	52	153	28	120	60	142	252	122	95	48	48	26
P_{32}	32	21	71	44	123	24	100	52	121	205	104	80	41	44	24
P_{33}	30	20	63	38	105	22	86	47	105	173	91	73	39	42	23
P_{34}	30	19	61	40	111	22	88	47	106	174	89	75	40	44	22

表 7 pair-wise 配置集生成覆盖表消耗时间

	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
P_1	124.7	126.9	255.11	576.346	395.384	343.17	1303.92	1836.19	4256.82	785.918	3756	**218.42**	1224.05	38.657	151.883
P_2	196.3	159.4	388.58	431.14	682.645	251.43	1014.19	965.365	2247.35	1147.47	2533	796.65	876.616	139.028	201.132
P_3	12.3	12.45	27.285	59.717	45.849	34.663	139.309	206.733	479.157	89.622	460.4	36.036	140.635	**2.917**	12.636
P_4	1.96	1.716	4.477	7.113	7.472	4.04	17.191	16.942	39.359	13.713	18.89	4.244	7.02	0.811	**1.825**
P_5	0.406	0.328	0.936	1.467	1.732	0.827	3.697	3.447	8.175	3.213	3.93	0.811	1.358	0.14	0.344
P_6	6.598	5.928	13.962	27.846	22.885	15.554	63.758	74.584	171.835	44.024	186.8	17.113	58.781	2.091	6.131
P_7	696.1	485.6	1327.9	1341.4	2192.5	716.63	3033.81	2817.99	6692.29	3760.31	6569	1884.06	1808.05	575.207	703.798
P_8	146.1	97.53	322.81	292.798	548.734	163.957	692.926	558.983	1311.05	949.984	1231	675.906	307.042	124.91	145.128
P_9	1489	1123	3422.26	3282.93	6018.41	1818.74	7675.65	6090.19	14289.4	10288.7	13278	7073.29	3733.32	1218.49	1799.47
P_{10}	121.4	88.69	266.72	305.04	465.10	170.07	713.938	657.934	1525.18	840.705	1699	539.062	350.222	119.232	118.296
P_{11}	801.1	563.4	1586.3	1462.6	2565.8	852.28	3087.67	**2516.31**	6030.95	4446.86	5770	3602.51	1762.69	734.421	927.551
P_{12}	2.137	1.856	4.852	8.268	8.3	4.259	20.047	19.547	46.27	17.332	21.92	6.864	7.098	0.687	2.199
P_{13}	2.855	**2.43**	6.208	**9.579**	**8.93**	5.99	**22.4**	23.26	**53.414**	**17.77**	**23.45**	8.253	10.046	1.154	3.261
P_{14}	0.343	0.312	0.827	1.295	1.342	0.718	3.042	2.995	6.927	2.714	3.34	1.466	1.077	0.234	0.608
P_{15}	695.6	511.7	1564.6	1604.0	2714.6	800.39	3662.59	3314.43	7847.8	9516.42	7005	3282.45	1834.4	652.615	740.755
P_{16}	370.7	281.5	819.29	881.27	1491.9	480.359	1993.18	1850.17	4284.3	5367.04	3526	1768.58	982.853	230.741	366.743
P_{17}	79.61	68.48	178.12	340.50	303.51	178.309	802.344	927.036	2177.24	1590.6	1714	285.169	369.083	21.996	88.094
P_{18}	18.17	13.38	35.522	40.763	59.202	22.183	89.763	87.719	198.823	223.908	180.9	77.673	57.346	14.602	19.687
P_{19}	140.7	133.1	278.42	671.83	465.35	384.95	1505.13	2115.12	4855.7	4060.78	3386	532.322	815.682	45.302	150.432
P_{20}	136.6	133.0	268.63	578.47	426.79	338.01	1307.88	1914.07	4636.33	2481.77	3510	519.515	**620.95**	37.846	156.952
P_{21}	84.396	80.48	182.52	396.96	308.69	220.68	904.743	1219.8	2986.44	2316.97	2333	336.588	501.044	21.201	86.394
P_{22}	136.4	93.88	273.09	266.43	463.05	147.88	607.062	515.365	1184.25	1263.51	951	578.545	293.594	134.925	147.421
P_{23}	0.375	0.328	0.92	1.389	1.513	0.78	3.339	3.089	7.707	3.089	3.59	0.749	1.124	0.094	0.483
P_{24}	338.9	257.0	721.11	774.65	1197.3	421.40	1712.39	1671.07	3744.48	3971.43	3159	939.906	906.459	284.624	417.49
P_{25}	116.5	85.50	246.84	292.05	437.94	149.59	665.375	587.781	1403.49	1406.24	1265	301.191	346.977	95.909	127.531
P_{26}	118.2	92.22	264.41	299.68	464.32	158.80	693.456	604.192	1503.99	1471.95	1303	305.512	343.109	125.409	133.911
P_{27}	**3.09**	2.995	**6.93**	11.23	11.31	**6.02**	26.146	26.723	61.963	22.776	28.1	6.287	11.388	1.872	4.82
P_{28}	849.1	557.5	1618.7	1628.9	2895.02	884.35	3437.12	3066.03	6772.35	7634.66	5555	2023.26	1846.54	838.849	966.489
P_{29}	57.77	42.63	124.94	134.85	213.54	77.064	292.767	281.145	664.408	620.151	651.9	165.283	153.255	53.492	66.347
P_{30}	20.12	22.68	42.557	107	68.313	58.329	223.83	309.459	732.799	357.086	648.4	48.298	133.319	6.584	24.305
P_{31}	4.79	3.963	10.609	17.176	18.86	9.079	40.201	38.158	91.058	38.158	44.1	9.017	14.742	1.794	5.476
P_{32}	382.9	291.2	850.99	882.16	1454.7	466.21	1954.01	1816.52	4260.86	4007.92	3378	1066.63	991.37	355.323	446.693
P_{33}	1496.	1094.	3023.1	2786.1	4958.5	1579.9	6372.31	5476.74	12436.2	12170	9921	3844.5	3310.33	1256.71	1701.78
P_{34}	816.2	562.9	1621.6	1632.3	2932.8	870.30	3540.49	2962.76	6801.13	6821.83	5463	2147.67	1859.83	834.793	866.039

表中数据显示，所有待测实例除了 6^4，其余最优参数配置均选择 climb 算法，由此得出初步结论：利用遗传算法生成覆盖表时，算法选择以 climb 算法为优.

Pair-wise 实验对有限的 34 个不同参数配置下的待测实例进行覆盖表生成，可以得出不同的参数配置对覆盖表生成效果影响很大这一结论. 而这些影响具体是由哪一组参数的两两组合所引起，需进行更有针对性的实验才能确定. 目前对于如何确定参数两两交互作用对算法性能的影响这一问题，仅有 Bryce 等[6]基于贪心算法 6 个决策点的两两交互作用的相关研究，其实验过程和文章篇幅都较长. 所以要验证本文中遗传算法的参数交互作用，需以 Bryce 的实验设计为参考，另文详细讨论.

Pair-wise 实验主要考虑了遗传算法各配置参数之间的二维组合关系对算法效果的影响，并从中得到各待测实例的覆盖表生成的最优参数配置 P_x，但实验没有对遗传算法的 5 个配置参数进行单独地深入探索，因此无法准确评价这 5 个配置参数各自对算法效果的影响性质和影响程度，这在很大程度上影响了实验结论的完备性. 为了解决这一问题，我们在 pair-wise 实验的基础上设计了 Base choice 实验. Base choice 实验以 pair-wise 实验所得的配置 P_x 为基准参数配置，通过改变 P_x 中的某一个参数取值得到参数配置集并进行实验，更注重于发掘各配置参数自身对算法效果的影响.

6.2　Base choice 实验

实验以 pair-wise 法所得的最优参数配置 P_x 作为基准参数配置. 由于在 pair-wise 实验中得到的不同待测实例的覆盖表生成的最优参数配置不一致（如表 6、表 7 所示），相应的 Base choice 实验配置集也不相同. 考虑到演化搜索算法的不确定性，本文对每一个配置都进行了 5 次测试并取最优解为生成结果. 下面我们分别探索各个集合中每个配置参数对遗传算法生成覆盖表效果的影响性质和影响程度，找出该配置参数的最优取值.

6.2.1　遗传算法的变种算法

本文中遗传算法共有 6 个不同的变种算法，即每个待测实例对应 6 个不同的参数配置. 这 6 个配置下各覆盖表生成规模和消耗时间如表 8、表 9 所示（5 次测试取最优结果），由于基准参数配置 P_x 不一致，本文以算法选择为标准对生成结果进行统一排序，其中 P_x 用加粗标记.

表 8　Base choice/爬山实验中选择不同算法时覆盖表的生成规模

算法	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
0	33#	19#	76#	44#	126#	26#	104#	48#	126#	212#	107#	78	38#	44	24
1	36	22	81	50	147	27	115	52	139	249	121	77	40	45	24
2	35	21	82	49	147	27	116	52	136	245	121	76#	41	**42#**	24#
3	**28**	19	**59***	36	99*	**21**	74*	41	89*	154*	82	74	38	42	**20**
4	28	19	60	35	100	21*	77	**41**	90	160	83	70*	37	43	20
5	28*	**17***	61	**35***	**100**	21	**77**	41*	**90**	**157**	**82***	**71**	**36***	42*	20*

表 9　Base choice/爬山实验中选择不同算法时覆盖表的消耗时间

算法	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
0	3.9#	2.9#	9.19#	13.2#	13.76#	7.94#	32.18#	2995#	76.3#	27#	32.7#	2225	1588.6#	56.1	2.93
1	4.24	3.50	9.68	15.09	16.19	8.22	35.21	3322	82.46	33.20	37.47	2204	1698.8	54.7	2.18
2	3.68	2.9	8.83	13.79	14.15	7.83	33.38	1965	79.2	28.74	34.88	225.3#	869.1	**2.9#**	1.69#
3	**3.09**	2.8	**6.93***	10.42	10.25*	**6.02**	22.93*	2547	53.3*	19.8*	25.02	2141	1619.8	44.9	**1.83**
4	3.14	2.7	7.01	10.16	10.39	5.67*	23.29	**2516**	54.87	19.97	25.11	2034*	1562.2	45.4	1.93
5	2.9*	**2.3***	6.48	**9.58***	**8.93**	5.79	**22.4**	1524*	**52.87**	**17.8**	**23.5***	**218.42**	**627.7***	2.7*	1.78*

为了准确分析表中的数据，本文用“*”标记出各待测实例的覆盖表生成的最优算法选择，并将算法进行如下划分：$A=\{$GA, GA-, GAr$\}$；$B=\{$GA climb, GA- climb, GAr climb$\}$. 通过比较 A，B 集合的数据可以发现表 8 和表 9 中“*”都集中于 B 集合，即无论是覆盖表的生成规模还是消耗时间，B 集合中算法的整体性能明显优于 A 集合. 以待测实例 10^{11} 为例，B 中覆盖表最大生成规模（90 个测试用例）相比 A 中覆盖表最小生成规模（126 个测试用例）仍少 36 个测试用例，且消耗时间也少于 A，这表明，对遗传算法的种群最优个体进行爬山优化能够在加快算法收敛速度的同时有效提高算法的性能. 通过进一步的分析，我们发现在 A 集合中 GA 的性能总体优于 GA-和 GAr，但在 B 中这种性能上的差距却基本消失，这表明选择机制的不同对算法性能也有影响（优胜劣汰的轮盘赌选择机制要优于优汰

劣胜的轮盘赌选择机制和随机选择机制),但这种影响会在种群最优个体的爬山优化过程中被削弱.

6.2.2 种群规模

本文中种群规模的参数取值集合为{100, 2100, 4100, 6100},对应4个不同的参数配置,这4个参数配置下各待测实例的覆盖表生成规模和消耗时间如表10、表11所示(5次测试取最优解).加粗部分是基准参数配置 P_x,"*"标记的是Base choice实验所得各待测实例覆盖表生成的遗传算法种群规模最优取值.

表10 Base choice实验中选择不同种群规模时覆盖表的生成规模

种群规模	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^{6}6^{7}5^{6}$	$8^{2}7^{2}6^{2}5^{2}$	$6^{1}5^{1}4^{6}3^{8}2^{3}$	6^{4}	$5^{1}3^{8}2^{2}$
100	**28***	**18***	**59***	**35***	**100**	**21***	**77***	41*	**90***	**157***	**82***	75	39	46	**20***
2100	29	19	59	36	100	22	77	42	91	159	85	72	39	42*	21
4100	28	18	59	35	99*	21	77	41	91	159	83	73	**37***	**42**	21
6100	29	18	59	35	100	22	77	**41**	91	157	83	**71***	38	43	21

表11 Base choice实验中选择不同种群规模时覆盖表的消耗时间

种群规模	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^{6}6^{7}5^{6}$	$8^{2}7^{2}6^{2}5^{2}$	$6^{1}5^{1}4^{6}3^{8}2^{3}$	6^{4}	$5^{1}3^{8}2^{2}$
100	**3.09***	**2.43***	**6.93***	**9.58***	**8.94**	**6.02***	**22.4***	13.0*	**53***	**17.8***	**23.5***	3.21	10.733	0.05	**1.83***
2100	195.1	58.52	381.1	240.8	197.4	249.2	544.1	476.1	1451	631	1146	93.5	307.2	1.1*	113.8
4100	622.5	118.3	1239	506.4	391*	696.3	1177.	1371	3928	1657	3133	231	**620.95***	**2.92**	361.5
6100	1363	191.7	2842	861.4	599.7	1503	2002	**2516**	6354	3408	4914	**218***	1079.3	5.16	799.6

表10、表11中数据显示"*"大多集中在第一行,即种群规模取值为100时算法性能最优.通过进一步比较我们发现,种群规模取值不同时同一待测实例的覆盖表生成规模相互之间差距并不大,但消耗时间会随着种群规模的增加而明显增加.如待测实例 6^{20} 中4个配置下的覆盖表生成规模均为77,消耗时间的差距最大时却达到100倍之多.这表明,在本实验中种群规模对算法的影响主要体现在时间上,即种群规模的增加会增加时间负担,但不会为算法性能的提升提供较大帮助.不过除了待测实例 6^{4},上述4个配置中的算法选择都是climb算法,而没有涉及非climb算法,为了保证结论的准确性,我们利用各待测实例另做了一组验证实验,以探索种群规模取值对GA,GA-,GAr这3个非climb算法的性能影响.

验证实验步骤如下:先从表8和表9中的非climb算法中选出对应各待测实例的覆盖表生成效果最优的算法,用"#"标记(观察可知,算法集中于GA),然后用其代替各待测实例原基准配置 P_x 中的算法.在此基础上重复6.2.2节实验.实验数据如表10′和表11′所示.

表10′ Base choice验证实验中选择不同种群规模时覆盖表的生成规模

种群规模	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^{6}6^{7}5^{6}$	$8^{2}7^{2}6^{2}5^{2}$	$6^{1}5^{1}4^{6}3^{8}2^{3}$	6^{4}	$5^{1}3^{8}2^{2}$
100	33	19	76	44	126	26	104	53	126	212	107	93	43	46	24
2100	30	19	67	41	114	23	94	49	115	189	97	79	39	42	22
4100	29	19	67	41	112	23	93	49	113	185	96	79	38	42	22
6100	30	18	65	41	111	24	93	48	112	183	95	76	38	43	22

表11′ Base choice验证实验中选择不同种群规模时覆盖表的消耗时间

种群规模	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^{6}6^{7}5^{6}$	$8^{2}7^{2}6^{2}5^{2}$	$6^{1}5^{1}4^{6}3^{8}2^{3}$	6^{4}	$5^{1}3^{8}2^{2}$
100	3.9	2.9	9.19	13.2	13.76	7.94	32.18	17.14	76.3	27	32.7	3.94	12.32	0.047	1.69
2100	192.7	133.8	430.6	466.1	707.7	261.7	1086	521.1	2339	1337	1162	69.59	442.2	1.12	30.56
4100	595.7	416.3	1410	1300	2242	706.3	3020	1570	6951	4314	3383	136.2	1588.6	2.9	62.33
6100	1342	872.9	3080	2653	5026	1499	6269	2995	13553	9421	7072	225.3	2680	5.164	102

数据显示,虽然非climb算法的整体性能不如climb算法,但在非climb算法中,一定范围内增加种群规模能够减小覆盖表的生成规模.

因此我们得出结论,种群规模对非climb遗传算法的性能具有一定影响,在一定范围内种群规模越大,算法性能越好,但这种影响会被climb遗传算法中种群最优个体的爬山优化过程削弱.

6.2.3 进化代数

本文中进化代数的取值集合为{100, 600, 1100},对应3个不同的配置,这3个参数配置下各待测实例的覆盖表生成规模和消耗时间如表12、表13所示(5次测试取最优解).

表 12　Base choice 实验中选择不同进化代数时覆盖表的生成规模

进化代数	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
100	28*	18*	62	39	108	22	87	45	103	178	90	73	37*	**42***	20*
600	29	19	60	36	100*	21*	79	**41***	94	160	85	**71***	38	42	**20**
1100	**28**	**18**	**59***	**35***	**100**	**21**	**77***	41	**90***	**157***	**82***	72	**37**	43	21

表 13　Base choice 实验中选择不同进化代数时覆盖表的消耗时间

进化代数	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
100	0.30*	0.23*	0.67	1.0	0.94	0.61	2.31	479.3	5.6	1.92	2.43	69.1	52.28*	**2.9***	0.44*
600	1.78	1.40	3.82	5.40	5.04*	3.43*	12.46	**2516***	30.1	9.77	13.4	**218.4***	330.07	17.3	**1.83**
1100	**3.09**	**2.43**	**6.93***	**9.58***	**8.93**	**6.02**	**22.4***	4463	**53***	**17.8***	**23.5***	602.9	**620.95**	32.1	3.78

根据以往的经验来看，进化代数的取值会对算法的性能产生较大影响，但本实验的数据显示，对于不同大小待测实例的覆盖表生成，进化代数的影响程度并不相同. 例如在较大的待测实例 10^{11} 和 6^{20} 中，随着进化代数的增加，覆盖表的规模明显减小；但在较小的待测实例 4^{10} 和 3^{13} 中，进化代数取值不同时覆盖表的规模变化不大. 上述结论表明，进化代数的取值改变不会对较小规模待测实例的覆盖表生成产生很大影响，但对于较大规模待测实例的覆盖表生成，进化代数在一定范围内取值越大，生成规模越小. 为了验证这种现象是否是由算法种群最优个体的爬山优化所引起，我们将 3 个配置中的 climb 算法全部替换为非 climb 算法并再次测试，方法同 6.2.2 节的验证实验，实验数据如表 12′和表 13′所示.

表 12′　Base choice 验证实验中选择不同进化代数时覆盖表的生成规模

进化代数	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
100	33	20	76	44	127	25	103	49	126	210	108	78	39	42	24
600	33	21	77	44	128	25	104	48	125	210	108	76	40	42	24
1100	33	19	76	44	126	26	104	48	126	212	107	76	38	43	24

表 13′　Base choice 验证实验中选择不同进化代数时覆盖表的消耗时间

进化代数	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
100	0.34	0.266	0.8	1.22	1.29	0.70	2.95	508.2	7.22	2.56	2.86	34.58	119.4	2.9	0.25
600	2.06	1.685	4.93	7.25	7.69	4.19	17.6	2995	42.9	14.95	17.1	225.3	731.97	17.3	1.69
1100	3.9	2.9	9.19	13.2	13.8	7.94	32.2	5512	76.3	27	32.7	352.2	1588.6	32.1	2.81

数据显示：在非 climb 算法中，随着进化代数的增加，算法性能未出现明显变化，即 climb 遗传算法中种群最优个体的爬山优化过程会削弱进化代数对算法性能的影响.

6.2.4　交叉概率

本文中交叉概率的取值集合为{1，0.8，0.6，0.4，0.2}，对应 5 个不同的配置，这 5 个配置下各待测实例的覆盖表生成规模和消耗时间如表 14、表 15 所示(5 次测试取最优解).

表 14　Base choice 实验中选择不同交叉概率时覆盖表的生成规模

交叉概率	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	29	19	59	36	100	21	77	42	91	158	82	**71***	36	42	20
0.8	28	**18**	59*	**35**	**100**	21	**77**	42	**90**	**157**	**82***	71	36*	43	**20***
0.6	28	18	59	35	100	22	76	**41**	90	157	84	73	36	41*	21
0.4	29	18*	60	36	99*	21	76	41*	89	158	84	72	37	42	21
0.2	**28***	19	**59**	35*	100	**21***	76*	42	88*	156*	84	72	**37**	**42**	20

表 15　Base choice 实验中选择不同交叉概率时覆盖表的消耗时间

交叉概率	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	3.42	2.56	7.254	9.89	9.13	6.08	22.29	2567	53.6	17.58	23.67	**218***	627.67	2.86	1.97
0.8	3.28	**2.43**	6.83*	**9.58**	**8.93**	6.07	**22.4**	2551	**53**	**17.8**	**23.5***	219.9	568.05*	2.87	**1.83***
0.6	3.26	2.45	6.99	9.58	9.03	6.35	21.97	**2516**	52.8	17.58	23.84	219.7	569.9	2.5*	1.86
0.4	3.26	2.42*	7.08	9.83	8.83*	6.13	21.86	2510*	51.6	17.63	23.70	225	620.79	2.82	1.95
0.2	**3.09***	2.54	**6.93**	9.55*	8.97	**6.02***	21.84*	2574	51.4*	17.3*	23.80	214.1	**620.95**	**2.92**	1.86

表 14、表 15 中“*”分布十分分散，且比较每一列数据我们发现，交叉概率取不同值时，同一待测实例的覆盖表生成规模最大差距仅为 2 个测试用例. 由于算法性能十分接近，我们将其看做是遗传算法的不确定性所引起的随机扰动，即交叉概率在一定范围内的改变不会对覆盖表生成的遗传算法性能造成太大影响. 为了验证这一现象是否是由 climb 算法中种群最优个体的爬山优化所引起，同 6.2.2 节，我们进行了验证实验，实验数据如表 14′和表 15′所示. 数据显示，交叉概率的改变对算法性能影响仍然很小，即交叉概率对算法性能的影响程度与算法是否是 climb 算法无关.

表 14′ Base choice 验证实验中选择不同交叉概率时覆盖表的生成规模

交叉概率	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	34	21	77	45	127	25	104	48	125	210	107	76	39	42	24
0.8	33	19	76	44	126	25	104	48	126	212	107	75	39	42	24
0.6	33	20	77	44	124	25	104	48	126	211	105	78	39	41	24
0.4	33	20	76	44	124	25	103	47	125	209	106	77	39	42	23
0.2	33	20	76	44	123	26	103	49	125	209	104	79	38	43	25

表 15′ Base choice 验证实验中选择不同交叉概率时覆盖表的消耗时间

交叉概率	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	3.93	3.09	9.08	13.59	13.93	7.74	32.12	3008	78.66	27.86	30.78	225.3	1287	2.86	1.53
0.8	3.79	2.9	8.92	13.2	13.76	7.72	32.18	2986	76.3	27	32.7	196.6	1288	2.9	1.69
0.6	3.65	2.93	9.0	13.39	13.57	7.74	32.11	2995	79.03	27.59	30.30	202.1	1284	2.53	1.53
0.4	3.78	2.93	9.03	13.23	13.57	7.75	31.78	2974	78.08	26.83	30.75	196.7	1344	2.82	1.47
0.2	3.9	2.92	9.19	13.23	13.37	7.94	31.68	3070	77.86	27.09	30.03	199.2	1588.6	2.78	1.59

6.2.5 变异概率

本文中变异概率的取值集合为{1，0.8，0.6，0.4，0.2}，对应 5 个不同的配置，这 5 个配置下各待测实例的覆盖表生成规模和消耗时间如表 16、表 17 所示(5 次测试取最优解).

表 16 Base choice 实验中选择不同变异概率时覆盖表的生成规模

变异概率	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	29	18	63	41	113	22	92	47	112	184	97	73	40	42	**20***
0.8	29	19	62	40	112	22	92	46	112	185	95	**71**	40	42	21
0.6	28	18	61	39	107	21	91	46	110	178	93	71*	39	42	20
0.4	**28**	18	**59**	37	103	**21**	83	45	102	164	87	72	**37***	42	20
0.2	28*	**18***	59*	**35***	**100***	21*	**77***	**41***	**90***	**157***	**82***	73	38	**42***	21

表 17 Base choice 实验中选择不同变异概率时覆盖表的消耗时间

变异概率	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	3.39	2.70	7.61	12.45	11.78	6.8	29.23	2973	69.7	23.68	30.3	219.9	689.38	3.24	**1.83***
0.8	3.53	2.9	7.41	12.4	12.06	6.93	29.31	2880	70.7	24.28	30.3	**218.4**	695.59	3.51	2.01
0.6	3.23	2.75	7.44	12.15	11.69	6.62	29.03	2883	69.1	23.56	30.2	206*	671.1	3.49	1.90
0.4	**3.09**	2.58	**6.93**	11.03	10.23	**6.02**	25.49	2809	62.7	20.47	27.5	214.8	**620.9***	3.43	2.11
0.2	2.9*	**2.43***	6.44*	**9.58***	**8.93***	5.77*	**22.4***	**2516***	**53***	**17.8***	**23.5***	197.2	588.11	**2.9***	2.06

表 16、表 17 中“*”集中在最后一行，即本实验中绝大多数待测实例的变异概率最优取值都为 0.2. 通过对每一列数据的进一步比较可以发现，在多数覆盖表中，其生成规模和消耗时间都随着变异概率的减小总体呈递减趋势. 如在待测实例 8^{10} 中，随着变异概率的逐渐减小，生成的覆盖表规模减小了 13 个测试用例，同时算法的消耗时间从 11.78 s 逐步减到了 8.93 s. 上述结果表明，变异概率的取值对覆盖表生成的遗传算法性能具有较大影响，多数情况下，变异概率越小，算法性能越好.

同样，为了验证这一现象是否是由 climb 算法中种群最优个体的爬山优化所引起，我们进行了验证实验，实验步骤同 6.2.2 节，即以表 8 和表 9 中的覆盖表生成效果为标准，将各配置中的 climb 算法改为非 climb 算法. 实验数据如表 16′和表 17′所示. 数据显示，随着变异概率的减小，多数覆盖表的规模和消耗时间仍总体呈减少趋势，即变异概率对算法性能的影响程度与算法是否是 climb 算法无关.

表 16′　Base choice 验证实验中选择不同变异概率时覆盖表的生成规模

变异概率	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	36	22	84	51	151	27	119	51	140	249	123	78	42	42	24
0.8	35	22	81	49	147	27	116	51	139	245	121	76	41	42	24
0.6	35	21	79	48	143	26	113	51	136	239	119	77	40	41	24
0.4	33	21	76	47	139	26	111	50	132	230	114	78	38	41	24
0.2	31	19	70	44	126	24	104	48	126	212	107	77	40	41	25

表 17′　Base choice 验证实验中选择不同变异概率时覆盖表的消耗时间

变异概率	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	4.17	3.53	9.83	16.29	17.93	8.82	38.92	3319	91.35	35.48	38.03	198.0	1485.2	3.25	1.69
0.8	4.18	3.62	9.95	16.22	18.32	8.61	38.95	3362	92.59	35.13	38.31	225.3	1462.5	2.9	1.58
0.6	4.01	3.43	9.52	15.65	17.71	8.35	37.81	3262	90.75	34.71	37.22	196.1	1446.6	2.49	1.56
0.4	3.9	3.29	9.19	14.87	16.37	7.94	35.88	3191	86.21	31.92	34.71	190.9	1588.6	3.32	1.48
0.2	3.18	2.9	7.58	13.2	13.76	6.89	32.18	2995	76.3	27	32.7	173.0	1382.2	2.49	1.43

综合上述实验数据可以得到遗传算法 5 个配置参数在不同待测实例中的覆盖表生成的最优参数取值.我们将这些最优参数取值进行组合,即得到不同待测实例的遗传算法覆盖表生成的最优参数配置.如表 18 所示,待测实例不同,其最优参数配置也不同.下面我们利用这些最优参数配置生成 15 个待测实例的覆盖表,为了保证结果的准确性,我们从 10 次生成结果中取最优解.

表 18　base choice 实验所得各待测实例的最优参数配置及覆盖表生成结果

实例	算法	m	T	Pc	Pm	$Size$	$Time$/s	实例	算法	m	T	Pc	Pm	$Size$	$Time$/s
4^{10}	GAr climb	100	100	0.2	0.2	**28**	**0.234**	6^{30}	GA climb	100	1100	0.2	0.2	**87**	**71.99**
3^{13}	GAr climb	100	100	0.4	0.2	**18**	**0.187**	10^{11}	GA climb	100	1100	0.2	0.2	**154**	**22.09**
6^{10}	GA climb	100	1100	0.8	0.2	**59**	**6.724**	$7^6 6^7 5^6$	GAr climb	100	1100	0.8	0.2	**82**	**29.39**
4^{20}	GAr climb	100	1100	0.2	0.2	**35**	**11.75**	$8^2 7^2 6^2 5^2$	GA- climb	6100	600	1.0	0.6	**70**	**2413**
8^{10}	GAr climb	4100	600	0.4	0.2	**101**	**252.6**	$6^1 5^1 4^6 3^8 2^3$	GAr climb	4100	100	0.8	0.4	**39**	**69.51**
3^{20}	GA- climb	100	600	0.2	0.2	**21**	**3.57**	6^4	GAr climb	2100	100	0.6	0.2	**42**	**1.123**
6^{20}	GA climb	100	1100	0.2	0.2	**75**	**30.9**	$5^1 3^8 2^2$	GAr climb	100	100	0.8	1.0	**21**	**0.234**
4^{30}	GAr climb	100	600	0.4	0.2	**41**	**16.97**								

Base choice 实验能够准确评价遗传算法 5 个配置参数各自对算法性能的影响性质和影响程度,并找出每一个配置参数在不同待测实例的覆盖表生成问题中的最优取值,但实验只注重算法配置参数的局部优化,没有考虑算法整体性能是否得到提升.实验数据显示,在待测实例 8^{10} 中,覆盖表最小生成规模为 99(见实验过程),而 Base choice 实验得出最优参数配置的覆盖表生成规模为 101,因此算法各配置参数达到局部最优时,算法整体性能不一定最优.为了更好地探索覆盖表生成的遗传算法最优配置,我们设计了另一条实验路线——爬山实验,爬山实验能够较好地处理算法局部配置参数优化和算法整体性能提升之间的平衡关系,实验对参数配置的每一轮优化都是在上一轮优化的基础上进行,算法的整体性能会随着配置参数的逐步优化而一步步提升,从而得到能使算法整体性能达到最优的参数配置.

6.3　爬山实验

实验以 pair-wise 实验所得最优参数配置 P_x 作为基准参数配置.由于爬山实验是在实验过程中一步步对遗传算法的配置参数进行优化,所以配置集是动态生成的,下面我们将按照算法配置参数的优化顺序对实验进行逐步分析并找出该配置参数的最优取值.考虑到演化搜索算法结果的不确定性和运行时间较长这一实际问题,实验从 5 次运行结果中选取最优解作为测试结果.

6.3.1　遗传算法的变种算法

首先改变遗传算法的变种算法这一参数取值,由于基准参数配置 P_x 不一致,每个待测实例各自产生 5 个参数配置,对应算法的不同参数取值.实验结果见表 8、表 9.综合覆盖表的生成规模和消耗时间这两个标准,我们得到算法选择的最优取值,其所对应的参数配置 C_1 如表 19 所示.

表 19　爬山实验各待测实例中算法优化所对应的参数配置C_1

实例	a_1	a_2	a_3	a_4	a_5	实例	a_1	a_2	a_3	a_4	a_5
4^{10}	**5**	0	2	4	3	6^{30}	**3**	0	2	1	4
3^{13}	**5**	0	2	1	4	10^{11}	**3**	0	2	1	4
6^{10}	**3**	0	2	4	3	$7^6 6^7 5^6$	**5**	0	2	1	4
4^{20}	**5**	0	2	1	4	$8^2 7^2 6^2 5^2$	**4**	3	1	0	1
8^{10}	**3**	0	2	1	4	$6^1 5^1 4^6 3^8 2^3$	**5**	2	2	4	3
3^{20}	**4**	0	2	4	3	6^4	**5**	2	0	4	4
6^{20}	**3**	0	2	1	4	$5^1 3^8 2^2$	**5**	0	1	1	0
4^{30}	**5**	3	1	2	4						

6.3.2 种群规模

在 C_1 的基础上我们对种群规模的取值进行改变，每个待测实例各自对应 4 个不同的参数配置，测试结果如表 20、表 21，表中“*”标记种群规模的最优参数取值. 由此我们得到各待测实例的参数配置 C_2，如表 22 所示. 通过观察表 20、表 21 中数据可知，种群规模的改变对算法性能的影响主要体现在消耗时间上，对覆盖表的生成规模没有产生较大影响，这进一步验证了 Base choice 实验中的结论.

表 20 爬山实验中选择不同种群规模时覆盖表的生成规模

种群规模	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
100	29	17*	59	35*	99	21*	74*	40*	89*	154*	82*	75	39	42*	20*
2100	28*	18	60	35	98*	22	77	41	89	155	85	70*	39	42	21
4100	29	18	60	36	99	21	77	41	90	156	83	72	36*	42	20
6100	28	18	58*	36	98	21	76	41	91	155	83	70	38	43	21

表 21 爬山实验中选择不同种群规模时覆盖表的消耗时间

种群规模	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
100	2.99	2.28*	6.16	10.1*	10.25	5.67*	22.9*	12.7*	53.3*	19.8*	23.5*	3.68	10.73	0.13*	1.78*
2100	64.4*	56.1	135	394.5	614*	246	897	459	1898	1529	1146	283.2*	37.21	1.31	38.75
4100	131	118.9	269	1222	1972	679	2612	1354	5591	5150	3133	899.7	627.7*	2.92	77.84
6100	202	190.3	413*	2431	4379	1465	5272	2474	10500	9267	4914	2034	1079.3	4.56	132.4

表 22 爬山实验各待测实例中种群规模优化所对应的参数配置 C_2

实例	a_1	a_2	a_3	a_4	a_5	实例	a_1	a_2	a_3	a_4	a_5
4^{10}	**5**	**1**	2	4	3	6^{30}	**3**	**0**	2	1	4
3^{13}	**5**	**0**	2	1	4	10^{11}	**3**	**0**	2	1	4
6^{10}	**3**	**3**	2	4	3	$7^6 6^7 5^6$	**5**	**0**	2	1	4
4^{20}	**5**	**0**	2	1	4	$8^2 7^2 6^2 5^2$	**4**	**1**	1	0	1
8^{10}	**3**	**1**	2	1	4	$6^1 5^1 4^6 3^8 2^3$	**5**	**2**	2	4	3
3^{20}	**4**	**0**	2	4	3	6^4	**5**	**0**	0	4	4
6^{20}	**3**	**0**	2	1	4	$5^1 3^8 2^2$	**5**	**0**	1	1	0
4^{30}	**5**	**0**	1	2	4						

6.3.3 进化代数

在 C_2 的基础上改变进化代数的参数取值，对应各待测实例的 3 个不同参数配置，测试结果如表 23、表 24 所示，其中“*”标记进化代数的最优取值. 比较表中的数据，我们发现在 $7^6 6^7 5^6$、10^{11} 和 8^{10} 这些规模较大的待测实例中，进化代数取值越大时覆盖表生成规模越小；而在 3^{20}、4^{10} 和 $5^1 3^8 2^2$ 这些规模较小的待测实例中覆盖表生成规模变化却没有明显规律，这验证了 Base choice 实验中的结论. 实验最终得到各待测实例的参数配置 C_3，见表 25.

表 23 爬山实验中选择不同进化代数时覆盖表的生成规模

进化代数	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
100	29	19	61	37	103	22	83	43	101	166	90	72	37	42*	20*
600	28*	19	60	36	100	21*	77	40*	91	158	85	70*	38	43	20
1100	28	17*	58*	35*	98*	21	74*	40	89*	154*	82*	71	36*	43	21

表 24 爬山实验中选择不同进化代数时覆盖表的消耗时间

进化代数	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
100	6.11	0.172	39.1	1.03	58.9	0.61	2.34	2.36	5.59	2.18	2.43	45.24	52.28	0.13*	0.44*
600	34.8*	1.7	224	5.71	339	3.34*	12.83	12.7*	30.0	11.6	13.4	283*	330.07	0.33	1.78
1100	64.4	2.28*	413*	10.1*	614*	5.67	22.9*	22.2	53.3*	19.8*	23.5*	489.1	627.7*	0.70	3.26

表 25 爬山实验各待测实例中进化代数优化所对应的参数配置 C_3

实例	a_1	a_2	a_3	a_4	a_5	实例	a_1	a_2	a_3	a_4	a_5
4^{10}	**5**	**1**	**1**	4	3	6^{30}	**3**	**0**	**2**	1	4
3^{13}	**5**	**0**	**2**	1	4	10^{11}	**3**	**0**	**2**	1	4
6^{10}	**3**	**3**	**2**	4	3	$7^6 6^7 5^6$	**5**	**0**	**2**	1	4
4^{20}	**5**	**0**	**2**	1	4	$8^2 7^2 6^2 5^2$	**4**	**1**	**1**	0	1
8^{10}	**3**	**1**	**2**	1	4	$6^1 5^1 4^6 3^8 2^3$	**5**	**2**	**2**	4	3
3^{20}	**4**	**0**	**1**	4	3	6^4	**5**	**0**	**0**	4	4
6^{20}	**3**	**0**	**2**	1	4	$5^1 3^8 2^2$	**5**	**0**	**0**	1	0
4^{30}	**5**	**0**	**1**	2	4						

6.3.4 交叉概率

在 C_3 的基础上改变交叉概率的取值，对应各待测实例的 5 个不同参数配置，测试结果如表 26、表 27 所示，其中“*”标记交叉概率的最优取值. 通过对表中数据进行观察可知，交叉概率的改变对覆盖表的生成规模及消耗时间没有太大影响，不同交叉概率取值下覆盖表的生成规模之间差距很小，且没有明显的规律可循. 结论与 Base choice 实验一致. 在对交叉概率进行优化后我们得到各待测实例的参数配置 C_4，见表 28.

表 26　爬山实验中选择不同交叉概率时覆盖表的生成规模

交叉概率	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	29	19	61	37	99	21	76	41	89	155	82	70	36	42	21
0.8	28	17*	60	35*	98	21	74*	41	89	154*	82*	70*	36*	42	20*
0.6	29	18	61	36	98*	22	75	40	88	155	84	71	36	41*	21
0.4	28*	19	59	36	99	21	75	41	88	156	84	72	37	43	22
0.2	28	19	58*	36	98	21*	76	40*	87*	155	84	71	36	42	21

表 27　爬山实验中选择不同交叉概率时覆盖表的消耗时间

交叉概率	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	37.44	2.76	416.4	11.2	607.7	3.37	23.1	12.41	54.2	20.8	23.67	283	626.2	0.047	0.45
0.8	35.68	2.28*	407.2	10.1*	614	3.4	22.9*	12.24	53.3	19.8*	23.5*	281*	568.1*	0.19	0.44*
0.6	35.37	3.06	418.1	11.3	604*	3.45	22.95	12.71	53.3	20.3	23.84	282	569.9	0.03*	0.47
0.4	34.3*	2.67	411.2	10.6	611	3.45	22.74	12.25	53.0	20.3	23.70	291	620.8	0.047	0.47
0.2	34.8	2.78	413*	10.3	614.	3.34*	22.93	12.4*	52.6*	20.15	23.80	263	627.7	0.13	0.45

表 28　爬山实验各待测实例中交叉概率优化所对应的参数配置 C_4

实例	a_1	a_2	a_3	a_4	a_5	实例	a_1	a_2	a_3	a_4	a_5
4^{10}	**5**	**1**	**1**	**3**	3	6^{30}	**3**	**0**	**2**	**4**	4
3^{13}	**5**	**0**	**2**	**1**	4	10^{11}	**3**	**0**	**2**	**1**	4
6^{10}	**3**	**3**	**2**	**4**	3	$7^6 6^7 5^6$	**5**	**0**	**2**	**1**	4
4^{20}	**5**	**0**	**2**	**1**	4	$8^2 7^2 6^2 5^2$	**4**	**1**	**1**	**1**	1
8^{10}	**3**	**1**	**2**	**2**	4	$6^1 5^1 4^6 3^8 2^3$	**5**	**2**	**2**	**1**	3
3^{20}	**4**	**0**	**1**	**4**	3	6^4	**5**	**0**	**0**	**2**	4
6^{20}	**3**	**0**	**2**	**1**	4	$5^1 3^8 2^2$	**5**	**0**	**0**	**1**	0
4^{30}	**5**	**0**	**1**	**4**	4						

6.3.5　变异概率

在 C_4 的基础上对变异概率的取值进行改变，对应各待测实例的 5 个不同参数配置，测试结果见表 29、表 30，其中“*”标记交叉概率的最优取值. 通过观察我们发现，随着变异概率的减小，覆盖表的生成规模和消耗时间总体呈减小趋势. 这表明在一定范围内变异概率的取值越小，覆盖表生成的遗传算法性能越好，与 Base choice 实验结论一致. 实验最终得到各待测实例的参数配置 C_5，见表 31.

表 29　爬山实验中选择不同变异概率时覆盖表的生成规模

变异概率	覆盖表规模														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	29	19	61	41	109	23	93	49	112	185	97	71	40	43	20
0.8	29	18	61	39	104	22	91	48	110	183	95	70	40	43	20
0.6	28	19	59	39	102	22	88	47	107	172	93	70*	39	42	20
0.4	28	19	58	36	99	21	81	44	98	161	87	71	36*	42	21
0.2	28*	17*	58*	35*	98*	21*	74*	40*	87*	154*	82*	70	38	41*	20*

表 30　爬山实验中选择不同变异概率时覆盖表的消耗时间

变异概率	消耗时间/s														
	4^{10}	3^{13}	6^{10}	4^{20}	8^{10}	3^{20}	6^{20}	4^{30}	6^{30}	10^{11}	$7^6 6^7 5^6$	$8^2 7^2 6^2 5^2$	$6^1 5^1 4^6 3^8 2^3$	6^4	$5^1 3^8 2^2$
1.0	37.71	2.87	431.2	12.1	716.9	3.84	30.22	16.47	71.1	27.5	30.3	290.2	699.38	0.047	0.44
0.8	38.36	3.70	425.4	12.2	680.3	3.74	30.14	16.54	71.5	27.6	30.3	281	695.59	0.047	0.45
0.6	37.04	3.15	422.2	11.5	666.5	3.74	29.16	16.21	69.0	25.8	30.2	277*	671.1	0.047	0.45
0.4	34.3	3.01	413.7	10.8	635.4	3.34	26.04	14.52	61.7	22.7	27.5	273.4	568.1*	0.047	0.442
0.2	32.1*	2.3*	402*	10.1*	604*	3.31*	22.9*	12.4*	52.6*	19.8*	23.5*	279.5	588.1	0.03*	0.43*

表 31　爬山实验各待测实例中变异概率优化所对应的参数配置 C_5

实例	a_1	a_2	a_3	a_4	a_5	实例	a_1	a_2	a_3	a_4	a_5
4^{10}	**5**	**1**	**1**	**3**	**4**	6^{30}	**3**	**0**	**2**	**4**	**4**
3^{13}	**5**	**0**	**2**	**1**	**4**	10^{11}	**3**	**0**	**2**	**1**	**4**
6^{10}	**3**	**3**	**2**	**4**	**4**	$7^6 6^7 5^6$	**5**	**0**	**2**	**1**	**4**
4^{20}	**5**	**0**	**2**	**1**	**4**	$8^2 7^2 6^2 5^2$	**4**	**1**	**1**	**1**	**2**
8^{10}	**3**	**1**	**2**	**2**	**4**	$6^1 5^1 4^6 3^8 2^3$	**5**	**2**	**2**	**1**	**3**
3^{20}	**4**	**0**	**1**	**4**	**4**	6^4	**5**	**0**	**0**	**2**	**4**
6^{20}	**3**	**0**	**2**	**1**	**4**	$5^1 3^8 2^2$	**5**	**0**	**0**	**1**	**4**
4^{30}	**5**	**0**	**1**	**4**	**4**						

通过这 5 轮实验，我们得到各待测实例覆盖表生成的最优参数配置. 如表 32 所示，不同覆盖表的最优参数配置各不相同. 下面我们利用这些最优参数配置生成各待测实例的覆盖表，为了保证结果的准确性，我们从 10 次生成结果中取最优解.

表 32 爬山实验所得各待测实例的最优参数配置及覆盖表生成结果

实例	算法	m	T	Pc	Pm	$Size$	$Time$/s	实例	算法	m	T	Pc	Pm	$Size$	$Time$/s
4^{10}	GAr climb	2100	600	0.4	0.2	**28**	**32.1**	6^{30}	GA climb	100	1100	0.2	0.2	**87**	**52.6**
3^{13}	GAr climb	100	1100	0.8	0.2	**17**	**2.28**	10^{11}	GA climb	100	1100	0.8	0.2	**154**	**19.8**
6^{10}	GA climb	6100	1100	0.2	0.2	**58**	**402**	$7^6 6^7 5^6$	GAr climb	100	1100	0.8	0.2	**82**	**23.5**
4^{20}	GAr climb	100	1100	0.8	0.2	**35**	**10.1**	$8^2 7^2 6^2 5^2$	GA- climb	2100	600	0.8	0.6	**70**	**277**
8^{10}	GA climb	2100	600	0.6	0.2	**98**	**604**	$6^1 5^1 4^6 3^8 2^3$	GAr climb	4100	1100	0.8	0.4	**36**	**568.1**
3^{20}	GA- climb	100	600	0.2	0.2	**21**	**3.31**	6^4	GAr climb	100	100	0.6	0.2	**41**	**0.03**
6^{20}	GA climb	100	1100	0.8	0.2	**74**	**22.9**	$5^1 3^8 2^2$	GAr climb	100	100	0.8	0.2	**20**	**0.43**
4^{30}	GAr climb	100	600	0.2	0.2	**40**	**12.4**								

6.4 实验结论

本实验通过 pair-wise 实验，Base choice 实验以及爬山实验这 3 条实验路线系统地探索了覆盖表生成的遗传算法中各配置参数的取值及其相互作用对算法性能的影响. 3 条实验路线探索算法性能的角度和侧重点各不相同，pair-wise 实验主要探索遗传算法各配置参数之间的二维组合关系对算法性能的影响，实验的参数配置集是遗传算法 5 个配置参数的二元组合覆盖表. Base choice 实验更侧重于发掘各配置参数自身对算法效果的影响，实验对遗传算法的 5 个配置参数进行了单独的深入探索. 爬山实验则是对参数配置进行逐步优化，参数配置的整体性能会随着每一轮参数优化的完成而一步步提高.

通过实验我们得出结论，算法选择、进化代数和变异概率这 3 个配置参数对覆盖表生成的遗传算法整体性能影响较大，种群规模的影响只体现在算法消耗时间上，而交叉概率的影响相对较小. 其中算法选择以 GA climb 和 GAr climb 算法较优，且进化代数取值较大，交叉概率取值较小时算法性能会有明显提高. 结论表明，对于某个特定问题的二维覆盖表生成，遗传算法中存在一组最优参数配置，使其总是能发挥较优的性能. 表 33 中是比较三组实验后所得 15 个待测实例的最优参数配置和覆盖表生成结果. 观察可知，不同待测实例中，覆盖表生成的最优参数配置之间差异很大，即不存在一组适用于所有二维覆盖表生成的最优参数配置.

表 33 各待测实例的最终最优配置和覆盖表生成结果

实例	算法	m	T	Pc	Pm	$Size$	$Time$/s	实例	算法	m	T	Pc	Pm	$Size$	$Time$/s
4^{10}	GAr climb	100	100	0.2	0.2	**28**	**0.234**	6^{30}	GA climb	100	1100	0.2	0.2	**87**	**52.6**
3^{13}	GAr climb	100	1100	0.8	0.2	**17**	**2.28**	10^{11}	GA climb	100	1100	0.8	0.2	**154**	**19.8**
6^{10}	GA climb	6100	1100	0.2	0.2	**58**	**402**	$7^6 6^7 5^6$	GAr climb	100	1100	0.8	0.2	**82**	**23.5**
4^{20}	GAr climb	100	1100	0.8	0.2	**35**	**10.1**	$8^2 7^2 6^2 5^2$	GA- climb	2100	600	0.8	0.6	**70**	**277**
8^{10}	GA climb	2100	600	0.6	0.2	**98**	**604**	$6^1 5^1 4^6 3^8 2^3$	GAr climb	4100	1100	0.8	0.4	**36**	**568.1**
3^{20}	GA- climb	100	600	0.2	0.2	**21**	**3.31**	6^4	GAr climb	100	100	0.6	0.2	**41**	**0.03**
6^{20}	GA climb	100	1100	0.8	0.2	**74**	**22.9**	$5^1 3^8 2^2$	GAr climb	100	100	0.8	0.2	**20**	**0.43**
4^{30}	GAr climb	100	600	0.2	0.2	**40**	**12.4**								

7 总结和展望

本文系统地研究了遗传算法这一演化搜索算法在进行覆盖表生成时的性能优化问题，通过 3 条实验路线探索遗传算法的变种算法、种群规模、进化代数、交叉概率和变异概率这 5 个因素及其相互作用对算法性能的影响，得出以下结论：

(1) 覆盖表生成的遗传算法性能受算法的参数配置影响很大，不同参数配置下的覆盖表生成结果之间具有明显差异. 例如参数配置中算法选择为 GA climb 或 GAr climb 时算法生成覆盖表的规模普遍较小，而算法选择为 GA-时，则会生成较大规模的覆盖表.

(2) 遗传算法的各配置参数对算法生成覆盖表性能存在不同程度的影响，其中算法选择、进化代数和变异概率这 3 个配置参数对覆盖表生成的遗传算法整体性能影响较大，种群规模的影响只体现在算法消耗时间上，而交叉概率的影响相对较小.

(3) 对于特定的二维覆盖表生成问题，存在一组最优参数配置，使得遗传算法总能发挥较优的性能. 对于一般的二维覆盖表生成问题，则不存在这样一组通用的最优参数配置. 不同的覆盖表生成问题，其最优参数配置之间存在差异，但也具有一定的共性，实验结论显示，在本文涉及的 15 个待测实例的最优参数配置中，算法选择集中于 GA climb 和 GAr climb 算法，进化代数取值总体较大，变异概率取值总体较小.

在以后的工作中，我们将在已有的工作基础上对实验中所得到的遗传算法最优参数配置进行验证

和进一步优化，例如，以实验结论为基础扩大配置参数取值集合，进一步探索 climb 算法下更大进化代数，更小变异概率的遗传算法性能；对相同参数配置下的待测实例进行 10 次甚至 20 次的覆盖表生成实验以使实验结果更加精确等. 另一方面，由于实验规模和复杂度会进一步增加，我们计划将实验迁移到云计算环境中，利用云计算平台优越的计算能力更好更快地设计和实施实验.

针对覆盖表生成过程中，遗传算法的通用最优配置不存在的情况，我们将研究利用本文实验方法构建一种两阶段多配置并行遗传算法，该方法首先使用 pair-wise 参数覆盖表配置一组遗传算法先行进行计算，然后选择其中性能好的配置，以这几个配置为基础分别作爬山和 Base choice 配置试验，用它们中产生的最好的结果作为并行遗传算法的结果. 关于这个方法的效果验证我们将另文讨论.

针对特定的覆盖表生成实例，另外一个改进遗传算法性能的方向是我们拟采用配置演化的方法，演化过程中首先随机生成一组遗传算法的配置作为初始种群，运行这组配置对应的遗传算法，生成该特定实例的覆盖表，以每个算法生成的该特定实例的覆盖表规模作为适应值度量，覆盖表规模越小，适应值越高，从而对这组配置进行适应值排序，并对其进行选择、交叉和变异，形成第二代种群. 循环该过程，直到性能不再提高或达到某种收敛标准，系统研究这种方法的性能和成本.

参 考 文 献

[1] Nie Changhai, Leung Hareton. A survey of combinatorial testing. ACM Computing Survey, 2011, 43(2), Article 11: 1-29

[2] Kuhn D, Reilly M. An investigation of the applicability of design of experiments to software testing//Proceedings of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop. Los Alamitos, CA, 2002: 91

[3] Williams Alan W, Prober Robert L. A practical strategy for testing pair-wise coverage of network interfaces//Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE1996). White Plaints, NY, USA, 1997: 246-254

[4] Cohen D M, Dalal S R, Fredman M L, Patton G C. The AETG system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering, 1997, 23(7): 437-444

[5] Colbourn C J, Cohen M B, Turban R C. A deterministic density algorithm for pairwise interaction coverage//Proceedings of the IASTED International Conference on Software Engineering. Innsbruck, Austria, 2004: 242-252

[6] Bryce Renée C, Colbourn Charles J, Cohen Myra B. A framework of greedy methods for constructing interaction test suites//Proceedings of the 27th International Conference on Software Engineering (ICSE2005). St. Louis, Missouri, USA, 2005: 146-155

[7] Cohen Myra B, Gibbons Peter B, Mugridge Warwick B, Colbourn Charles J. Constructing test suites for interaction testing//Proceedings of the 25th International Conference on Software Engineering (ICSE2003). Portland, Oregon, USA, 2003: 38-48

[8] Nurmela Kari J. Upper bounds for covering arrays by tabu search. Discrete Applied Mathematics, 2004, 138 (1-2): 143-152

[9] Ghazi S A, Ahmed M A. Pair-wise test coverage using genetic algorithms//Proceedings of the 2003 Congress on Evolutionary Computation. Canberra, Australia, 2003, 2: 1420-1424

[10] Shiba Toshiaki, Tsuchiya Tatsuhiro, Kikuno Tohru. Using artificial life techniques to generate test cases for combinatorial testing//Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC2004). Hong Kong, China, 2004: 72-78.

[11] McCaffrey James D. An empirical study of pairwise test set generation using a genetic algorithm//Proceedings of the 7th International Conference on Information Technology. Las Vegas, NV, 2010: 992-997

[12] Grindal Mats, Lindstrom Birgitta, Jefferson Offutt A, Andler Sten F. An evaluation of combination strategies for test case selection. Department of Computer Science, University of Skovde: Technical Report HS-IDA-TR-03-001, 2004

[13] Grindal Mats, Lindstrom Birgitta, Offutt Jefferson A, Andler Sten F. An evaluation of combination strategies for test case selection. Empirical Software Engineering, 2006, 11 (4): 583-611

[14] Frenzel James F. Genetic algorithms, a new breed of optimization. IEEE Potentials, 1993: 21-24

LIANG Ya-Lan, born in 1988, master candidate. His research interests is software testing, especially on combinatorial testing.

NIE Chang-Hai, born in 1971, professor, Ph. D. supervisor. His research interests are software testing techniques, including test suite reduction and optimization, metamorphic testing, evolutionary testing, and especially combinatorial testing.

Background

This work was supported by the National Natural Science Foundation of China (60773104, 61021062, The research on the key issues of combinatorial testing), the National High Technology Research and Development Program (863 Program) of China (2008AA01Z143, Combinatorial testing technology and its support tools) and the National Natural Science Foundation of Jiangsu province, China (BK2010372, The theory, method and application of combinatorial testing).

Combinatorial testing is an effective software testing method which can detect the failures triggered by various factors and their interactions in the software system. To date, the existing research in the world are mainly focused on test case generation, the application of combinatorial testing et al. we will strive to put forward the existing research to a new height. Our research has got some achievements involving the following aspects: combinatorial testing model; mining on the parameters, their values and interactions; various test case generation and prioritization algorithm; the design, evolution and application of combinatorial testing procedure and strategy; software quality evaluation method and testing automation. Our research can not only make combinatorial testing find faults effectively with low cost, but also can diagnosis and reveal faults further around the exposed ones, provide reasonable evaluation method et al. our work in this area will enhance combinatorial testing and provide theory, method and tool for its application and extension.

This paper focused on the study of test case generation and prioritization algorithm. We take genetic algorithm, one of the typical evolutionary search methods, as an example to systemically explore the different influences of its five configurable parameters on the performance of 2-way covering array generation, trying to find the optimal configuration of genetic algorithm and further optimize the performance of the algorithm.