软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(4):855−878 [doi: 10.13328/j.cnki.jos.004974]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

利用蚁群算法生成覆盖表:探索与挖掘*

曾梦凡, 陈思洋, 张文茜, 聂长海

(计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

通讯作者: 聂长海, E-mail: changhainie@nju.edu.cn

摘 要: 覆盖表生成问题是组合测试的重要研究内容之一,目前已有许多数学方法、贪心算法、搜索算法用于求解这一问题.蚁群算法作为一种能够有效求解组合优化问题的演化搜索算法,已被应用到求解覆盖表生成问题中.已有的研究工作表明:蚁群算法适于求解一般覆盖表、变力度覆盖表生成以及覆盖表排序等问题,但算法结果与其他覆盖表生成方法相比并不具有优势.为了进一步探索与挖掘蚁群算法生成覆盖表的潜力,进行了如下 4 个层次的改进工作:(1) 算法变种集成;(2) 算法参数配置优化;(3) 演化对象结构调整及演化策略改进;(4) 利用并行计算优化算法时间开销.实验结果表明:通过以上 4 个层次的改进,蚁群算法生成覆盖表的性能有了显著提升.

关键词: 覆盖表;蚁群算法;演化搜索算法;并行计算;组合测试;软件测试

中图法分类号: TP311

中文引用格式: 曾梦凡,陈思洋,张文茜,聂长海.利用蚁群算法生成覆盖表:探索与挖掘.软件学报,2016,27(4):855−878. http://www.jos.org.cn/1000-9825/4974.htm

英文引用格式: Zeng MF, Chen SY, Zhang WQ, Nie CH. Generating covering arrays using ant colony optimization: Exploration and mining. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):855−878 (in Chinese). http://www.jos.org.cn/1000-9825/4974.htm

Generating Covering Arrays Using Ant Colony Optimization: Exploration and Mining

ZENG Meng-Fan, CHEN Si-Yang, ZHANG Wen-Qian, NIE Chang-Hai

(State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

Abstract: Generation of covering arrays, which has been solved by many mathematical methods and greedy algorithms as well as search based algorithms, is one of significant problems in combinatorial testing. As an effective evolutionary search algorithm for solving combinatorial optimization problems, ant colony optimization has also been used to generate covering arrays. Existing research shows ant colony optimization suitable for generating general covering arrays, variable strength covering arrays and the prioritization of covering arrays. Unfortunately, compared with other methods, ant colony optimization doesn't have significant advantages. To further explore and mine the potential of ant colony optimization in generating covering arrays, this paper focuses on four levels of improvement: 1) the integration of ant colony variants; 2) parameter tuning; 3) the adjustment of solution structure and the improvement of evolutionary strategy; 4) using parallel computing to save executing time. The experimental results show that ant colony optimization is much more effective in generating covering arrays after the improvements.

Key words: covering array; ant colony optimization; evolutionary algorithm; parallel computing; combinatorial testing; software testing

软件测试是软件生命周期的重要阶段,是构建安全、稳定、可靠的高质量软件的必要环节[1].如今,软件自

* 基金项目: 国家自然科学基金(61272079, 61321491, 91318301); 教育部博士点基金(20130091110032)

Foundation item: National Natural Science Foundation of China (61272079, 61321491, 91318301); Research Fund for the Doctoral Program of Higher Education of China (20130091110032)

收稿时间: 2015-08-30; 修改时间: 2015-10-15; 采用时间: 2015-11-20; jos 在线出版时间: 2016-01-13

CNKI 网络优先出版: 2016-01-14 13:15:58, http://www.cnki.net/kcms/detail/11.2560.TP.20160114.1315.003.html

身和软件运行环境都变得越来越复杂,为了能够对一个软件的质量做出全面的测试,往往需要综合使用各种测试方法.

在众多的软件测试方法中,组合测试提供了一种对系统中各个组件交互影响所引发故障的有效检测方法.

组合测试的重要研究内容之一是:如何为待测软件系统(software under test,简称 SUT)生成一个满足特定条件的测试集.这个问题也被称为组合测试覆盖表生成问题[2].所谓覆盖表,就是一个 N(行)×k(列)的表,其对应的 SUT 模型中任意 t 个参数的任意值组合在表中至少出现一次,其中,N 代表了测试用例的数目,k 是软件系统参数的数目,t 是覆盖表强度.如何生成一个 N 尽可能小的覆盖表,是覆盖表生成问题的关键所在.目前,已有一系列基于数学方法、贪心方法和搜索方法的算法被应用于求解这一问题,然而这些算法中并没有一种方法能够针对任意情形给出最优解.在这些求解算法中,一类基于演化搜索技术的算法表现出了良好的性能,如遗传算法[3]、粒子群算法、蚁群算法等.

蚁群算法是一种模仿蚂蚁觅食行为的演化搜索算法,通过群智演化的方法来对一系列组合优化问题进行求解.目前,已有相关工作将蚁群算法应用于求解一般覆盖表生成、变力度覆盖表生成和覆盖表排序中.这些研究工作表明,蚁群算法适于求解覆盖表生成等问题.然而,这些研究工作侧重于应用蚁群算法求解覆盖表生成问题,而没有对算法在这一问题求解中的性能进行深入挖掘.例如:蚁群算法是多变种算法,究竟何种变种更适合求解覆盖表生成问题?蚁群算法性能优劣依赖于参数配置,究竟何种参数配置能够发挥算法性能极限?一次生成一条测试用例的生成方式能否发挥出蚁群算法的潜力?如何解决算法运算时间开销大的问题?

为了回答以上问题,探索与挖掘利用蚁群算法生成覆盖表的潜力,本文进行了以下 4 个层次的研究工作:第 1 层,算法变种集成,结合覆盖表生成问题将蚁群算法的 3 个主要变种整合为一个利用蚁群算法生成覆盖表的框架;第 2 层,算法参数调优,在第 1 层的算法框架上对算法参数配置进行调优,给出推荐参数配置,并比较优化后 3 个变种算法性能上的差距;第 3 层,演化对象结构调整及演化策略改进,改变原有的一次生成一条测试用例的生成方法,使蚁群算法一次对一个表进行演化,同时对算法策略进行相应研究;第 4 层,利用并行计算降低算法时间开销.

通过以上 4 个层次的研究,我们发现:经过优化后的算法在时间开销和生成结果上均得到了显著提升,部分结果甚至接近当前已知最优解.

本文第 1 节介绍覆盖表生成、蚁群算法的相关概念与研究现状.第 2 节介绍利用蚁群算法生成覆盖表的集成框架.第 3 节介绍利用蚁群算法生成覆盖表各种变种算法的参数优化,并比较参数优化后的 3 个基本变种的优劣.第 4 节介绍如何使用蚁群算法对一个表进行演化.第 5 节将算法的某些部分并行化.第 6 节进行全文总结.

1 背 景

1.1 覆盖表及其生成问题

组合测试是一种特殊的软件测试方法,它的使用一般包括下面几个步骤:参数建模、覆盖表生成、测试用例约简和排序、测试用例执行、故障定位、评估.然而,覆盖表生成是其中的一个重要环节,现有的大部分对组合测试的研究工作是针对覆盖表生成而展开的[4].

下面我们给出一些常用的定义.

定义 1. 如果一个 $N\times k$ 的二维表满足下列两个条件:

(1) 每一列 $i(1\leqslant i\leqslant k)$中元素的所有取值都取自集合 V_i;

(2) 对于它的任意一个 $N\times t(1\leqslant t\leqslant k)$的子表,都包含对应 t 个参数的所有值组合至少一次.

那么这样的表称为覆盖强度为 t 的覆盖表,也称为 t 维(t-way)覆盖表.

定义 2. 设 T 为 SUT 的一个 t 维覆盖表:

(1) 若 SUT 所有参数的取值数目$|V_i|$均相等,则 T 可以表示为 CA(N;t;$|V_i|^k$),称这样的覆盖表为水平覆盖表,其中,N 为覆盖表规模,即测试用例数目,k 为参数的个数;

(2) 若$|V_i|$不尽相等,则 T 可以表示为 MCA(N;t;k;$|V_1|$,$|V_2|$,…,$|V_k|$),称这样的覆盖表为混合覆盖表.

表 1 给出了一个 Android 应用可能的运行环境,包括 Android 版本、内存大小、系统分辨率、网络环境,对这个系统的完备测试至少需要配置 3×3×3×3=81 种运行环境.

Table 1 A configuration table of the running environment of Android system

表 1 一个 Android 系统运行环境配置选项

Android 版本	内存	分辨率	网络
2.1	256M	320×240	Off
2.3	512M	640×480	Wi-Fi
4.0	1G	800×480	3G

若使用组合测试方法对该系统进行测试,令覆盖强度为 2,则只需在 9 种配置下对该系统进行测试.在表 2 中,任意两个参数的所有值组合都出现了至少一次,由定义 1 可知,表 2 就是表 1 所示系统对应的 2-way 覆盖表,可以表示为 $CA(9;2;3^4)$.

Table 2 Covering array of the running environment combinations of Android system

表 2 Android 系统环境组合覆盖表

配置	Android 版本	内存	分辨率	网络
t_1	2.1	256M	320×240	Off
t_2	2.1	512M	640×480	Wi-Fi
t_3	2.1	1G	800×480	3G
t_4	2.3	256M	640×480	3G
t_5	2.3	512M	800×480	Off
t_6	2.3	1G	320×240	Wi-Fi
t_7	4.0	256M	800×480	Wi-Fi
t_8	4.0	512M	320×240	3G
t_9	4.0	1G	640×480	Off

事实上,构造一个一般的覆盖表不是一件非常困难的事情,而真正的困难在于如何在比较短的时间内生成尽量小的覆盖表.这也是组合测试覆盖表生成问题所研究的重要内容.如何构造最小的覆盖表已被证明是一个 NP 完全问题[5].

对于求解覆盖表生成问题,目前已有的方法包括:(1) 数学方法,最为典型的是各种正交表构造方法,如构造拉丁方阵和正交拉丁方阵等,该种方法往往能够快速求得规模非常小的覆盖表,但其缺陷是限制条件多,不适合在各种实际情况下使用;(2) 贪心算法,如 AETG,TCG,DDA 等算法,这一类是最为常用的覆盖表生成方法,其优点是实现简单、算法时间开销较小,其缺点是算法比较容易陷入局部最优[6];(3) 搜索算法,如模拟退火、禁忌搜索、蚁群算法、遗传算法、粒子群算法等,这类算法虽然执行时间较长,但是往往可以计算出非常好的结果,很多覆盖表的已知最优解都是由这类方法求得的.

1.2 蚁群算法

蚁群算法是一种启发式算法,由意大利学者 Dorigo 首先提出[7].蚁群算法的灵感来源于对自然界中蚂蚁觅食行为的观察.蚂蚁在觅食过程中,总能够找到一条从蚁穴到食物之间的最短路径.虽然蚂蚁个体极其简单并且不具有智能,但是蚂蚁群体在觅食的过程中能够依靠一种叫做信息素的化学物质来实现群体智能.受此启发,蚁群算法模拟自然界蚂蚁觅食行为,以信息素为优化的线索,依靠蚂蚁的群体智能来解决各种组合优化问题.相关研究表明,蚁群算法在旅行商问题、车间作业调度问题等优化问题中均获得了很好的优化效果.

蚁群算法提出 20 余年以来,出现了各种不同的算法变种,其中最为广泛应用的是蚂蚁系统 AS(ant system)算法、蚁群系统 ACS(ant colony system)算法以及最大最小蚂蚁系统 MMAS(min-max ant system)算法[8].这些算法都是基于蚁群算法的基本原理的,只是在若干实现细节上有所不同.

2 利用蚁群算法生成覆盖表

目前已有利用蚁群算法一次生成一条测试用例(以下简称“一次一条测试用例”)的覆盖表生成算法,由于其生成方法使用了 AETG[9]算法的框架,我们将其称为 AETG-ACO 算法.Toshiaki 等人对遗传算法和蚁群算法在

覆盖表生成问题中的应用进行了研究,给出了一些覆盖表生成结果,并与其他算法,如 AETG[10]等进行了对比;Chen 等人对蚁群算法在组合测试变强度覆盖表生成中以及测试集优化问题中的应用[11,12]进行了研究;Nie 等人提出了基于搜索的组合测试,其中包括对蚁群算法生成覆盖表性能的研究[13].

本节将在已有工作的基础上,对蚁群算法一次一条测试用例生成覆盖表的方式进行深入的探究.首先,介绍蚁群算法一次一条测试用例生成覆盖表的基本原理;其次,结合 3 种主要的蚁群算法变种设计一个用蚁群算法生成覆盖表的框架,通过这个框架可以轻易地实现算法在各个变种之间的转变,同时也可以发现各种算法变种之间的异同点;最后,通过对各个决策点策略的详细介绍来具体展示 3 种算法变种之间的区别.

2.1 利用蚁群算法生成覆盖表的原理

利用蚁群算法来求解组合优化问题,就需要根据基本原理建立良好的模型,设计有效的算法.

一次一条测试用例的覆盖表生成过程就是通过每次执行蚁群算法来求得覆盖表中的一行,也就是求得一条测试用例,反复执行这个过程,直到所有要求被覆盖的组合对均被覆盖为止.而每次求得的这一条测试用例,恰好可以视为是一条蚂蚁移动的路径,如图 1 所示.图 1 表示的是一次一条测试生成算法的搜索空间,图中每一个点 $F_i(1\leqslant i\leqslant k)$对应着 SUT 中的参数 P_i,从点 F_i 引出的各条边分别对应着参数 P_i 的各个取值.每当蚂蚁移动到一个点上时,就需要按照一定策略选择一条边移动到下一个点,当蚂蚁移动到点 F_e 时,就完成了一次搜索.F_e 仅代表一只蚂蚁的构造结束,不对应 SUT 中的任何参数.此时,蚂蚁经过的路径对应一个候选解(即候选测试用例).蚂蚁在一个点时,根据概率公式选择路径.由于在早期信息素几乎没有差别,可以引入启发值来进行路径选择指导.最后路径选择公式由信息素和启发值共同决定.

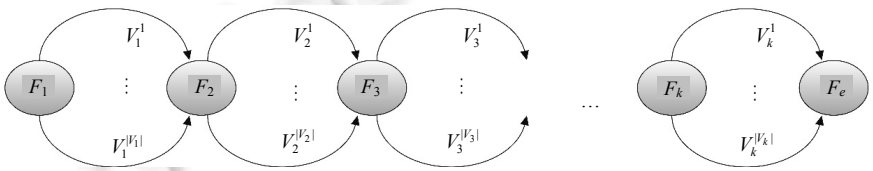


Fig.1 Search space of one-test-a-time covering array generation problem

图 1 一次一条测试用例覆盖表生成问题搜索空间

如果一只蚂蚁每次选择最上面一条路径,那么就得到如下所示的一条测试用例: $(V_1^1,V_2^1,...,V_k^1)$.

蚁群算法是一种群智算法,在每次迭代过程中,每只蚂蚁都会得到一个解,在整个迭代过程中将会产生很多候选解,需要一个函数来评估这些解的质量.我们称这样的函数为适应值函数 F.适应值一方面在迭代过程中控制蚂蚁留下信息素的多少,另一方面用来选择一个质量最好的解加入到覆盖表中来完成一次求解过程.在覆盖表生成问题中,设 S 为一条测试用例,则 $F(S)$为 S 覆盖的未被覆盖的组合对的数目.

在蚂蚁构造解的过程中或者构造完毕之后,蚂蚁根据解的质量对信息素进行更新,指导后来的蚂蚁进行路径选择[14].

2.2 利用蚁群算法生成覆盖表的框架

根据第 2.1 节的描述,可以给出利用蚁群算法生成覆盖表的基本框架,如算法 1 所示.在这个框架下,对不同决策点进行相应的选择和设计,就可以得到不同的算法变种[15].

算法 1. 利用蚁群算法生成覆盖表的框架 AETG-ACO.

输入:

1) 算法参数配置;

2) SUT 各个参数的取值个数($|V_1|,|V_2|,\ldots,|V_k|$)以及覆盖强度(t);

输出:满足输入要求的覆盖表(CA_set).

```
1: Begin
2:     Initial_UncoverList(); //初始化未覆盖组合对
```

```
3:    设置 CA_set 为空
4:    初始化算法参数配置:Nc←最大迭代次数;m←蚁群规模(蚂蚁数量)等
5:    While (uncover_list.size()>0){
6:        初始化信息素为一个常数;
7:        Compute_Heuristic();  //计算启发值                          (决策点 A)
8:         For (t=0; t<Nc; t++){
9:            For (ant h from 1 to m)
10:               For (node F_i from 1 to F_k){  //生成一个解 Sh
11:                   Select_MoveingEdge();  //选择移动路径          (决策点 B)
12:                     Local_Pheromones_Update();  //局部信息素更新策略  (决策点 C)
13:               }
14:           选择{Sh:1≤h≤m}中 F(Sh)值最大的解加入候选测试用例集 Candidate
15:           Pheromones_Update();  //信息素更新策略                  (决策点 D)
16:           Global_Pheromones_Update();  //应用全局信息素更新策略    (决策点 E)
17:       }
18:       选择 argmax_{S∈Candidate}F(S)添加到覆盖表 CA_set 中
19:       Update_UncoverList();  //更新未被覆盖组合对列表 uncover_list
20:    }
21:   输出覆盖表 CA_set
22: END
```

在算法 1 所描述的利用蚁群算法生成覆盖表框架中,算法的输入需要给出参数配置、覆盖强度和需要生成的覆盖表的相关信息,而输出则给出满足输入要求的覆盖表.算法中存在如下两个迭代过程:

1) 外层迭代(第 5 行~第 20 行).当未被覆盖组合对数目大于 0 时,令 m 只蚂蚁运算 Nc 次,并从中寻找一个最优解加入到 *CA_set* 中,直到未被覆盖组合对数目为 0 时循环终止,算法输出覆盖表.同时,每执行一次外层迭代就会计算一次启发值,并置信息素为初始值,在本文中,各种算法的信息素初始值固定;
2) 内层迭代(第 8 行~第 13 行).在每次迭代中,如图 1 所示,令每只蚂蚁从点 F_1 移动到 F_e,完成一次蚂蚁寻优过程,其中包含了与不同变种相关的 4 个决策点.

显然,算法的内层迭代是应用蚁群算法求解的过程,而外层迭代主要关注未被覆盖组合对 *uncover_list* 的大小.在外层迭代开始之前,初始化了算法参数,并初始化了未被覆盖组合对,函数 *Initial_UncoverList*()的作用就是统计 SUT 中 t-way 组合的个数,并标记这些组合的覆盖状态为“未覆盖”.

2.3 利用蚁群算法基本变种生成覆盖表

在算法 1 给出的利用蚁群算法生成覆盖表框架中,有 *A*,*B*,*C*,*D*,*E* 这 5 个决策点.在 *A* 决策点,可以采用不同的启发值生成策略.对 *A*,*B*,*C*,*D*,*E* 决策点进行选择和设计,可以得到不同的算法变种.本文主要对蚁群算法的变种进行研究,不考虑多种启发值生成策略,在算法中,使用单一的策略来进行计算.

决策点 *A*:启发值计算 *Compute_Heuristic*().

所谓启发值,就是具有某种直观意义并对算法起到指示作用的值,这种指示作用将避免算法盲目寻优,以加快算法收敛速度,好的启发值还有助于提高解的质量.一般来说,启发值使用贪心方式来构造.对于不同的组合优化问题,启发值的计算方式和所表示的意义有很大差别.对于覆盖表生成问题而言,一般使用的是由公式(1)所示的公式来计算启发值.

$$\eta_{i,j}=\frac{C_{i,\max}-C_{i,j}+1}{C_{i,\max}-C_{i,\min}+1} \tag{1}$$

公式(1)给出的是利用蚁群算法生成覆盖表的一种常用的启发值计算公式,其中各个符号的含义是:$\eta_{i,j}$ 代表

参数 P_i 的取值 j 所对应的启发值,$C_{i,j}$ 代表的是参数 P_i 的取值 j 在 CA_set 中出现的次数,$C_{i,\max}$ 代表的是参数 P_i 的所有取值在 CA_set 出现次数的最大值,而 $C_{i,\min}$ 则是相应的最小值.由于可能存在 $C_{i,\max}=C_{i,\min}=C_{i,j}(1\leqslant i\leqslant k, 1\leqslant j\leqslant |V_i|)$的情况,因此,公式(1)为分子分母同时加 1,避免出现分母为 0 的情形.

这样计算启发值具有非常直观的意义:当一个值在覆盖表中出现的次数越少,那么选择该值就有可能覆盖更多的未被覆盖对,从而得到比较好的解.此外,在生成一个解的迭代过程中,启发值始终不会发生改变.

图 2 具体演示了利用公式(1)计算启发值的过程,其中,覆盖表 CA(N;2;5^4)已经生成了 8 条测试用例,如图 2 中第 1 列所示.当生成第 9 条测试用例时,需要计算各个参数的每个取值的启发值,这里以第 1 个参数 P_1 为例.此时,P_1 的 5 个取值在已经生成的 8 条测试用例中,各个取值出现的次数如图 2 中第 2 列所示.因此,由公式(1)可以得到参数 P_1 的各个取值的启发值,如图 2 中第 3 列所示.

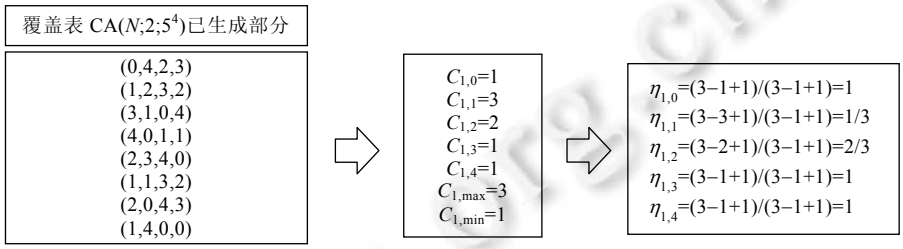


Fig.2 An example of calculating heuristic value

图 2 启发值计算公式示例

不同算法变种的差别不包括启发值的计算,于是我们将启发值的计算单独考虑.不同变种的算法体现在两个方面:路径选择策略和信息素更新策略.下面分别考虑各种变种算法对决策点 B,C,D,E 的选取和设计细节.

蚂蚁系统(AS 算法)是较早出现的蚁群优化算法,需要选择决策点 B 和 D.

路径的选择由启发值和信息素共同决定,选择各个边的概率如公式(2)所示.

$$Pr_{i,j}=\frac{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}{\sum_{k=1}^{|V_i|}\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}} \tag{2}$$

其中,各个符号的含义是:$Pr_{i,j}$ 表示第 i 个节点处选择第 j 条边的概率,$\eta_{i,j}$ 代表参数 P_i 的取值 j 所对应的启发值,$\tau_{i,j}$ 代表第 i 个节点上第 j 条边上信息素的含量.参数 α,β 用来调节启发值和信息素对概率造成影响的大小:当 $\alpha=0$ 时,概率只和启发值有关,算法将接近贪心算法;当 $\beta=0$ 时,概率只与信息素有关,算法很容易陷入局部最优,性能可能很糟糕.

当所有的蚂蚁都构建好路径之后,各条边上的信息素将会更新.信息素的更新分为两部分.

- 首先,所有边上的信息素都会减少一个常量因子的大小,只留下一部分.信息素蒸发如公式(3)所示.

$$\tau_{i,j}=\rho\tau_{i,j},\forall(i,j)\in E(0<\rho\leqslant 1) \tag{3}$$

其中,ρ 被称为信息素残留系数,E 为所有边的集合.参数 ρ 可以避免信息素无限积累,还能够让算法“忘记”一些不好的路径;

- 蒸发步骤完成之后,蚂蚁都会在各自经过的边上释放信息素,如公式(4)所示.

$$\tau_{i,j}=\tau_{i,j}+\sum_{k=1}^{m}\Delta\tau_{i,j}^{k},\forall(i,j)\in E \tag{4}$$

其中,$\Delta\tau_{i,j}^{k}$ 代表第 k 只蚂蚁在路径 (i,j) 上释放的信息素的大小,取值如公式(5)所示.

$$\Delta\tau_{i,j}^{k}=\begin{cases}Q\cdot|V_i|\cdot\dfrac{F(S_k)}{F(S^*)}, & (i,j)\in S_k\\ 0, & \text{else}\end{cases} \tag{5}$$

其中,Q 是一个常数,F 是适应值函数,S_k 为第 k 条候选解,S^*为本次迭代得到的最优解.

最大最小蚂蚁系统(MMAS)在 AS 的基础上进行了 4 项改进,需要选择决策点 B 和 D.

首先,MMAS 强调了最优路径的寻找,只有本次迭代过程中的最优蚂蚁或者从迭代开始到本次迭代的整个过程中的最优蚂蚁才释放信息素;但是这样会让某些边信息素增长过快,于是,MMAS 采用了另外一项改进:将信息素取值大小限制在一个区间内;然后将信息素的初始值设定为较大,配合较小的蒸发速率,让算法能够在最初探索到更多的路径;最后,每当 MMAS 在一定数量的迭代中不再有更优解出现或者算法出现停滞状态(收敛),则重新初始化信息素.

MMAS 路径选择公式和 AS 算法的路径选择公式一样,如公式(2)所示.

MMAS 的信息素更新仍然分为两个部分:首先对所有边上的信息素以固定的速率进行蒸发,更新公式如公式(3)所示;蒸发过程结束后,采用公式(6)让当前最优的蚂蚁释放信息素.

$$\tau_{i,j}=[\tau_{i,j}+\Delta\tau_{i,j}^{best}]_{\tau_{\min}}^{\tau_{\max}},\forall(i,j)\in E \tag{6}$$

其中,$\Delta\tau_{i,j}^{best}$ 为最优蚂蚁在路径(i,j)上释放的信息素的值,其表达式如公式(7)所示.

$$\Delta\tau_{i,j}^{best}=\begin{cases}Q\cdot F(S^*), & (i,j)\in S^*\\ 0, & \text{else}\end{cases} \tag{7}$$

其中,S^*为至今最优解.

映射 $[x]_{\min}^{\max}$ 用于将信息素的取值限制在区间范围内,取值如公式(8)所示.

$$[x]_{\min}^{\max}=\begin{cases}\max, & x>\max\\ \min, & x<\min\\ x, & \text{else}\end{cases} \tag{8}$$

蚁群系统(ACS)需要选择决策点 B,C,E,它和 AS 的不同体现在 3 个方面:首先,它采用了另外一种积极的路径选择规则;然后,信息素的挥发和释放动作都只在最优路径上进行;最后,蚂蚁每次走过一条边就会减少边上的信息素,增加其他边被搜索的可能性.

ACS 用公式(9)来进行路径选择:

$$L_i=\begin{cases}\arg\max\limits_{1\leqslant j\leqslant|V_i|}\{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}\}, & q\leqslant q_0\\ J, & \text{else}\end{cases} \tag{9}$$

其中,L_i代表蚂蚁在第 i 个节点所选择的路径的取值;q_0是一个常数,表示概率的阈值,q 是生成的随机数;J为根据公式(2)概率分布所生成的随机值.这种路径选择策略以 q_0的概率选择最佳移动方式,以 $1-q_0$的概率选择随机的方式.我们可以通过调节 q_0的大小来对搜索区间的大小进行控制.

ACS 信息素更新分为两种:全局更新和局部更新.

- 全局更新策略只允许当前最优蚂蚁或者是此次迭代最优的蚂蚁进行信息素的更新,无论是蒸发还是更新都只在最优路径上进行.这种方法不用让所有路径上的信息素都蒸发,从而降低了算法的时间开销.全局更新用公式(10)来描述.

$$\tau_{i,j}=\rho\tau_{i,j}+(1-\rho)\Delta\tau_{i,j}^{best},\forall(i,j)\in E \tag{10}$$

  注意,更新公式在 $\Delta\tau_{i,j}^{best}$ 前面增加了系数 $1-\rho$,这将信息素的值控制在原来信息素和释放的信息素之间.

- 除了全局更新策略外,ACS 还采用了局部更新策略.每次蚂蚁经过一条(i,j)边后,都会立刻用公式(11)更新该边上的信息素的值.

$$\tau_{i,j}=(1-\varphi)\tau_{i,j}+\varphi\tau_{init} \tag{11}$$

  其中,φ是常数,τ_{init} 是信息素的初始值.这样,每次蚂蚁经过一条边,该边上的信息素将会减少,从而增加了其他边被搜索到的机会.

在接下来的讨论中,我们利用蚁群算法以一次一条的方式生成覆盖表的各种变种算法统称为 AETG-ACO,将 AS,MMAS,ACS 以一次一条的方式生成覆盖表的变种算法分别称为 AETG-AS,AETG-MMAS,AETG-ACS.

3 利用蚁群算法生成覆盖表

3.1 实验设计

由于蚁群算法是一种多配置参数的演化搜索算法,且参数的不同取值对算法的性能会产生直接的影响,因此,参数配置调优是应用蚁群算法求解组合优化问题时必须考虑的因素.用蚁群算法生成覆盖表的方法继承了蚁群算法原有的参数,表 3 展示了蚁群算法生成覆盖表各种变种算法的参数及其常见取值范围.

Table 3 Parameters of ant colony optimization when generating covering arrays

表 3 蚁群算法生成覆盖表的配置参数

参数	类型	范围	AS	ACS	MMAS
信息素指数α	浮点型	$0<\alpha\leqslant 5$	√	√	√
信息素指数β	浮点型	$0<\beta\leqslant 5$	√	√	√
信息素残留指数ρ	浮点型	$0<\rho\leqslant 1$	√	√	√
蚂蚁数量m	整型	>0	√	√	√
迭代次数Nc	整型	>0	√	√	√
信息素初值τ_{init}	浮点型	$0\leqslant \tau_{init}\leqslant 10$	√	√	√
ACS路径选择概率q_0	浮点型	$0<q_0<1$	×	√	×
局部更新系数φ	浮点型	$0<\varphi<1$	×	√	×
信息素上界$\tau_{\max}$	浮点型	$\tau_{\min}<\tau_{\max}\leqslant 5$	×	×	√
信息素下界$\tau_{\min}$	浮点型	$0<\tau_{\min}<\tau_{\max}$	×	×	√

由表 3 可知,无论何种覆盖表生成算法变种,对其参数进行调优都会面临如下问题:(1) 参数配置空间庞大,若浮点数以 0.1 为步长,m 以 10 为步长且不大于 100,Nc 以 100 为步长且不大于 1 000,则由表 3 前 5 个参数构成的参数配置空间至少为 50×50×10×10×10=250000 个不同的参数配置;(2) 覆盖表的规模不同,生成覆盖表所需要的时间不同,一些覆盖表生成需要耗费大量时间;(3) 利用蚁群算法生成覆盖表是一种随机算法,为了评价其性能,对于每个覆盖表需要至少进行 30 次重复实验;(4) 需要选取多个各种类型的覆盖表进行实验.综合以上 4 点,在现有计算条件下对所有参数配置组合进行逐一验证的方法是行不通的.

本文介绍的组合测试不仅是一种软件测试方法,其本身就是一种实验设计方法.因此,本文使用二维实验(即 2-way 覆盖)进行参数间交互作用的检查,以确定蚁群算法生成覆盖表的各种变种的推荐参数配置[16].具体方案如下:

第 1 步:结合蚁群算法已有研究,选取一个参数配置作为基准,在此基础上为每个参数按照一定步长单独修改并进行实验,通过这样的方法先研究单个参数对算法性能的影响;

第 2 步:在第 1 步研究的基础上,为蚁群算法生成覆盖表的各种变种算法的每个参数选取若干生成结果较好的参数配置区间;

第 3 步:对各种算法变种,在由第 2 步得到的取值区间内选取若干值,并设计二维参数配置表;

第 4 步:对若干目标实验对象在二维参数配置表下运行蚁群算法生成覆盖表的各种变种算法,并根据结果选择一组推荐参数配置.

通过本节的研究工作,将解决如下 3 个问题:(1) 单个参数对各蚁群算法生成覆盖表的各种变种算法有怎样的影响;(2) 对于各种变种算法而言,能否给出推荐参数配置,使得该配置下算法的结果较其他参数配置更好;(3) 在各种算法的推荐配置下,比较各种变种算法的结果,探究何种算法变种更加适合求解覆盖表生成问题.

3.2 单参数配置优化

为了研究上述变种算法中单个参数对算法的影响,首先需要为各种算法变种寻找一组参数配置作为实验基准配置.单参数实验基准配置的选择基于以下两点:(1) 已有研究工作给出的蚁群算法参数配置建议;(2) 利用表 4 中的实验对象进行测试,选择比较好的配置.基于以上两点,本文选取了表 5 所示的实验默认配置.

Table 4 Experiment objects of single parameters optimization

表 4 单参数优化实验对象

MCA1(N;2;$10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1$)	CA2(N;2;8^8)	CA3(N;2;6^{10})
MCA4(N;2;$8^3 7^3 6^3 5^3$)	MCA5(N;2;$9^2 8^1 7^1 6^2 5^1 4^5$)	

Table 5 Base configuration of single parameters optimization experiments

表 5 单参数实验基准配置表

参数	AETG-AS	AETG-ACS	AETG-MMAS
信息素指数α	0.5	1.0	0.5
信息素指数β	0.3	0.3	0.3
信息素残留指数ρ	0.5	0.9	0.5
蚂蚁数量m	20	20	20
迭代次数Nc	200	200	200
信息素初值τ_{init}	0.4	0.4	1.0
ACS路径选择概率q_0	×	0.5	×
局部更新系数φ	×	0.5	×
信息素上界$\tau_{\max}$	×	×	5.0
信息素下界$\tau_{\min}$	×	×	1.0

表 4 给出了单参数优化配置实验使用的 5 个不同的覆盖需求,这些覆盖需求的计算量有大有小,有水平覆盖表也有混合覆盖表,对这 5 个表进行运算的结果能够在一定程度上代表各种变种算法配置的一般性.

由于 Nc 和 m 都是越大其实验效果越好,在本文的研究中,我们没有对这两个参数进行调节,将它们取为固定值.由于 MMAS 要求信息素初始值较大,我们将 AETG-ACO 的 τ_{init} 设置为 1.0,其他两种算法都为 0.4.

3.2.1 参数α与参数β对各种算法变种的影响

参数α与参数β均为路径选择概率公式的两个重要参数,信息素指数α的大小表示了信息素对路径选择的重要性,而启发值指数β表示启发值对路径选择的影响程度.一些关于蚁群算法的研究都给出了α与β的参考取值,然而对于不同的优化问题和具体的算法变种,这两个参数的具体最优取值并不相同.本节分别研究参数α和β对 3 种算法的影响:对每个参数,在基准配置下从 0.1 开始取值,并以 0.3 为步长分别对 MCA1,CA2,CA3,MCA4,MCA5 这 5 个覆盖表进行 30 次实验,并记录实验获得的最小值与均值.平均结果如图 3 所示.

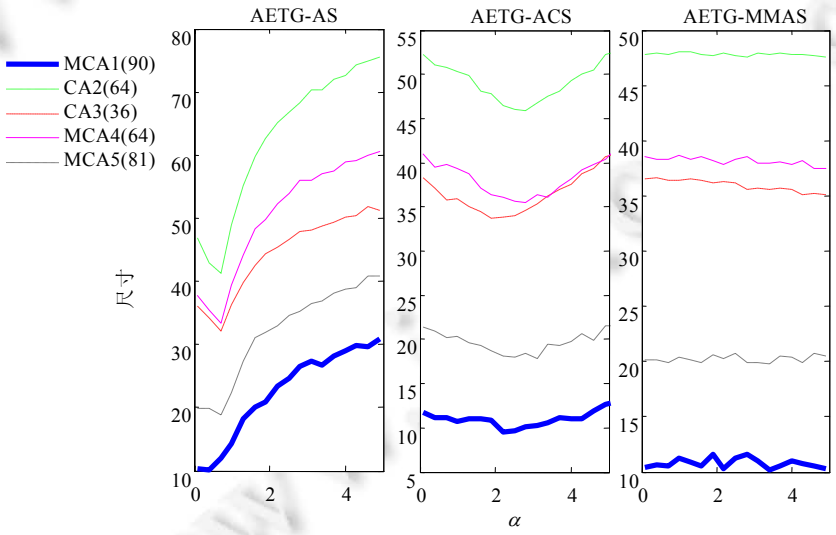


Fig.3 Average result of AETG-ACO when α varies

图 3 参数α不同取值下 AETG-ACO 的平均实验结果

需要说明的是,在本节,所有平均值结果图中的数据均为“增量值”,即覆盖表在基本尺寸上“膨胀”的部分,如

对MCA1,其二维覆盖表的基本尺寸(取值最多的两个参数取值个数的乘积)为10×9=90条测试用例,如果实际生成100条测试用例,则其“增量值”为100−90=10,而图中图例里括号内的数字给出了各个二维覆盖表的基本尺寸.

由以上参数α对3种算法变种的影响的实验结果可以得出如下结论:(1) 对于AETG-AS算法而言,参数α对算法结果的影响非常明显,且存在一个明显的最优取值区间,当α取值较大时算法结果变差,因此在二维实验中可以令α的取值区间为[0.1,1.0];(2) 对AETG-ACS算法而言,对规模较小的覆盖表MCA1、CA3,参数α对实验结果的影响有限,但是对于CA2、MCA4、MCA5这类生成规模相对较大的表,可以看出,当α的取值在[1.5,3.0]的区间内时算法平均结果较好,可以将这个区间作为下一步二维实验的参考配置区间;(3) 对于AETG-MMAS算法而言,参数α对算法结果的影响并不明显,这意味着参数α的选取较为自由,我们为α取值0.5、1.0、1.5、2.0作为二维实验的参考取值.

与对参数α所进行的实验类似,对参数β的实验也是在基准配置下,以0.3为步长,对3种算法变种下5个不同的覆盖表生成30次,并统计实验的最小结果与平均结果,平均结果如图4所示.

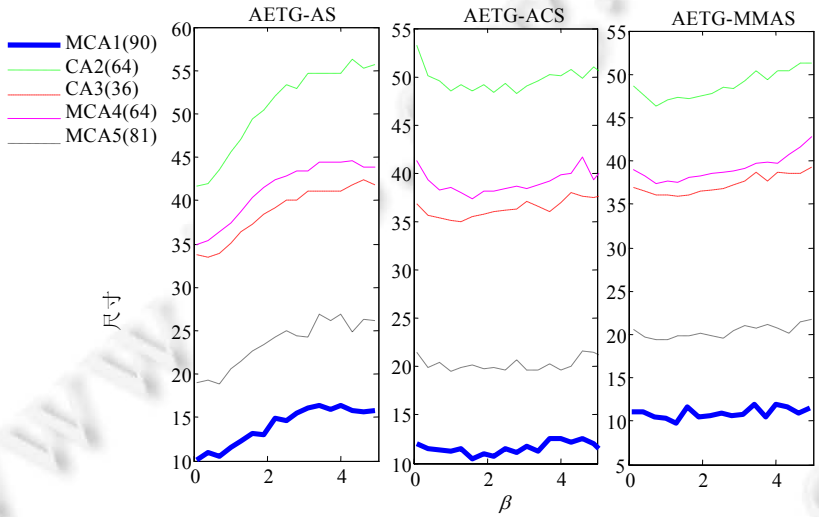


Fig.4 Average result of AETG-ACO when β varies

图4 参数β不同取值下AETG-ACO的平均实验结果

由参数β对3种AETG-ACO算法变种的实验结果可以得出如下结论:(1) 对于AETG-AS算法而言,参数β对算法结果的影响较为明显,当β的取值较大时算法结果变差,因此在二维实验中可以令β的取值区间为[0.1,1.0];(2) 对于AETG-ACS算法而言,虽然平均结果下不同取值的差异并不明显,但在[0.8,2.0]区间内生成的尺寸相对较小,可以将这个区间作为下一步二维实验的参考配置区间;(3) 对于AETG-MMAS算法而言,从平均值曲线上来看,当β的取值在[0.5,2.0]之间时,算法生成的覆盖表的平均规模低于其他区间,因此在二维实验中可以用[0.5,2.0]作为参数β的参考取值区间.

3.2.2 参数ρ对各种算法变种的影响

参数ρ是利用蚁群算法生成覆盖表的信息素残留系数,表示上一代信息素有多少能够保留到下一代,信息素的残留机制使得蚂蚁群体能够依靠已有的搜索信息指导进一步的解空间搜索.另一方面,信息素的挥发也使得蚂蚁群体依赖于已有信息素的程度减弱,这使得蚁群在搜索过程中能够保持“探索”能力,即保证随着算法的进行,蚁群仍然能够有机会探索其他解.本节通过实验研究参数ρ的不同取值对3种利用蚁群算法生成覆盖表变种的影响,实验平均结果如图5所示.

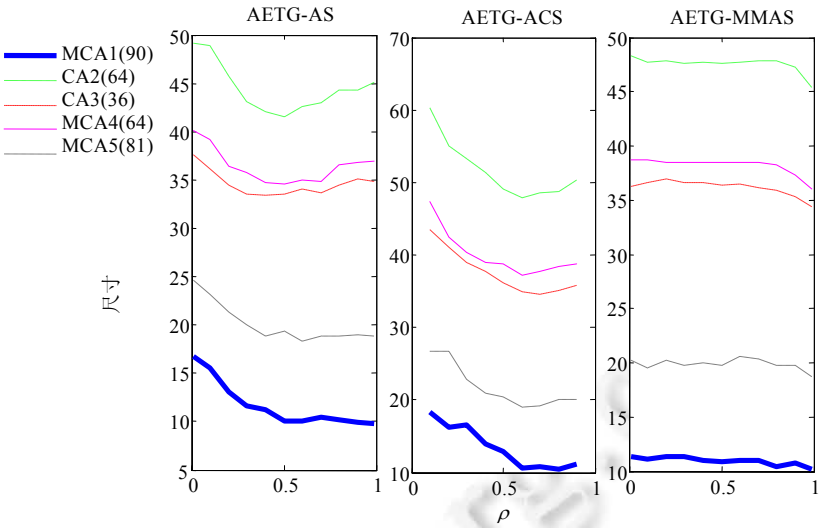


Fig.5 Average result of AETG-ACO when ρ varies

图 5 参数ρ不同取值下 AETG-ACO 的平均实验结果

从实验结果可以得出如下结论:(1) 对于 AETG-AS 算法而言,当信息素残留系数取 0.1 附近时,算法生成的平均规模较大,而当ρ取值较大时,算法生成覆盖表的平均规模有上升趋势,因此,在二维实验中,可令 AETG-AS 算法的信息素残留系数取[0.2,0.6]之间;(2) 对于 AETG-ACS 算法而言,当ρ的取值在 0.6 左右时,生成的覆盖表尺寸较小,因此在二维实验中,可令ρ取 0.5,0.6,0.7,0.8;(3) 对于 AETG-MMAS 算法而言,ρ对算法的影响较小,但当ρ的取值较大时(大于 0.8),算法的平均结果较好,因此,在二维实验中,可令ρ取 0.8,0.9,0.95,0.99 这 4 个值.

3.2.3 参数φ和 q_0 对 ACS 算法的影响

参数 q_0 是 ACS 路径选择方式概率,其值的大小表示了 AETG-ACS 算法选择两种路径转移方式的概率,而参数φ是 AETG-ACS 算法局部信息素更新系数,其值越小,一次局部信息素更新后,各个边上的信息素含量将更加接近信息素初始值τ_{init}.

对参数 q_0 与参数φ均为从 0.1 开始,并以 0.1 为步长,直到取值为 0.9 的 9 种情况下,分别记录 5 个覆盖表下生成平均结果如图 6 所示.

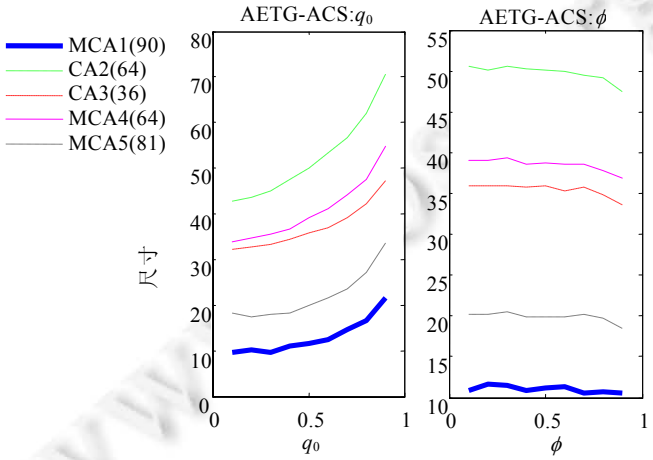


Fig.6 Average result of AETG-ACS when q_0, φ vary

图 6 参数 q_0,φ不同取值下 AETG-ACS 的平均实验结果

由实验结果可知,AETG-ACS 算法的参数 q_0 的取值越小,覆盖表生成的结果越好,在二维实验中,可令 q_0 取 0.01,0.05,0.1,0.2 作为实验参数.而对于参数φ而言,虽然参数在平均水平上表现比较一致,但在由表 6 给出的最小生成规模基础上,当φ取值较大时,生成的覆盖表的最小规模较小,因此在二维实验中,可令φ取 0.8,0.9,0.95, 0.99 作为实验参数.表 6 中,以“加粗、灰色背景”标记的是一行中的最小值.

Table 6 Minimun size of covering arrays when φ varies

表 6 参数 φ 配置实验最小覆盖表尺寸

φ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
MCA1	98	97	98	98	97	97	96	**95**	96
CA2	111	110	111	112	112	112	**109**	111	108
CA3	70	69	69	69	70	70	70	68	**67**
MCA4	100	100	100	100	99	99	99	99	**98**
MCA5	97	98	98	97	97	98	98	97	**95**

3.3 二维参数配置优化

为了对 AETG-ACO 算法的各个变种进行参数配置二维实验,首先需要确定实验对象.为了保证实验能够充分反映覆盖表的各种情形,在二维实验中,我们在表 4 的基础上扩展了 5 组覆盖表,构成表 7 所示的二维实验覆盖表.

Table 7 Experiment objects of pair-wise covering array for configurations

表 7 参数配置二维覆盖实验对象

MCA1(N;2;$10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1$)	CA6(N;2;4^{10})
CA2(N;2;8^8)	CA7(N;2;6^4)
CA3(N;2;6^{10})	CA8(N;2;3^{10})
MCA4(N;2;$8^3 7^3 6^3 5^3$)	MCA9(N;2;$7^1 6^1 5^1 4^5 3^8 2^3$)
MCA5(N;2;$9^2 8^1 7^1 6^2 5^1 4^5$)	MCA10(N;2;$10^2 9^2 5^1 3^8 2^2$)

3.3.1 AETG-AS 算法二维参数配置实验

根据单个参数对 AETG-AS 算法的影响的实验结果,得到各个参数比较理想的取值区间如下:α取 0.1~1.0 之间,β取 0.1~1.0 之间,ρ取 0.2~0.6 之间.为了进行 AETG-AS 参数配置二维实验,将上述取值区间内各取若干值见表 8,其中,蚂蚁个数取定值 20,最大迭代次数取定值 200.

Table 8 Value of parameters of AETG-AS

表 8 AETG-AS 参数取值表

符号	α	β	ρ	m	Nc
0	0.1	0.1	0.2	20	200
1	0.4	0.4	0.3	–	–
2	0.7	0.7	0.4	–	–
3	1.0	1.0	0.5	–	–
4	–	–	0.6	–	–

表 8 中,第 1 列给出的数值是对应参数在二维实验中的代表符号,为表 8 中的参数生成的参数配置二维覆盖表见表 9.

Table 9 Pair-Wise covering array of AETG-AS configurations

表 9 AETG-AS 参数配置二维覆盖表

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20
α	3	2	2	1	0	1	3	0	2	3	0	1	0	3	1	2	1	0	3	2
β	2	0	1	1	3	0	3	2	2	1	0	2	1	0	3	3	1	0	3	2
ρ	0	4	0	3	2	1	4	1	3	1	3	4	4	2	0	1	2	0	3	2

最后,我们利用所生成的参数配置进行实验,得到的结果见表 10.

AETG-AS 的二维实验在 20 组不同的参数配置下,为 10 个不同的覆盖表分别进行了 30 次生成实验,表 10 给出的是每个参数配置下每个覆盖表生成的最小结果、最小结果总和以及总平均值.

Table 10 Result of AETG-AS pair-wise configuration experiments

表 10 AETG-AS 二维实验结果

	MCA1	CA2	CA3	MCA4	MCA5	CA6	CA7	CA8	MCA9	MCA10	总和	平均值
C1	100	110	72	101	102	33	43	**18**	51	118	748	780.666 7
C2	94	100	66	**93**	94	31	**41**	**18**	**48**	113	**698**	**724.833 3**
C3	98	103	**65**	95	97	31	42	**18**	50	115	714	739.4
C4	96	103	68	94	94	31	42	**18**	**48**	111	705	734.633 3
C5	95	**95**	68	96	**93**	32	**41**	**18**	**48**	112	**698**	730.6
C6	96	103	68	96	94	32	42	**18**	**48**	113	710	738.266 7
C7	100	108	70	99	99	33	44	**18**	50	118	739	765.066 7
C8	**93**	105	70	97	95	31	42	**18**	48	112	711	739.633 3
C9	97	102	67	94	96	32	42	**18**	49	113	710	735.666 7
C10	99	108	68	98	99	32	43	**18**	50	118	733	761.666 7
C11	97	106	70	98	95	32	42	**18**	49	**110**	717	744.166 7
C12	95	104	67	96	96	32	**41**	**18**	**48**	112	709	735.7
C13	96	106	68	97	95	32	42	**18**	49	112	715	740.666 7
C14	98	105	67	95	96	31	42	**18**	50	113	715	743.833 3
C15	98	104	67	96	96	**30**	43	**18**	50	113	715	745.633 3
C16	99	107	68	97	97	32	43	**18**	50	117	728	754.6
C17	97	103	67	96	94	32	42	**18**	**48**	111	708	734
C18	96	106	69	96	95	32	**41**	19	49	112	715	744
C19	98	109	70	101	100	32	43	**18**	51	117	739	771.9
C20	95	103	66	96	96	31	42	**18**	50	115	712	739.133 3

表 10 中,以“加粗、灰色背景”标记的是一列中的最小值,从表中的结果可以得到如下结论:

1) 参数配置对 AETG-AS 的实验结果有一定的影响,一方面,部分参数配置之间的差异非常明显,如参数配置 C1 与 C2 代表了两个极端,两者的最小值总和相差 50;另一方面,C2 与 C5 两者仅在总均值上存在差异.此外,参数配置 C4 与 C16 的结果也相对较好;
2) 对某些覆盖表而言,参数配置对最小值的影响较弱,如对覆盖表 CA8,AETG-AS 仅在一个配置 C18 下多生成了一条测试用例,而考虑到算法的随机性,这样的影响几乎可以忽略不计.另一方面,在某些覆盖表,如 MCA1 上,各个参数配置对实验结果的影响较为明显;
3) AETG-AS 参数配置二维实验结果比较好的配置为 C2={α=0.7,β=0.1,ρ=0.6,m=20,Nc=200}以及 C5={α=0.1,β=1.0,ρ=0.4,m=20,Nc=200},考虑到参数配置 C2 的总均值更小,因此本文推荐使用 C2 配置.从平均结果比较好的配置 C2,C4,C5,C17 来看,AETG-AS 算法中ρ应该设置得较小,但也不宜太小.也就是说,AETG-AS 算法中信息素需要保持中等的蒸发速率才会有比较好的性能.分析其原因,AS 算法是最基本的一种蚁群算法,需要同时考虑算法的收敛性和搜索区间的大小,而蒸发速率越大,算法能够搜索到的解越多,收敛会越慢,综合考虑两个因素,蒸发速率设置在 0.5 左右比较好.

3.3.2 AETG-ACS 算法参数配置二维实验

根据第 3.2 节对单个参数影响 AETG-ACS 算法的实验结果,得到 AETG-ACS 各个参数的参考配置区间如下:α取 1.5~3.0 之间,β取 0.8~2.0 之间,ρ取 0.5~0.8 之间,q_0取 0.1~0.2 之间,φ取 0.8~0.99 之间.为了进行 AETG-ACS 参数配置二维实验,将上述取值区间内各取若干值见表 11,其中,蚂蚁个数取定值 20,最大迭代次数取定值 200.

Table 11 Value of parameters of AETG-ACS

表 11 AETG-ACS 参数取值表

符号	α	β	ρ	q_0	φ	m	Nc
0	1.5	0.8	0.5	0.01	0.8	20	200
1	1.8	1.1	0.6	0.05	0.9	–	–
2	2.1	1.4	0.7	0.1	0.95	–	–
3	2.4	1.7	0.8	0.2	0.99	–	–
4	2.7	2.0	–	–	–	–	–
5	3.0	–	–	–	–	–	–

表 11 中,第 1 列给出的数值是对应参数在二维实验中的代表符号,为表 11 中的参数生成的参数配置二维覆盖表,见表 12.对 AETG-ACS 的二维实验在 30 组不同的参数配置下,为 10 个不同的覆盖表分别进行了 30 次

生成实验,表 13 给出的是每个参数配置下每个覆盖表生成的最小结果、最小结果总和以及总平均值.

Table 12 Pair-Wise covering array of AETG-ACS configurations

表 12 AETG-ACS 参数配置二维覆盖表

	α	β	ρ	q_0	φ		α	β	ρ	q_0	φ
C1	0	4	0	3	2	C16	1	1	1	0	1
C2	1	2	2	2	0	C17	3	2	0	0	3
C3	3	0	1	1	0	C18	0	2	3	1	0
C4	4	3	3	0	3	C19	4	0	0	2	2
C5	0	1	2	1	3	C20	4	4	2	1	0
C6	2	4	3	2	1	C21	0	0	2	0	1
C7	5	3	0	1	1	C22	5	1	3	3	0
C8	4	2	1	3	1	C23	1	4	0	1	2
C9	5	0	2	0	2	C24	2	0	2	3	3
C10	2	1	0	0	0	C25	0	3	1	2	1
C11	1	0	3	3	3	C26	3	4	0	0	1
C12	3	1	3	2	2	C27	2	2	3	2	2
C13	5	4	1	2	3	C28	1	3	1	0	1
C14	2	3	1	1	2	C29	4	1	0	1	3
C15	3	3	2	3	0	C30	5	2	1	3	0

表 13 中,以“加粗、灰色背景”标记的是每组的最小值,从表中的结果可以得到如下结论:

1) 参数配置对 AETG-ACS 的实验结果有明显的影响,每组生成最小值总和的最大差异为 90;

2) C21 配置无论在最小覆盖表规模总和、总均值还是各个覆盖表最小规模上都表现出优于其他配置,因此,AETG-ACS 参数配置二维实验的推荐参数配置为 C21={α=1.5,β=0.8,ρ=0.7,q_0=0.01,φ=0.9,m=20,Nc=200}.此外,C2,C5,C18 的平均规模也比较小,而它们的共性在于α的取值大于β,ρ的取值稍微大一点,但是也不能太大,q_0取值要小.这表明,AETG-ACS 在选择路径时,信息素占的比重要大于启发值,信息素蒸发的速度要较慢,选择最佳移动方式的概率要较小.蚁群系统是一种更加强调信息素的变种,虽然采用了更加积极的路径选择策略,但是大多数情况下还是需要用概率选择的策略.

Table 13 Result of AETG-ACS pair-wise configuration experiments

表 13 AETG-ACS 二维实验结果

	MCA1	CA2	CA3	MCA4	MCA5	CA6	CA7	CA8	MCA9	MCA10	总和	平均值
C1	103	114	73	102	102	32	42	19	52	123	762	790.526
C2	99	104	67	95	97	31	42	18	**47**	117	717	747.341 8
C3	98	109	69	95	98	32	42	18	51	119	731	759.957 2
C4	98	109	69	95	96	31	**41**	19	50	116	724	753.01
C5	98	104	66	95	96	32	**41**	18	48	117	715	742.378 6
C6	98	109	69	99	98	32	43	18	49	120	735	756.584 2
C7	107	119	74	106	107	33	44	19	52	125	786	806.841 5
C8	106	119	75	106	106	33	44	18	51	125	783	804.878
C9	98	111	69	99	99	32	43	19	48	118	736	767.244 1
C10	100	109	70	97	99	32	42	19	49	119	736	762.656 7
C11	98	108	67	97	97	32	42	18	50	115	724	751.037 4
C12	102	109	68	97	99	31	42	19	51	118	736	759.521 6
C13	100	114	73	103	102	33	44	18	51	121	759	787.257 8
C14	103	112	71	103	103	32	43	19	50	120	756	784.465
C15	102	113	69	101	100	33	42	18	50	122	750	777.009 6
C16	100	108	69	98	100	32	42	18	50	117	734	758.942 5
C17	101	113	72	99	100	33	42	18	50	119	747	775.323 2
C18	97	104	66	95	93	31	42	**17**	**47**	115	707	734.346 6
C19	104	116	71	102	103	32	44	19	52	123	766	792.299
C20	101	113	70	100	100	32	43	18	50	121	748	775.558 9
C21	**95**	**101**	**65**	**92**	**92**	**30**	42	18	48	**113**	**696**	**730.92**
C22	99	109	68	99	99	31	**41**	18	49	117	730	759.040 7
C23	103	113	73	103	104	33	44	19	51	124	767	788.596 8
C24	100	112	70	99	97	32	43	18	50	120	741	771.108 5
C25	99	111	70	99	99	32	43	18	50	119	740	763.910 5
C26	106	121	73	105	102	32	44	19	51	125	778	800.658 6
C27	100	109	67	96	98	32	42	18	49	118	729	759.070 1
C28	101	112	70	100	100	32	43	18	48	122	746	772.259 9
C29	102	111	72	100	99	32	44	18	50	119	747	777.564 8
C30	104	118	73	103	103	33	44	19	50	125	772	801.334

3.3.3 AETG-MMAS 算法参数配置二维实验

根据第 3.2 节对单个参数影响 AETG-MMAS 算法的实验结果,得到各个参数的参考参数配置区间如下:β 取 0.5~2.0 之间,ρ取 0.8~0.99 之间.为了进行 AETG-MMAS 参数配置二维实验,将上述取值区间内各取若干值,见表 14,其中,蚂蚁个数取定值 20,最大迭代次数取定值 200.

Table 14 Value of parameters of AETG-MMAS

表 14 AETG-MMAS 参数取值表

符号	α	β	ρ	m	Nc
0	0.5	0.5	0.8	20	200
1	1.0	0.8	0.9	–	–
2	1.5	1.1	0.95	–	–
3	2.0	1.4	0.99	–	–
4	–	1.7	–	–	–
5	–	2.0	–	–	–

表 14 中,第 1 列给出的数值是对应参数在二维实验中的代表符号,为表 14 中的参数生成的参数配置二维覆盖表见表 15.

Table 15 Pair-Wise covering array of AETG-MMAS configurations

表 15 AETG-MMAS 参数配置二维覆盖表

	α	β	ρ		α	β	ρ		α	β	ρ
C1	1	0	0	C9	3	4	0	C17	1	5	1
C2	2	0	1	C10	0	5	0	C18	0	4	3
C3	0	3	2	C11	2	1	2	C19	2	3	0
C4	0	1	3	C12	2	5	3	C20	1	1	1
C5	1	4	1	C13	3	5	2	C21	2	1	2
C6	3	0	3	C14	1	3	3	C22	0	0	2
C7	3	3	1	C15	2	2	0	C23	3	2	3
C8	1	2	3	C16	0	2	1	C24	3	1	0

对 AETG-MMAS 的二维实验在 24 组不同的参数配置下,分别为 10 个不同的覆盖表进行了 30 次生成实验,表 16 给出的是每个参数配置下每个覆盖表生成的最小结果、最小结果总和以及总平均值.

Table 16 Result of AETG-MMAS pair-wise configuration experiments

表 16 AETG-MMAS 二维实验结果

	MCA1	CA2	CA3	MCA4	MCA5	CA6	CA7	CA8	MCA9	MCA10	总和	平均值
C1	97	106	68	96	94	31	41	18	48	113	712	738.266 7
C2	95	102	63	92	94	**30**	42	18	47	112	695	723.066 7
C3	95	103	66	95	95	31	42	18	48	113	706	731.7
C4	95	103	66	94	93	32	42	18	48	**111**	702	730.366 7
C5	94	103	63	93	93	**30**	42	**17**	47	114	696	729.266 7
C6	**92**	94	**61**	**86**	**91**	**30**	42	18	47	112	**673**	**702.966 7**
C7	94	100	63	89	92	31	41	18	49	114	691	718.466 7
C8	95	99	63	92	92	31	42	18	47	**111**	**690**	717.4
C9	95	103	65	94	95	31	41	18	49	112	703	732.566 7
C10	94	106	68	95	95	31	42	**17**	48	113	709	740
C11	95	96	62	88	92	31	42	18	48	113	685	710.366 7
C12	94	95	62	87	91	31	43	**17**	48	114	682	709.433 3
C13	94	98	62	**86**	92	31	41	**17**	48	114	683	711.9
C14	93	98	63	90	92	31	42	18	47	112	686	715.3
C15	94	104	65	94	94	31	43	18	**46**	113	702	733.2
C16	96	104	67	96	94	**30**	42	17	49	113	708	736.166 7
C17	96	103	64	93	96	31	41	18	49	113	704	729.560 9
C18	96	103	66	95	95	31	42	18	48	**111**	705	730.666 7
C19	95	104	65	94	95	31	42	17	49	113	705	733.166 7
C20	97	103	64	94	93	31	42	19	47	114	704	728.5
C21	94	98	62	89	93	30	42	18	47	**111**	684	712.166 7
C22	95	105	67	95	94	31	41	18	48	113	707	733.833 3
C23	94	**93**	62	**86**	93	**30**	**39**	**17**	48	113	675	704.466 7
C24	97	103	66	94	95	31	41	18	48	114	707	732.5

表 16 中,以“加粗、灰色背景”标记的是每组的最小值,从表中的结果可以得到如下结论:

1) 参数配置对 AETG-MMAS 的实验结果有一定的影响,从最小值以及平均值来看,有 3 组参数配置取得了较好的实验结果:C6、C12、C23,其他结果与这 3 组结果有较为明显的差异;
2) 在 C6、C12、C23 这 3 组参数配置中,C6 配置无论在最小覆盖表规模总和、总均值还是各个覆盖表最小规模上都优于其他配置,因此,AETG-MMAS 参数配置二维实验的推荐参数配置为 C6={α=2.0, β=0.5,ρ=0.99,m=20,Nc=200}.从实验结果我们可以发现,C6、C12、C23 中的α和ρ的取值都比较大,这表明 AETG-MMAS 当信息素作用较大、信息素挥发较慢时性能会比较好.分析其原因,最大最小蚂蚁系统强调了信息素的收敛性,于是在选择边时,信息素所起的作用应该大于启发值,为了让算法搜索到更多的解,信息素蒸发速度要缓慢些.

3.4 3种算法在推荐配置下结果的比较

为了研究这 3 种算法中哪一种生成的覆盖表的规模比较小,我们将它们在推荐配置下的实验结果进行了对比,见表 17.

Table 17 Comparison of three kind of algotithms

表 17 3 种算法生成覆盖表的尺寸对比

	MCA1	CA2	CA3	MCA4	MCA5	CA6	CA7	CA8	MCA9	MCA10	总和	总均值
AETG-AS	94	100	66	93	94	31	41	18	48	113	698	725
AETG-ACS	95	101	65	92	92	30	42	18	48	113	696	731
AETG-MMAS	92	94	61	86	91	30	42	18	47	112	673	703

其中每一行代表该算法生成 30 次指定规格的覆盖表的最小值,总和表示生成最小表的尺寸之和,总均值表示 30 次生成 10 个覆盖表的尺寸总和的平均值.由此我们可以看出,AETG-MMAS 生成覆盖表的效果最好.

4 基于解结构调整的蚁群算法生成覆盖表的方法

4.1 为何要调整解的结构

一次一条测试用例生成技术确实是一种简单、有效的覆盖表生成方法.但在生成的过程中,被选定的测试用例将不会发生变化,这样会让算法存在局限性.首先我们观察如表 18 所示的例子.假设 SUT 有 4 个参数,每个参数都有 3 个取值,分别用 0,1,2 表示,覆盖强度是 2.我们用一次一条的生成方法得到左边的 9 条测试用例.这个表并不满足覆盖需求,但是我们如果将表中黑色背景位置上的值用右边表中对应的值替换,就能得到一个满足要求的覆盖表.若继续使用一次一条的生成方式,则更多的测试用例需要被加入到表 18 中.

Bryce 和 Colbourn 等人提出一个问题:一次一条测试用例生成方式能否生成最小的覆盖表?他们对这个问题进行了实验.实验步骤如下:首先穷举出所有可能的测试用例,接着每次选择覆盖最多未覆盖对的测试用例,将其加入到测试用例集,直到得到一个覆盖表.如果有多条测试用例都覆盖最多的未被覆盖对,那么等概率地从中选择一条测试用例.对每个覆盖需求,做 100 次重复的实验[17].实验结果显示,对于覆盖需求 CA(N;4;2^8),这种实验方法得到的覆盖表规模在 31~37 之间,而当前已知的最优覆盖表的尺寸为 24.对于其他的覆盖需求,这种情况也非常常见.由此我们可以看出,一次一条测试用例生成方式是存在局限性的,下面我们对此进行分析.

Table 18 Disadvantage of one-test-at-a-time method

表 18 一次一条生成方法的局限性

	参数 1	参数 2	参数 3	参数 4		参数 1	参数 2	参数 3	参数 4
t_1	0	0	0	0	t_1	0	0	0	0
t_2	0	1	1	2	t_2	0	1	1	2
t_3	0	2	2	1	t_3	0	2	2	1
t_4	1	0	1	1	t_4	1	0	1	1
t_5	0	1	2	0	t_5	1	1	2	0
t_6	1	1	1	2	t_6	1	2	0	2
t_7	2	0	2	2	t_7	2	0	2	2
t_8	2	1	0	1	t_8	2	1	0	1
t_9	2	2	1	2	t_9	2	2	1	0

覆盖表生成的目标是得到一个覆盖全部未被覆盖对的表.如果想要得到最优表,就需要在所有可能出现的测试用例集中选出行数最少的覆盖表.也就是说,我们的优化目标其实是找到满足公式(12)的表 *TS*.

$$TS = \underset{ts \in CA}{\operatorname{argmin}}\{Size(ts)\} \tag{12}$$

其中,*Size*(*T*)表示表 *T* 的行数,*CA* 是所有满足覆盖需求的表的集合.

然而,我们所用的一次一用例生成方法的目标是每次找到一条最好的测试用例加入到表中,直到所得到的表满足覆盖需求成为一个覆盖表.实际上,一次一用例的生成方法将最终目标进行了分解,将待寻找的目标 *TS* 分解成为寻找 *TS* 的行.每次优化目标变为满足公式(13)的测试用例 *TC*.

$$TC = \underset{tc \in T\text{-all}}{\operatorname{argmax}}\{fit(tc)\} \tag{13}$$

其中,*T*-all 表示测试用例所有可能取值的集合,*fit*(*tc*)表示测试用例 *tc* 覆盖的未被覆盖对的个数.我们发现,这样的变化使得优化目标从一个无限取值范围缩小到的一个有限取值范围.这的确大大降低了优化开销.但是 Bryce 和 Colbourn 等人的实验告诉我们,即使一次一用例的生成方式每次都找到了最好的测试用例,到最后得到的覆盖表也不一定是最小的覆盖表.使用上述贪心策略容易让结果陷入局部最优而找不到比较好的结果.于是,我们将优化目标还原成最初的目标,将优化目标从得到一条测试还原成得到一个尺寸足够小的覆盖表.

4.2 调整解的结构后的覆盖表生成算法

利用蚁群算法以一次性生成一个表的覆盖表生成方法(简称“整表演化”)主要有 3 个决策点,分别为表的演化策略,初始表的生成和表的演化位置的选择.

4.2.1 表的演化策略

表的演化方式有两种.其一是最初选择一个行数比较小的表,然后每次添加一条测试用例,让所得到的表通过整表演化的方法覆盖更多未被覆盖对,直到得到一个覆盖表为止.另一种方法则是反其道而行之,最初生成一个比较大的表(一般是覆盖表),然后每次减少一条测试用例,直到再也得不到更小的覆盖表为止.由于事先估计所要生成的覆盖表的大小有一定的困难(如果估计值过大,那么会得到比较大的覆盖表,这与我们的需求相矛盾;如果估计值过小,那么将要进行大量的计算,影响算法的性能),于是我们采取了后一种生成办法,即首先生成一个较大的表,然后慢慢减少测试用例的条数再进行演化,直到得不到更小的覆盖表为止.本质上来说,这种整表演化的方法是一种后优化的方法,用于优化一个覆盖表.

4.2.2 初始表的生成

对于整表生成算法而言,首先需要解决的问题是如何确定一个 $N\times k$ 的初始表,其中,*k* 由 SUT 参数个数来确定,而 *N* 需要指定.生成初始表主要有两种方式.

1) 方式 1:随机表法.此种方法需要手动指定 *N* 的值,*N* 的确定可以依赖于已有的经验或各种相关研究资料中的数据,在 *N* 确定之后,可以根据 SUT 参数的形式为 $N\times k$ 的表中每个位置生成一个合法的随机值.
2) 方式 2:覆盖表法.由于随机表法需要手动指定 *N*,为实际使用带来不便,因此可以为整表演化算法集成一个覆盖表生成算法,用这种算法首先生成一个覆盖表作为初始表.出于对时间和尺寸的考虑,我们选择了贪心算法 AETG.

4.2.3 表的演化位置

给定一个表,如何对这个表采用蚁群算法进行演化从而提高表的适应度呢?这就需要确定表的演化位置.位置的选择策略可以有多种,下面介绍两种选择方式.

1) 方式 1:随机选择法.此种方法是最直接、最简单的一种方法.但是每次选择多少个位置进行演化合适呢?为了解决这一问题,可以指定选择位置的个数等于参数的个数,在表的每一列随机选择一个位置.但是经过尝试我们发现,这种方式在实际中的应用效果不是很好.
2) 方式 2:灵活位置法.由于随机选择的方式具有一定的盲目性.我们引入了随机后优化算法中“灵活位置”这一概念[18].

定义 3. 对于表中的一个位置,如果修改它的取值不会导致已经被覆盖的组合变为“未被覆盖”,那么我们称

这个位置为“灵活位置”.

性质. 表中的某一行某一列的位置是灵活位置当且仅当该位置所在的测试用例中,包含该位置的所有 t 维组合都被表中的其他测试用例所覆盖.

下面我们给出一个例子来说明灵活位置的概念以及寻找方法.

在图 7 中,左边是一个测试用例集.我们对 $T1$ 这条测试的灵活位置用例进行研究.首先,我们列出所有被 $T1$ 覆盖的组合以及它们在测试用例集中被覆盖的情况,见表 19.

$T1$:	1 2 3 3
$T2$:	1 2 1 2
$T3$:	2 1 3 1
$T4$:	2 2 3 1
$T5$:	1 2 1 3

$T1$:	1 * 3 3
$T2$:	* * 1 2
$T3$:	2 1 3 1
$T4$:	2 2 3 1
$T5$:	* * 1 3

Fig.7 An example of flexible position

图 7 灵活位置的例子

Table 19 All the combinations $T1$ covers

表 19 $T1$ 覆盖的所有组合

	$T1$ 所覆盖的组合	其他包含该组合的测试用例
c1	1 2 – –	$T2$: **1 2** 1 2
c2	1 – 3 –	无
c3	1 – – 3	$T5$: **1** 2 1 **3**
c4	– 2 3 –	$T4$: 2 **2 3** 1
c5	– 2 – 3	$T5$: 1 **2** 1 **3**
c6	– – 3 3	无

接着,我们分别观察 $T1$ 的 4 个位置的取值.包含第 1 个位置的组合有 c1,c2,c3,由于 c2 没有被其他测试用例覆盖,故 $T1$ 的第 1 个位置不是灵活位置,修改第 1 个位置上的值会让 c2 从已被覆盖变为未被覆盖.包含第 2 个位置的组合有 c1,c4,c5,但是它们还分别被 $T2$,$T4$,$T5$ 所覆盖,故 $T1$ 的第 2 个位置为灵活位置,我们可以将该位置标记为“*”.c6 未被除 $T1$ 之外的其他测试用例所覆盖,但是 c6 包含 $T1$ 的第 3,4 个位置,故 $T1$ 的第 3,4 个位置都不是灵活位置.如果将这个步骤重复下去,我们可以得到一个表中所有的灵活位置.最终结果如图 7 右边的测试用例集所示.

在求解灵活位置的过程中,我们只关心一条测试用例中包含某个位置的所有组合是否还被其他测试用例所覆盖,而不关心它们具体被哪些测试用例覆盖.如果一个组合只被当前的测试用例所覆盖,那么它在测试用例集中被覆盖的次数为 1.这可以成为我们寻找灵活位置的一个线索.于是我们可以先扫描一次测试用例集,得到测试用例集所覆盖的所有组合被覆盖的次数,然后再遍历表中的所有位置,依次判断每个位置是否为灵活位置.如果包含某个位置的所有组合被覆盖的次数都大于 1,那么这个位置为灵活位置.具体过程如算法 2 所示.

算法 2. 灵活位置 *FlexiblePosition*.

输入:

1) 一个 $N\times k$ 的表 *Table*($N\times k$);

2) 覆盖强度 t;

输出:标出所有灵活位置的 *flexiblePointTable*.

```
1: Begin
2:     计算 Table 中所有 t 维组合被 Table 覆盖的次数;
3:     flexiblePointTable=Table;
4:     For (testCase in flexiblePointTable)
5:         For (value in testCase)
```

```
6:                    If (testCase 所有包含 value 的 t-way 组合被覆盖的次数大于 1)
7:                        将 flexiblePointTable 中 testCase 取值为 value 的位置修改为“*”
8:      return flexiblePointTable;
9: End
```

算法 2 中,*testCase* 代表 *Table* 中的一条测试用例,而 *value* 代表 *testCase* 中一个位置的取值(形如 $P_{i,j}$). *flexiblePointTable* 最终为标记了所有灵活位置之后的表.

我们先求出所有灵活位置,然后利用蚁群算法在灵活位置上进行演化,这样会更有助于覆盖更多未被覆盖对.修改一个灵活位置可能会让原先本来是灵活位置的点变成非灵活位置,但是修改同行中任意多个灵活位置的值不会将剩下的灵活位置变成非灵活位置.根据这一点,我们将蚁群算法的演化目标设置为灵活位置最多的那一行上的所有灵活位置.在图 7 所示的例子中,我们可以在 *T*2 或 *T*5 上进行演化.

4.2.4 基于蚁群算法的整表演化算法 CA-ACO

我们仍然使用图 1 所示的路径图,但是图中各个参数点的含义发生了变化.在一次一条测试用例的生成方式中,每一个点代表一个参数,而在整表生成算法中,F_i 代表的则是一个表中的某个位置.由于路径的含义发生了变化,适应值函数也会随之变化.此时,某条路径的适应值变为采用某条路径能够增加的未被覆盖对的数量.此外,启发值的计算公式还是使用式(1)进行计算.

整体的过程如算法 3 所示.其中,*FlexiblePosition*(*Table*)即为算法 2 中求 *Table* 中灵活位置的函数. *isCoveringArray*(*Table*)用于判断 *Table* 是否为一个覆盖表.算法中有两个循环,外层 while 循环每次删除一条测试用例,里层 For 循环用于将删除测试用例之后的表演化为覆盖表.一旦达到最大迭代次数或者是不再存在灵活位置,那么就停止算法,返回覆盖表.

算法 3. 整表演化 CA-ACO.

输入:

1) 蚁群算法参数配置;
2) SUT 各个参数的取值个数($|V_1|,|V_2|,\ldots,|V_n|$)以及覆盖强度(*t*);
3) 最大尝试次数 max*Count*;

输出:满足要求的覆盖表(*CA*).

```
1: Begin
2:     生成覆盖表 CA   //可利用 AETG,IPO 等贪心算法
3:     While (True){
4:         Table=任意删除一条测试用例所得到的表;
5:         For (i=0; i≤maxCount; i++){
6:             flexiblePointTable=FlexiblePosition(Table);   //利用算法 2 得到标出所有灵活位置的表
7:             If (flexiblePointTable 中不含“*”)
8:                 return CA;
9:             得到 flexiblePointTable 中含“*”最多的一行,设行标为 row;
10:            在 Table 第 row 行的灵活位置上利用蚁群算法进行演化;
11:            If (isCoveringArray(Table)){
12:                CA=Table;
13:                break;   //若演化得到覆盖表,则进入到下一次 while 循环中
14:            }
15:            If (i==maxCount) return CA;
16:        }
17:    }
```

18: **End**

4.3 实验及讨论

为了验证本节给出的利用蚁群算法基于解结构调整的覆盖表生成算法的有效性,本节给出了相关实验数据.

本文设计的整表演化算法是一种后优化算法.算法引入了随机后优化算法中“灵活位置”的概念,下面我们给出基于蚁群算法的整表演化算法和先用 AETG-MMAS 算法然后再利用随机后优化算法的性能比较,见表 20.其中,AETG-MMASp 表示对 AETG-MMAS 算法生成覆盖表使用随机后优化算法的方法.CA-AS 算法表示基于 AS 算法的整表生成算法.实验选取了 10 个覆盖表作为实验对象.从表中我们可以看出,除了第 10 个表 CA-AS 表现略差于 AETG-MMASp 外,CA-AS 算法所得结果的尺寸以及稳定性都要好于 AETG-MMASp.算法采用的是第 4 节给出的 AS 算法推荐参数配置,实验中覆盖表 CA9 与 CA10 的最大迭代次数为 10 000,而其余表的最大迭代次数为 1 000.

Table 20 Comparison of AETG-MMAS,AETG-MMASp and CA-AS

表 20 AETG-MMAS,AETG-MMASp 与 CA-AS 的比较

CA1(N;2;3^4)				MCA2 (N;2;$5^1 3^8 2^2$)			
	AETG-MMAS	AETG-MMASp	CA-AS		AETG-MMAS	AETG-MMASp	CA-AS
Min	9	9	9	Min	21	19	17
Max	13	12	9	Max	25	22	19
Avg	10.366 67	9.466 667	9	Avg	23.4	20.366 67	18
Dev	0.808 717	0.973 204	0	Dev	0.894 427	0.808 717	0.454 859
CA3(N;2;3^{13})				MCA4(N;2;$4^1 3^{39} 2^{35}$)			
	AETG-MMAS	AETG-MMASp	CA-AS		AETG-MMAS	AETG-MMASp	CA-AS
Min	21	16	16	Min	30	24	24
Max	23	19	17	Max	33	28	25
Avg	21.566 67	17.4	16.733 33	Avg	31.466 67	25.833 33	24.733 33
Dev	0.626 062	0.813 676	0.449 776	Dev	0.776 079	1.1768 85	0.449 776
MCA5(N;2;$4^{15} 3^{17} 2^{29}$)				MCA6(N;2;$6^{15} 4^6 3^8 2^3$)			
	AETG-MMAS	AETG-MMASp	CA-AS		AETG-MMAS	AETG-MMASp	CA-AS
Min	42	35	34	Min	38	34	33
Max	47	41	37	Max	44	41	36
Avg	43.866 67	38.466 67	35.333 33	Avg	40.4	37.333 33	34.266 67
Dev	1.279 368	1.479 36	0.606 478	Dev	1.631 585	1.397 864	0.739 68
MCA7(N;2;$7^1 6^1 5^1 4^5 3^8 2^3$)				CA8(N;2;4^{10})			
	AETG-MMAS	AETG-MMASp	CA-AS		AETG-MMAS	AETG-MMASp	CA-AS
Min	48	45	42	Min	31	27	26
Max	54	53	47	Max	34	30	27
Avg	50.533 33	49.133 33	44.5	Avg	32.366 67	28.166 67	26.966 67
Dev	1.795 268	1.775 957	1.332 615	Dev	0.718 395	0.592 093	0.182 574
CA9(N;2;4^{20})				CA10(N;2;6^{10})			
	AETG-MMAS	AETG-MMASp	CA-AS		AETG-MMAS	AETG-MMASp	CA-AS
Min	41	33	33	Min	60	58	60
Max	43	37	34	Max	65	62	62
Avg	42.166 67	34.4	33.033 33	Avg	62.3	59.633 33	60.933 33
Dev	0.746 64	0.968 468	0.182 574	Dev	1.055 364	1.066 2	0.520 83

表 21 给出了 CA-AS 和其他算法生成覆盖表最小尺寸的对比.其中,MMAS 表示 AETG-MMAS 算法,IPOs 是 IPO 算法的一种改进,SA 表示模拟退火算法[19].实验对象为表 20 中的 10 个表.根据结果可以发现,CA-AS 比 AETG-MMAS 性能有很大的提升,只在 MCA4 和 MCA5 两种输入时生成的尺寸比 IPOs 要大[20],但是当前最优结果都是由 SA 生成的,因此 CA-AS 算法仍然具有一定的改进空间.

Table 21 Comparison of CA-AS and other algorithms

表 21 CA-AS 和其他算法的比较

CA	CA1	MCA2	CA3	MCA4	MCA5	MCA6	MCA7	CA8	CA9	CA10	4^{30}	4^{40}	4^{50}
CA-AS	9	17	16	24	34	33	42	26	33	60	37	40	42
MMAS	9	21	21	30	42	38	48	31	41	60	–	–	–
IPOs	9	17	–	23	32	–	–	28	34	–	38	41	43
SA	9	15	15	21	30	30	42	–	–	–	–	–	–
Best	9	15	15	21	30	30	42	–	–	–	–	–	–

蚁群算法本身非常耗时.在用蚁群算法进行整表演化实验时,这个问题尤为严重.我们对 CA(N;2;4^k)(k=10,20,30,40,50)这 5 个输入进行了实验.实验结果如图 8 所示.从图 8 中我们可以看到,随着 k 的增长,CA-AS 算法的时间(s)会显著增加.当 k=50 时,算法一次运行的时间接近 2 小时.针对时间开销问题,本文余下部分对算法进行了并行化探索.

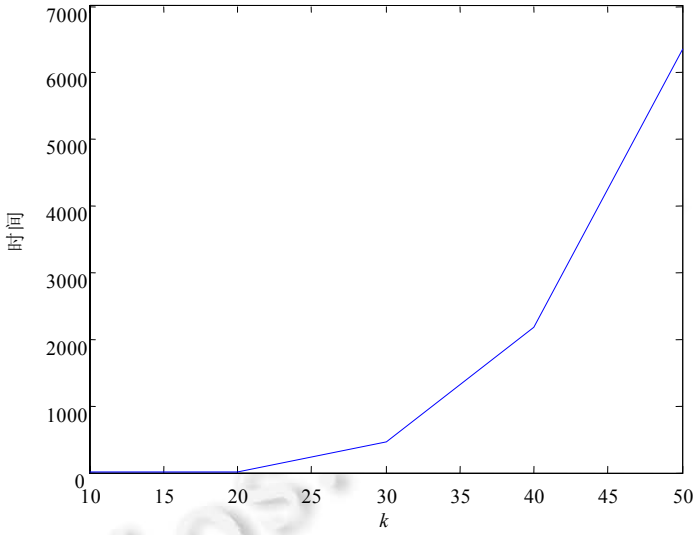


Fig.8 Time cost of CA-AS

图 8 CA-AS 算法的时间开销

5 利用蚁群算法生成覆盖表的并行化探索

5.1 并行化探索

由上述讨论我们可以看到,用蚁群算法生成覆盖表是一个非常耗时的工作.即便仅使用计算速度较快的 AETG-ACO,当覆盖表规模达到一定程度时,其计算时间可能仍然是难以承受的.所以,我们尝试从两方面来考虑将利用蚁群算法生成覆盖表的方法并行化.其一是将覆盖表本身的一些步骤并行化,其二是将蚁群算法中的某些步骤并行化.

在整表生成的过程中,需要计算灵活位置.当表的行数较多时,计算灵活位置会花费较大的计算代价.因此,我们提出并行化地寻找灵活位置的方法:将算法 2 的第 4 行~第 7 行以及计算所有 t 维组合被覆盖次数的过程并行化处理.

蚁群算法本身是一种群智算法,演化学习过程主要是依靠每一代蚂蚁对环境的改变.在 AS 算法和 MMAS 算法中,每一代蚂蚁中的个体的行为都具有独立性,因此可以考虑将这两个变种的蚁群算法的蚂蚁迭代过程改为并行过程,从而节约计算时间.但是需要注意的是,每一代蚂蚁的迭代过程都依赖之前所有代蚂蚁的行为,约束性太强,所以不同代的蚂蚁不能并行化.另外,由于在 ACS 算法中,每次一只蚂蚁经过一条路径都会更新信息素,所以不方便将蚂蚁的演化过程并行化.对于 AS 和 MMAS,算法 1 中第 9 行~第 13 行也可以并行处理.

5.2 实验及讨论

上面我们分析了利用蚁群算法生成覆盖表的过程中的一些可并行的成分,下面我们以并行化的蚂蚁为例设计实验.选取如表 22 所示的 6 个二维覆盖表来进行实验.对于每一个表,分别设置并行的线程数为 1~8,每一种线程设置都重复进行 10 次实验.

Table 22 Experiment objects of parallel experiments

表 22 并行化实验对象

CA(N;2;3^{13})	CA(N;2;4^{10})	CA(N;2;6^{10})
MCA(N;2;5^1,3^8,2^2)	MCA(N;2;6^1;5^1;4^6;3^8;2^3)	MCA(N;2;7^1;6^1;5^1;4^5;3^8;2^3)

我们首先在 hadoop 集群上进行算法并行化的实验,但是 hadoop 本身不适合完成计算密集型的任务[21].在 hadoop 集群上得到的实验结果还不如单机的实验结果,于是我们放弃了实验 hadoop 进行并行化实验的想法.由于现在的计算机往往都是多核的,CPU 使用率很低,于是我们采取另外一种并行化策略,即利用 Java 的多线程机制[22].我们使用了 Java 版本的 OpenMP,将蚂蚁寻找解的过程并行化.实验环境为 Java1.7,Windows8.1 64 位系统,Intel(R) Core(TM)i7-3840QM CPU(4 核 8 处理器 2.8GHz)内存 16GB.算法的参数我们固定为如表 23 所示.

Table 23 Configuration of parallel AETG-AS

表 23 并行化 AETG-AS 算法的配置参数

Nc	m	ρ	τ_{init}	Q	α	β
200	100	0.6	0.4	0.01	0.7	0.1

最后我们得到的实验结果见表 24,将 3 个水平覆盖表和 3 个混合覆盖表从左到右分别记为 CA1~CA3, MCA1~MCA3.表中记录的数据是不同的表在不同线程数下分别运行 10 次所需要的平均时间.

Table 24 Run time of parallel AETG-AS when thread number varies

表 24 并行化 AETG-AS 算法不同线程数的运行时间

线程数	1	2	3	4	5	6	7	8
CA1	8.86	4.87	4.17	3.47	3.16	3.15	3.10	3.53
CA2	6.29	3.71	3.12	2.50	2.24	2.15	2.17	2.37
CA3	14.08	8.21	6.77	5.55	5.02	4.80	4.74	5.36
MCA1	6.06	3.54	2.96	2.44	2.21	2.11	2.05	2.35
MCA2	58.98	32.68	27.94	24.81	23.93	24.86	23.94	27.45
MCA3	79.20	43.77	35.98	30.69	30.78	31.15	31.00	33.84

为了表现得更加直观,我们将实验结果以折线图的形式表现出来,如图 9 所示.

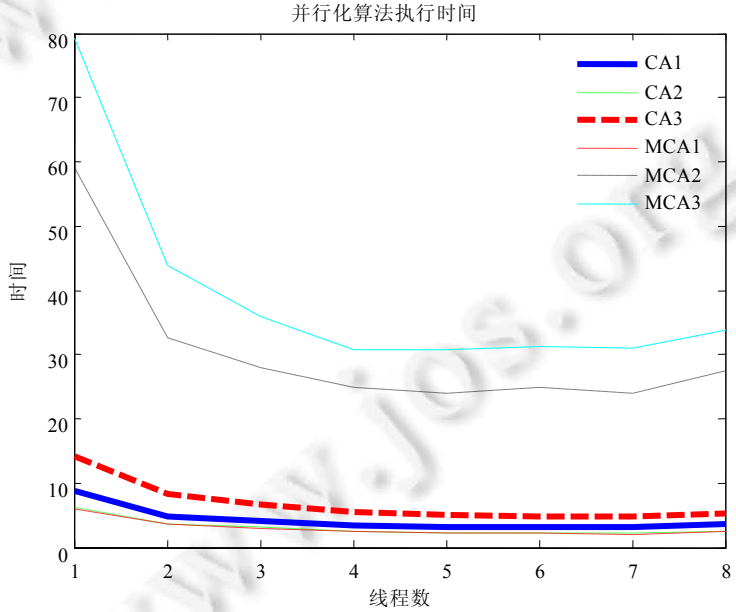


Fig.9 Result of parallel experiments

图 9 并行实验的结果

根据结果我们可以看出,并行化后算法的运行速度更快,但是运行时间不是随着线程数的增加而一直变短.

当线程个数接近处理器的个数时,算法运行的时间反而变长.在实际的运行过程中,我们要根据自己所拥有的计算资源合理设置并行的尺度.

6 总 结

本文主要对利用蚁群算法生成覆盖表的性能进行深入的探索与挖掘.蚁群算法作为一种常见的演化搜索算法,成功地解决了一系列组合优化问题.而在求解覆盖表生成问题的过程中,虽然已有的研究工作说明了蚁群算法的可用性,但却缺乏对其实际性能的深入探讨.本文在已有工作的基础上,结合对蚁群算法、覆盖表生成算法的相关研究成果,研究了利用蚁群算法生成覆盖表的算法变种、算法参数、解结构调整以及并行化,充分挖掘了蚁群算法生成覆盖表的潜力.主要包括以下几个方面的工作:

1) 对已有的蚁群算法一次一条测试用例生成覆盖表的算法进行了深入挖掘,研究了算法的变种以及参数的配置.研究算法变种时,我们提出利用蚁群算法生成覆盖表的框架,选择了 3 种算法变种 AS,ACS,MMAS 来进行覆盖表的生成,并得到相应的 AETG-AS,AETG-ACS,AETG-MMAS 算法;
2) 进行了单参数配置的研究,然后研究了二维参数覆盖表,最后为每种算法变种提供了一种推荐配置.在推荐配置下,将 3 种覆盖表生成算法的结果进行比较,发现效果最好的是 AETG-MMAS 算法;
3) 对蚁群算法生成覆盖表的演化目标进行了调整,从已有的演化一条测试用例调整到演化一个整表,提出了在灵活位置上进行演化的 CA-AS 算法.该算法能够克服一次一条测试用例生成方法的局限性,得到更好的结果.实验结果表明,CA-AS 的实验效果要比 AETG-MMAS 的实验效果好得多,非常接近已知最优解;
4) 将蚁群算法生成覆盖表的算法中的一些过程(包括灵活位置求解,每一代蚂蚁的演化)进行并行化处理,解决了在表的规模较大时非常耗时的问题.在 hadoop 平台上的实验失败后,使用 Java 版本的 OpenMP,将蚂蚁寻找解的过程并行化,最后有效地缩短了算法的运行时间.

虽然经过改进的结果未能超越最优解,但是这项工作仍然是有意义的.从蚁群算法的角度来看,该算法作为一种元启发式算法,能够经过小的改动而应用在不同的优化问题中.但是,不同的问题具有不同的特点,所以对特定的问题,对蚁群算法的设计在其性能方面会产生很大的影响.而本文研究了蚁群算法在覆盖表生成问题上的应用,针对这个问题对蚁群算法进行设计,使得在这个问题上的潜力充分发挥出来.从覆盖表生成的角度来看,生成覆盖表有很多种方法,本文对蚁群算法进行了评估,展示了利用蚁群算法来生成覆盖表的优劣.

尽管以上研究探索很好地挖掘了蚁群算法生成覆盖表的潜力,但仍然存在很多不足.例如对算法变种选择而言,蚁群算法的变种有很多,我们却只选择了其中的 3 种算法进行研究.在参数调优的过程中,一方面,参数的离散程度不够,覆盖强度也不够;另一方面,我们选择的实验对象比较少,实验次数也不充足.整表演化时,我们采取逐渐减小覆盖表尺寸的方法,没有对逐渐增加测试用例条数的方法做过多的讨论.此外,表尺寸减小的方式、演化位置的选择也存在多种策略,我们没有进行过多的讨论.在缩短算法时间方面,我们只是让能够并行执行的步骤并行化执行,没有更加深入地设计更有效的分布式算法,也没有成功地利用分布式平台.针对上述不足,将来会展开,作进一步的研究,让利用蚁群算法生成覆盖表的方法能够发挥出更大的潜力.

References:

[1] Nie CH. Concepts and Methods of Software Testing. Beijing: Tsinghua University Press, 2013. 1−22 (in Chinese).
[2] Nie CH. Combinatorial Testing. Beijing: Beijing Science Press, 2015. 1−119 (in Chinese).
[3] McCaffrey JD. Generation of pairwise test sets using a genetic algorithm. In: Proc. of the Int'l Conf. on Computer Software and Applications (COMPSAC). 2009. 626−631. [doi: 10.1109/COMPSAC.2009.91]
[4] Nie CH, Leung H. A survey of combinatorial testing. ACM Computing Surveys (CSUR), 2011,43(2):11. [doi: 10.1145/1883612.1883618]
[5] Yu L, Tai KC. In-Parameter-Order: A test generation strategy for pairwise testing. In: Proc. of the High-Assurance Systems Engineering Symp. 1998. 254−261. [doi: 10.1109/HASE.1998.731623]

[6] Bryce RC, Colbourn CJ. The density algorithm for pairwise interaction testing. SoftwareTesting, Verification and Reliability, 2007,17(3):159−182. [doi: 10.1002/stvr.365]

[7] Dorigo M, Maniezzo V, Colorni A. Ant system: Optimization by a colony of cooperating agents. IEEE Trans. on Systems, Man, and Cybernetics (Part B Cybernetics), 1996,26(1):29−41. [doi: 10.1109/3477.484436]

[8] Dorigo M, Stützle T. Ant Colony Optimization. Cambridge: MIT Press, 2004. 1−151.

[9] Cohen DM, Dalal SR, Fredman ML, Patton GC. The AETG system: An approach to testing based on combinatorial design. IEEE Trans. on Software Engineering (TSE), 1997,23(7):437−444. [doi: 10.1109/32.605761]

[10] Shiba T, Tsuchiya T, Kikuno T. Using artificial life techniques to generate test cases for combinatorial testing. In: Proc. of the Computer Software and Applications Conf. 2004. 72−77. [doi: 10.1109/CMPSAC.2004.1342808]

[11] Chen X, Gu Q, Li A, Chen DX. Variable strength interaction testing with an ant colony system approach. In: Proc. of the Asia-Pacific Software Engineering Conf. (ASPEC). 2009. 160−167. [doi: 10.1109/APSEC.2009.18]

[12] Chen X, Gu X, Zhang X, Chen DX. Building prioritized pairwise interaction test suites with ant colony optimization. In: Proc. of the Int'l Conf. on Quality Software (QSIC). 2009. 347−352. [doi: 10.1109/QSIC.2009.52]

[13] Nie CH, Wu HY, Liang YL, Leung H, Kuo FC, Li Z. Search based combinatorial testing. In: Proc. of the Asia-Pacific Software Engineering Conf. (ASPEC). 2012. 778−783. [doi: 10.1109/APSEC.2012.16]

[14] Dorigo M, Gambardella LM. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Trans. on Evolutionary Computation, 1997,1(1):1−24. [doi: 10.1109/TEVC.1997.585887]

[15] Dorigo M, Blum C. Ant colony optimization theory: A survey. Theoretical Computer Science, 2005,344(2-3):243−278. [doi: 10.1016/j.tcs.2005.05.020]

[16] Liang YL, Nie CH. The optimization of configurable genetic algorithm for covering arrays generation. Chinese Journal of Computers, 2012,35(7):1522−1538 (in Chinese with English abstract).

[17] Bryce RC, Colbourn CJ. One-Test-at-a-Time heuristic search for interaction test suites. In: Proc. of the 9th Annual Conf. on Genetic and Evolutionary Computation (GECCO). 2007. 1082−1089. [doi: 10.1145/1276958.1277173]

[18] Nayeri P, Colbourn CJ, Konjevod G. Randomized post-optimization of covering arrays. European Journal of Combinatorics, 2013,34(1):91−103. [doi: 10.1016/j.ejc.2012.07.017]

[19] Torres-Jimenez J, Rodriguez-Tello E. New bounds for binary covering arrays using simulated annealing. Information Sciences, 2012,185(1):137−152. [doi: 10.1016/j.ins.2011.09.020]

[20] Calvagna A, Gargantini A. IPO-s: Incremental generation of combinatorial interaction test data based on symmetries of covering arrays. In: Proc. of the IEEE Int'l Conf. on Software Testing Verification and Validation Workshops. 2009. [doi: 10.1109/ICSTW.2009.7]

[21] White T. Hadoop: The Definitive Guide. 4th ed., Sebastopol: O'Reilly Media, 2015. 3−97.

[22] Eckel B. Thinking in Java. 4th ed., Upper Saddle River: Prentice Hall, 2006. 797−822.

附中文参考文献:

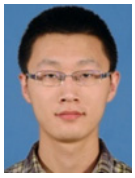
[1] 聂长海.软件测试的概念与方法.北京:清华大学出版社,2013.1−22.

[2] 聂长海.组合测试.北京:科学出版社,2015.1−119.

[16] 梁亚澜,聂长海.覆盖表生成的遗传算法配置参数优化.计算机学报,2012,35(7):1522−1538.

曾梦凡(1992－),男,湖北仙桃人,学士,主要研究领域为软件测试.

张文茜(1994－),女,学士,主要研究领域为软件测试.



陈思洋(1989－),男,硕士,CCF 学生会员,主要研究领域为软件测试.

聂长海(1971－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试.