软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(7):1469−1483 [doi: 10.3724/SP.J.1001.2013.04326]

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

覆盖表生成的可配置贪心算法优化*

聂长海[1], 蒋 静[2]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

[2](中国电力科学研究院,江苏 南京 211106)

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

摘 要: 覆盖表生成是组合测试研究的关键问题之一,其中,贪心算法因为速度快、生成的覆盖表规模小而得到人们的青睐.人们提出了很多基于不同策略的贪心算法,其中,多数算法可以归结到一个统一的算法框架,即形成一个可配置贪心算法,从该框架又可以衍生出很多新的算法.如何科学地配置优化受多个因素影响的算法框架、有效生成覆盖表是一个新的挑战.针对具有 6 个决策点的贪心算法框架,设计了 3 条不同的实验路线,系统地探索各个决策点以及它们之间相互作用对生成覆盖表规模的不同影响,寻找最佳配置,从而可以有效地生成规模更小的覆盖表,为覆盖表生成的贪心算法的设计和优化提供理论和实践基础.

关键词: 组合测试;贪心算法;覆盖表;软件测试;测试用例生成

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 聂长海,蒋静.覆盖表生成的可配置贪心算法优化.软件学报,2013,24(7):1469−1483. http://www.jos.org.cn/1000-9825/4326.htm

英文引用格式: Nie CH, Jiang J. Optimization of configurable greedy algorithm for covering arrays generation. Ruan Jian Xue Bao/Journal of Software, 2013,24(7):1469−1483 (in Chinese). http://www.jos.org.cn/1000-9825/4326.htm

Optimization of Configurable Greedy Algorithm for Covering Arrays Generation

NIE Chang-Hai[1], JIANG Jing[2]

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

[2](China Electric Power Research Institute, Nanjing 211106, China)

Corresponding author: NIE Chang-Hai, E-mail: changhainie@nju.edu.cn, http://www.nju.edu.cn

Abstract: Covering an array generation is one of the key issues in combinatorial testing, and algorithms are popular due to its ability to deliver smaller covering array in shorter time. People have proposed many greedy algorithms based on different strategies, and most of these can be integrated into a framework, which forms a configurable greedy algorithm. Many new algorithms can be developed within this framework, however, deploying and optimizing the framework affected by multiple factors to construct more efficient covering arrays is a new challenge. The paper designs three different experiments under the framework with six decisions, systematically explore the influence of each of the decisions and interactions among them, to find the best configuration for generating smaller covering array, and provide theoretical and practical guideline for the design and optimization of greedy algorithms.

Key words: combinatorial testing; greedy algorithm; covering array; software testing; test case generation

软件系统的正常运行受到很多因素的影响,这些因素间未知的相互作用可能会造成一些难以预料的系统故障,需要设计一个科学而有效的测试计划系统地检测这些因素间的交互关系.组合测试是一种有效的软件测试方法,它生成少量高质量的测试数据[1],对参数之间的组合进行系统的检测.Kuhn 等人对一些软件系统进行研

* 基金项目: 国家自然科学基金(61272079, 61021062); 国家高技术研究发展计划(863)(2008AA01Z143); 江苏省自然科学基金(BK2010372)

收稿时间: 2012-06-30; 修改时间: 2012-08-20; 定稿时间: 2012-09-29

究发现,测试检测任意两个参数之间的交互关系能够找出 70%的错误[2].基于测试的成本、测试用例的生成难度、各种组合覆盖程度对测试用例集(覆盖表)规模的影响等因素,覆盖任意两个参数之间组合的二维组合测试(使用二维覆盖表作为测试用例集的测试方法)是人们最常用的方法.

覆盖表生成是组合测试研究中的一个热点问题,在关于组合测试研究的论文中,有 50%以上的论文是研究覆盖表生成问题的[1,3−5].人们已经提出很多数学方法、启发式搜索方法和各种贪心算法,但这些方法都存在各自的局限性,并只能在解决某些特定问题时具有一定的优势.例如:TConfig[6]是一种利用正交表等基本成分进行递归构造的数学方法,该方法具有生成速度快的特点,但不足之处就是需要依赖已有的代数或组合对象;在启发式搜索方面,可以使用禁忌搜索[7]和模拟退火(SA)[8]等方法生成较小的测试用例集,但这些方法一般需要较长的时间.相比之下,启发式贪心算法不仅灵活,而且速度快,目前已经分别提出了 AETG[9−11],TCG[12],DDA[13,14]和 IPO[15]等多种启发式贪心方法,这些方法之间既有区别又有很大的相似性.

Bryce 等人[16]根据 AETG,TCG 和 DDA 方法构建了一个贪心算法框架,该框架不仅包括已有这 3 种方法,而且从该框架又可以衍生出很多新的贪心算法.因此,如何科学地配置、优化受多个因素影响的该算法框架,有效生成覆盖表是一个新的挑战.Bryce 等人利用统计工具 ANOVA 来分析各层框架对覆盖表规模的影响,给出了一些定性的结果,但是他们没有给出具体可用的框架配置.本文针对该算法框架的 6 个决策点设计了 3 条不同的实验路线,系统地探索各个决策点以及它们之间相互作用对生成覆盖表规模的不同影响.针对具体问题,可以寻找最佳配置,生成规模更小的覆盖表,为覆盖表生成的贪心算法的设计和优化提供理论和实践基础.

本文第 1 节通过一个例子介绍组合测试基本模型和覆盖表的定义.第 2 节简要介绍贪心算法框架.第 3 节描述具体的实验设计.第 4 节分析实验数据,并给出实验结果.最后是总结和展望.

1 基本模型和相关定义

假定影响待测系统软件 SUT(software under testing)的参数有 k 个,记为 $F=\{f_1,f_2,\ldots,f_k\}$,其中,参数 f_i 有 a_i 个可能取值,每个参数取值为 $V_i=\{0,1,\ldots,a_i-1\},1\leqslant i\leqslant k$.

定义 1. 记 k 元组$(v_1,v_2,\ldots,v_k)(v_i\in V_i)$为 SUT 的一条测试用例.

覆盖表是组合测试的测试用例集,其定义如下:

定义 2. 待测试系统 SUT 的 t 维覆盖表 $CA(N;t,k,(v_1,v_2,\ldots,v_k))$(或 $CA(N;t,v^k)$,当 $v_1=v_2=\ldots=v_k=v$ 时)是一个 $N\times k$ 的数组,其中第 i 列对应第 i 个参数,该参数取值记为 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[16],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,即利用二维覆盖表 $CA(N;2,k,(v_1,v_2,\ldots,v_k))$进行测试.

为了便于清楚地说明问题,我们以一个网络软件系统为例,它有多个浏览器(f_1)、操作系统(f_2)、网络连接方式(f_3)以及内存配置(f_4),见表 1,其中 4 个参数,每个参数都有 3 个取值.表 2 就是相应的二维覆盖表 $CA(9;2,3^4)$.可以看出,表 1 中任意两个参数之间的 9 个组合均出现在表 2 中相应的两列中.

Table 1 Configuration of the netware system

表 1 网络软件系统的配置

Web browser	Operating system	Connection type	Memory
Netscape	Windows	LAN	256MB
IE	Macintosh	PPP	512MB
Mozilla	Linux	ISDN	1GB

Table 2 Test suite for the netware system

表 2 网络软件系统的测试用例集

Test No.	Web browser	Operating system	Connection type	Memory
1	Netscape	Windows	LAN	256MB
2	IE	Macintosh	LAN	512MB
3	Netscape	Macintosh	PPP	1GB
4	Mozilla	Linux	LAN	1GB
5	Netscape	Linux	ISDN	512MB

Table 2 Test suite for the netware system (Continued)

表 2 网络软件系统的测试用例集(续)

Test No.	Web browser	Operating system	Connection type	Memory
6	IE	Linux	PPP	256MB
7	Mozilla	Windows	PPP	512MB
8	IE	Windows	ISDN	1GB
9	Mozilla	Macintosh	ISDN	256MB

2 覆盖表生成的贪心算法框架

Bryce 等人[16]从已有的贪心算法(AETG,TCG 和 DDA)中提炼出一个 4 层的具有 6 个决策点的算法框架(如图 1 所示).框架的基本思想是:每次只生成一条测试用例,使得该测试用例尽可能多地覆盖已有测试用例集 C 中未出现的参数组合.该方法每次确定一个参数,然后为该参数赋值,依次执行直到所有参数值都被确定,一条测试用例就生成了,具体过程如图 1 所示.该过程涉及表 3 所列的 6 个主要决策点(图 1 中的阴影部分 2.1~2.6 所示).下面我们分别介绍图 1 所示框架中的各个决策点.

待测试系统具有 k 个参数,每个参数 f_i 有 a_i 个可能取值,已有测试用例集 C(可事先指定部分测试用例,本文设为空集)

1. 按照**参数排序策略**选择第 i 个参数 //2.3:参数排序策略
2. 若有多个参数可选,采取**同序参数选择**策略选择参数 f_i //2.5:同序参数选择策略
3. 给选定的参数 f_i 按照**值选择策略**赋值 //2.4:值选择策略
4. 若参数 f_i 有多个值可选,采取**同序值选择**策略选择一个值 l_i //2.6:同序值选择策略
5. 反复执行步骤 1~步骤 4,直到所有参数都被取值,生成测试用例 T //第 4 层
6. 对步骤 1~步骤 5 循环执行 **Candidates** 次,生成多个候选测试用例 //2.2:候选用例集数据
7. 选择覆盖未被覆盖对最多的测试用例作为最终的测试用例 Bst$T$,加入测试用例集 C 中 //第 3 层
8. 反复执行步骤 1~步骤 7,直到参数之间的两两组合都被覆盖为止,覆盖表就生成了 //第 2 层
9. 循环执行步骤 1~步骤 8**Repetitions** 次,选择规模最小的作为最终覆盖表 //2.1:循环次数 //第 1 层

Fig.1 Framework of the greedy algorithms

图 1 贪心算法框架

Table 3 Values of the six decision points in the greedy algorithm framework

表 3 贪心算法框架中 6 个决策点的具体决策

Value No.	Repetition count	Candidate count	Factor ordering	Level selection	Factor tie-breaking	Level tie-breaking
0	1	1	Random	Random	Random	Random
1	5	5	Uncovered pairs	Uncovered pairs	Uncovered pairs	Uncovered pairs
2	10	10	Density	Density	Take first	Take first
3	20	20	Level			Least frequent
4			Hybrid			

2.1 算法的循环次数(repetitions)

由于某些决策点的策略是随机的,算法的每次运行一般结果都不相同,多次运行会有更多机会获得较小的覆盖表,但运行次数也不能过多.如果算法循环次数过多,不仅会导致时间成本增加,而且我们在实践中发现,当次数增加到一定程度时,覆盖表规模不再减小.所以在本文中,循环次数只考虑 1,5,10 和 20 这 4 种情况.

2.2 候选用例集的数目(candidates)

由于某些决策点策略的随机性,贪心算法框架每一轮都可以生成多个候选测试用例,选择覆盖未被覆盖对最多的一条用例加入到测试用例集 C 中.这里,我们将候选用例集数目定为 1,5,10 和 20 这 4 种取值.

2.3 参数排序(factor ordering selection)

这是框架的核心之一,目前从已有算法中提炼出 5 种策略,分别是:

(1) 与已确定参数相关,即选择和已确定参数覆盖最多未被覆盖对的参数(uncovered pairs)[9−11];

(2) 与已确定参数及待定参数的期望相关(density)[13,14];

(3) 与参数的取值数目相关(level),按参数的取值数目降序排列[12];

(4) 随机排序(random);

(5) 杂交策略(hybrid).

第 1 个参数的选择和后续参数策略不同,第 1 个参数是选择的未被覆盖对最多的参数,其余参数是随机排序[9−11].

2.4 值选择(level selection)

值选择的目标是覆盖最多的未被覆盖对[1],Bryce 等人提出以下 3 种确定参数取值的标准:

(1) 随机选择;

(2) 与已确定参数相关[9−11],即所选择的值和已确定参数之间存在最多的未被覆盖对;

(3) 与所有参数未被覆盖对的期望值相关,即取决于该参数值的密度(density)[13,14].

2.5 同序参数选择(factor tie-breaking)

当采用第 2.3 节中 5 种方法选择参数时,可能会出现多个参数具有相同的排序而无法确定.

为了处理这种情况,设计出了打破平局的策略.按字典序(take first)、随机选择以及选择未被覆盖对最多的参数是本文采用的 3 种方法.

2.6 同序值选择(level tie-breaking)

按照第 2.4 节中值选择的方法,也可能出现多个取值具有相同排序而无法为参数赋值的情况.这里采用 4 种方法:随机选择、按字典序、选择未被覆盖对最多的值以及最少使用过的值(least frequently).

2.7 贪心算法框架与已有贪心算法的关系

当框架中的策略组合为(–,50,hybrid,uncovered pairs, –,uncovered pairs)时,是已有的 AETG[9−11]算法,其中,“–”表示该策略可以随意指定.在 AETG 中,循环次数、同序参数选择策略均可任意指定,它的参数排序采用了杂交策略,第 1 个参数选择了未被覆盖对最多的参数,其余参数随机排序.

当框架中策略组合为(1,1,density,density,take first,take first)时生成的是 DDA[13,14]算法,它的参数排序也考虑了未赋值参数,DDA 给所有参数都定义了一个密度值,它是一种确定型的算法.

TCG[12]与 AETG 非常接近,差异在于它的参数排序是按照参数取值数目的降序排列的,候选测试用例的数目也是确定的,即排序后第 1 个参数的取值数目为 v_{max}.每条候选测试用例的第 1 个参数也依次赋值为 1,2,…,v_{max}.TCG 中,值选择也是一个杂交策略,第 1 个参数的取值异于其余参数,这里考虑的是所有参数按照同一策略赋值.在框架中,如果让候选用例集规模(candidate)与参数的最大取值个数(v_{max})保持一致,就形成了 TCG 算法[12].

3 实验设计

覆盖表的生成是一个 NP-Hard 问题,所有方法一般都只能求近似求解.AETG,TCG 和 DDA 等方法在解决具体问题时各有千秋,但是没有一种方法能够在精确性、有效性、一致性以及可扩展性等方面均具有优势[1]. Bryce 等人从这些算法中提炼出具有 6 个决策点的贪心算法框架[16],框架下的每个决策点均存在多种策略,任意组合这些决策点可以生成一大类(本例中,4×4×5×3×3×4=2880 种)贪心算法,这其中也包括 AETG 和 DDA 等已有算法.本文研究该框架下的各种贪心算法,关注这类算法在生成覆盖表规模上的性能,主要解决以下问题:

(1) 算法框架下,各个决策点的配置是否对其生成的覆盖表规模具有明显的影响?

(2) 如果影响明显,是否存在最优配置,对于某些实例使用该配置总能生成规模较小的覆盖表?

(3) 最优配置是否具有一定的普遍性,即利用该配置对其他实例能够生成较小规模的覆盖表?

(4) 最优配置下的贪心算法与已有的 AETG,DDA,TCG 方法相比是否具有竞争力?

为了回答这些问题,我们设计实现了一个可配置贪心算法下的覆盖表自动生成工具作为实验平台,它能够运行任意组合配置的贪心算法.该工具的输入有两个:(1) 待测系统的参数个数及相应取值,如 4^{40}(表示一个系统具有 40 个参数,每个参数有 4 个取值);(2) 配置集,框架中 6 个决策点的取值,形如(5,10,uncovered pairs,uncovered pairs,random,random)这样的配置组成的集合.为了记录方便,我们对表 3 中的各个策略用自然数表示,假定某个决策点有 n 种策略,我们记为{0,1,…,n–1},分别对应表 3 中最左边第 1 列的值编号(value No.).例如,配置(5,10,uncovered pairs,uncovered pairs,random,random)即可表示成(1,2,1,1,0,0).该工具的输出有两个:(1) 对于具体待测系统,能够生成最小规模覆盖表的最优配置及相应的覆盖表;(2) 每个配置下生成的覆盖表规模.

实验分为最佳配置的探索和验证两个阶段:首先是从不同角度生成算法框架的配置集,通过实验平台对每个配置集进行系统的实验,探索每个决策点以及相互之间关系对框架的不同影响(问题 1),找出最佳配置(问题 2);第 2 阶段则是对得出的最佳配置进行验证,检测它是否适用于其他实例(问题 3),并与 AETG,TCG 和 DDA 等已有算法进行比较,查看在生成覆盖表规模上是否具有竞争力(问题 4).在第 1 阶段,我们从 3 个角度来设计配置集,形成 3 条不同的实验路线.

(1) Pairwise 配置集

针对表 3 中的多个决策点及策略,设计生成一个二维覆盖的配置集.它能够保证任意两个决策点的所有组合均被覆盖,共 23 条配置(见表 4).表 4 第 1 行 f_0,f_1,…,f_5 表示框架中的 6 个决策点,第 1 列是配置的编号.这个配置集覆盖了所有决策点的二维组合,配置集包含了常见的贪心算法.例如,第 3 条配置对应的是 DDA 算法,第 10 条配置对应的是 AETG 算法.

Table 4 Configurations for pairwise experiment

表 4 Pairwise 实验的配置集

Test No.	f_0	f_1	f_2	f_3	f_4	f_5
P1	2	2	0	0	0	0
P2	0	0	1	1	1	1
P3	1	1	2	2	2	2
P4	3	3	3	1	0	3
P5	1	0	4	0	1	0
P6	0	1	0	0	2	3
P7	3	2	4	2	2	1
P8	2	3	1	2	1	2
P9	2	0	2	2	0	3
P10	2	1	4	1	0	1
P11	0	2	3	0	1	2
P12	0	3	2	1	2	0
P13	3	0	0	1	1	2
P14	3	1	1	0	0	0
P15	1	3	0	0	0	1
P16	1	2	1	1	2	3
P17	1	0	3	2	2	0
P18	0	3	4	2	0	2
P19	3	2	2	0	1	1
P20	2	1	3	1	1	1
P21	0	0	0	2	1	3
P22	1	1	4	0	2	3
P23	2	2	1	1	2	0

(2) Base Choice 配置集

Base Choice[4]是一种组合设计方法,它能够满足一维覆盖.该方法首先确定一个基准配置,后续配置则是一次只变动基准配置中一个决策点的取值来完成,其余决策点保持不变,这样直到所有决策点的取值都被覆盖,配置集就生成了.表 5 是以表 4 中配置 P23=(2,2,1,1,2,0)作为基准配置对应的一个 Base Choice 配置集.

Table 5 Configurations for Base Choice experiment (Basic configuration B1=P23)

表 5 Base Choice 配置集(基准配置 B1=P23)

Test No.	f_0	f_1	f_2	f_3	f_4	f_5	Test No.	f_0	f_1	f_2	f_3	f_4	f_5
B1	2	2	1	1	2	0	B10	2	2	3	1	2	0
B2	0	2	1	1	2	0	B11	2	2	4	1	2	0
B3	1	2	1	1	2	0	B12	2	2	1	0	2	0
B4	3	2	1	1	2	0	B13	2	2	1	2	2	0
B5	2	0	1	1	2	0	B14	2	2	1	1	0	0
B6	2	1	1	1	2	0	B15	2	2	1	1	1	0
B7	2	3	1	1	2	0	B16	2	2	1	1	2	1
B8	2	2	0	1	2	0	B17	2	2	1	1	2	2
B9	2	2	2	1	2	0	B18	2	2	1	1	2	3

(3) 爬山法的配置集

爬山法是首先选定基准配置 C,然后从第 1 个决策点 f_0 开始,只改变 C 中 f_0 的取值,其他决策点保持不变,直到该决策点的所有取值都被覆盖,这样就生成了一组配置.比较该组配置下贪心算法生成覆盖表的规模,从中挑选出生成规模最小的配置 $C1$ 作为新的基准配置.接着,我们改变配置 $C1$ 中下一个决策点 f_1 的取值,基于配置 $C1$ 生成一组配置,从中挑选出生成规模最小的配置 $C2$ 作为新的基准配置.重复以上步骤,直到第 6 个决策点 f_5,挑选出生成覆盖表规模最小的配置 $C6$ 为止.爬山法的配置集是动态生成的,需要在具体实验中一步步产生,更多细节可见第 4.3 节.

3 条实验路线体现了 3 种不同的可配置贪心算法优化机制.利用 Pairwise 配置集的实验路线,在配置空间中均匀地选择样本来配置贪心算法,可以系统地检查各个决策点以及它们之间的相互作用对算法性能产生的贡献;利用 Base Choice 配置集的实验路线,可以更专注于观察每个决策点各种取值对算法性能的影响,最后取出各个决策点的最佳取值作为最佳配置;与之非常相似的是利用爬山法配置集的实验路线.不同之处在于,爬山法的初始配置,在探索每个决策点各个取值对算法性能影响过程中是不断变化的,它始终被更新为使算法性能最好的配置,而 Base Choice 配置集中的初始配置不变,依次变更每个决策点的各个取值而产生新配置.这 3 种路线既可以相互独立使用,也可以互相配合使用.例如,可以对利用 Pairwise 实验产生的最优配置再进行爬山法或 Base Choice 法进行实验,从而进一步改善算法性能.

4 实验结果及分析

根据以上 3 个配置集,我们配置贪心算法,并且对多个规模不一的待测系统生成覆盖表,并记录它们各自的覆盖表规模.例如表 6 中的系统 6^4(4 个参数,每个参数 6 个取值),在配置 P1 下我们能得到 59 条测试用例.实验结果将在以下各节进行具体分析.

Table 6 Size of covering array generated by Pairwise configuration suite

表 6 运行 Pairwise 配置集生成的覆盖表规模

	P1 (size)	P2 (size)	P3 (size)	P4 (size)	P5 (size)	P6 (size)	P7 (size)	P8 (size)	P9 (size)	P10 (size)	P11 (size)	P12 (size)
6^4	59	44	42	42	76	62	43	42	44	42	58	43
$5^1 3^8 2^2$	29	22	22	20	43	32	21	21	20	22	29	20
$8^2 7^2 6^2 5^2$	113	72	71	73	279	129	70	73	71	73	119	71
$3^4 4^5$	33	25	24	24	44	40	24	24	24	24	33	23

4.1 Pairwise实验

实验结果见表 6 和表 7,其中,第 1 列是待测系统,例如,$5^1 3^8 2^2$ 表示一个系统中共 11 个参数.其中,1 个参数有 5 个取值,8 个参数有 3 个取值,2 个参数有 2 个取值.表中数据记录的是每个配置下贪心算法生成的覆盖表规模的最优解,f 表示在该配置下未能生成覆盖表.比较这些数据可以看出,不同配置对生成的覆盖表规模影响是很明显的,如在配置 P2 和 P5 下,对系统 6^4 时,前者生成的覆盖表规模是 44,后者是 76,差距较大.

Table 7 Size of covering array generated by Pairwise configuration suite

表 7 运行 Pairwise 配置集生成的覆盖表规模

	P13 (size)	P14 (size)	P15 (size)	P16 (size)	P17 (size)	P18 (size)	P19 (size)	P20 (size)	P21 (size)	P22 (size)	P23 (size)
6^4	*f*	62	53	44	45	45	55	46	47	63	42
$5^1 3^8 2^2$	*f*	29	27	21	21	21	28	22	23	32	19
$8^2 7^2 6^2 5^2$	*f*	120	103	75	71	78	112	75	81	125	68
$3^4 4^5$	*f*	36	30	f	25	25	31	28	28	37	22

有时,不合适的配置还会导致死循环,以第 13 条配置(P13)为例,它对 4 个待测系统生成覆盖表均以失败告终.这里以 3^4 为例,按照配置 P13 来生成覆盖表,分析算法失败的原因.表 8 是按照该配置已生成的 11 条测试用例,此时只有 f_0 和 f_1 之间的(2,1)未被覆盖,按照 P13=(3,0,0,1,1,2)继续生成测试用例,按照 random 策略随机选择一个参数作为第 1 个参数,再按照 uncovered pairs 策略对该参数进行赋值.由于该测试用例中还未有一个参数被确定,对第 1 个参数来说,所有取值和已确定参数之间的未被覆盖对均为 0.因此,该参数的所有取值均可被用来对该参数进行赋值.根据同序值选择的策略 take first,选择第 1 个值就将第 1 个参数赋值为 0,后续参数没有一个参数的取值可以和第 1 个参数的 0 形成新的未被覆盖对,因此后续参数取值也只能为 0,依次下去,得出测试用例(0,0, 0,0).无论循环多少次,都还是生成这条测试用例,它无法覆盖新的未被覆盖对,算法从而陷入了病态循环,不能得出最终结果.从这个例子我们可以看出,当值选择策略为 uncovered pairs、同序值选择为 take first 时,极有可能发生病态循环这样的情况.那么是这两个策略本身存在缺陷吗?显然不是,若改变配置的同序值选择为 random 或者 uncovered pairs,就可以避免病态循环的发生,因此应该是这两个策略组合使用造成了算法的失败,用户在实际应用时最好避免将这两个策略同时使用.

Table 8 Covering array for 3^4

表 8 3^4 的覆盖表

Test No.	f_0	f_1	f_2	f_3	新覆盖对数
1	0	0	0	0	6
2	1	0	1	1	6
3	2	0	2	2	6
4	1	1	0	2	6
5	0	2	1	2	6
6	2	2	0	1	6
7	0	1	2	1	6
8	0	1	1	0	3
9	0	2	2	0	3
10	1	2	2	0	3
11	2	0	1	0	2

除了发现无法得出结果的配置,我们还找到了最佳配置 P23(见表 6 和表 7).在配置 P23 下,本文的 4 个研究实例都可得到规模较小的覆盖表.例如,为 6^4 生成的覆盖表规模为 42,为 $3^4 4^5$ 生成的覆盖表规模为 22,都是这一系列算法中性能最优的.因此,我们将 P23=(2,2,1,1,2,0)这一配置作为 Pairwise 实验得到的最优配置.

4.2 Base Choice实验

实验首先选定一个基准配置,然后通过 Base Choice 方法系统地探索每个决策点对生成覆盖表规模的不同影响.我们的第 1 个实验选择的基准配置是第 4.1 节中 Pairwise 实验得出的最佳配置 P23=(2,2,1,1,2,0),其对应的 Base Choice 配置表见表 5,实验结果见表 9.

由于 Base Choice 配置集中的配置都是在单独改变基准配置中某个决策点取值的情况下生成的,它们与基准配置之间只改变了一个决策点的取值,其余保持不变.因此,我们将配置集(除基准配置)中改变同一个决策点的配置放入一个集合,这样可以划分成 6 个集合,再将基准配置放入所有集合.按照这种方法,表 5 中的配置集可以如下划分为{B1,B2,B3,B4},{B1,B5,B6,B7},{B1,B8,B9,B10,B11},{B1,B12,B13},{B1,B14,B15} 和 {B1,B16,B17,B18}.下面我们分别系统地观察在表 9 中,各个集合中每个决策点对生成覆盖表规模性能的变化,以找出该决策点的最优策略.

Table 9 Size of covering array generated by Base Choice configuration suite

表 9 运行 Base Choice 配置集生成的覆盖表规模

	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16	B17	B18
$5^1 3^8 2^2$	20	21	20	19	21	20	20	21	20	20	20	29	20	20	20	20	*f*	21
$3^4 4^5$	24	26	24	23	25	24	24	25	22	23	25	33	24	23	23	25	*f*	*f*
$6^1 5^1 4^6 3^8 2^3$	33	36	34	33	37	34	33	38	33	34	39	53	34	34	34	38	*f*	35
6^4	42	42	42	41	44	43	42	41	41	43	41	55	43	42	43	44	*f*	44
$8^2 7^2 6^2 5^2$	69	70	69	69	73	70	68	72	69	70	74	114	69	69	68	71	*f*	75

(1) 循环次数

配置 B1,B2,B3 和 B4 只改变了框架的循环次数,分别循环了 10 次、1 次、5 次和 20 次,将实验结果用折线图(如图 2 所示)表示,横轴表示循环次数,纵轴表示覆盖表的规模,每条线表示对于一个实例按照各种循环次数生成的覆盖表规模(为了更准确地考察循环次数的不同影响,这里增加了循环 50 次和 100 次这两种情况),从图 2 可以看出,增加循环次数可以适当地减小覆盖表规模,但增加到一定程度后,覆盖表规模不会发生改变.例如:对于实例 3 的 $6^1 5^1 4^6 3^8 2^3$,B2(循环 1 次)生成覆盖表规模是 36,B3(循环 5 次)生成的覆盖表规模是 34,减少了 2 条测试用例;而 B4(循环 20 次)的规模是 33,减小了 1;B1(循环 10 次)的规模仍为 33,没有改变覆盖表规模.从实验可以看出:增加循环次数可以生成规模更小的覆盖表,但是循环次数增加到一定程度后,覆盖表的规模将不再发生改变;而且增加循环次数将使得覆盖表的生成时间快速增长.以 6^4 为例,循环 1 次覆盖表的生成时间为 0.094s,循环 10 次覆盖表的生成时间为 1.844s.而对于本实验中的 4 个配置,B4(循环 20 次)在多个实例中均能生成较小规模的覆盖表.

(2) 候选用例集数目

B1,B5,B6,B7 只改变了框架中候选用例集的数目,这 4 条配置下候选用例集数目分别为 10,1,5 和 20,同样,将实验数据用折线图表示(如图 3 所示,这里也考虑了候选用例集数目为 50 和 80 这两种情况).比较这 4 条配置下的贪心算法在生成覆盖表规模上的性能可以发现,增加候选用例集的数目能够减小覆盖表的规模.以实例 4 的 6^4 为例,B5(候选用例集数目是 1)生成覆盖表规模为 44,B6(候选数目是 5)覆盖表规模为 43,B1(候选数目是 10)规模为 42.不过,同样在增加到一定程度后,算法性能可能不再提高.还以 6^4 为例:当候选数目为 10 时,生成规模是 42;当候选数目为 20 时,规模仍为 42,没有改变.同样,增加候选用例集数目也会造成覆盖表生成时间过长,用户要权衡好规模和时间这两者之间的关系.从图 3 可以看出,这 4 个配置中对所有研究实例生成覆盖表规模最小的是 B7(候选用例集数目为 20).

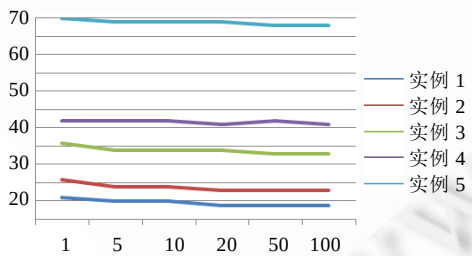

Fig.2 Size of covering array generated by repetitions

图 2 多个循环次数生成的覆盖表规模

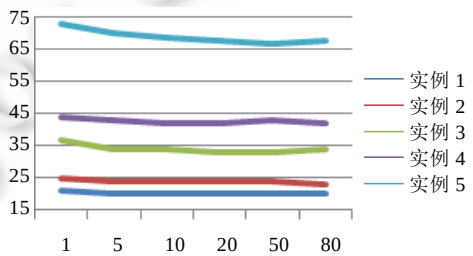

Fig.3 Size of covering array generated by candidates

图 3 多个候选用例集生成的覆盖表规模

(3) 参数排序

配置 B1,B8,B9,B10 和 B11 不同的是框架中参数排序的策略,依次是 uncovered pairs,random,density,level 和 hybrid.在这些配置下,对各个研究实例生成的覆盖表规模如图 4 所示,图中横轴表示的是 5 个实例,纵轴表示生成覆盖表的规模,每条线对应一个配置(参数排序策略)对不同研究实例生成覆盖表的规模.从图中可以观察到,各个参数排序策略效果相当,不过还是以 density 和 uncovered pairs 这两个策略最佳(其中 density 更好一些),level 性能居中,random 效果最差.本文还考虑了第 1 个参数按照 uncovered pairs、其余参数按照 random 策略的

杂交策略(hybrid).从图 4 可以看出,杂交策略的性能介于 uncovered pairs 和 random 之间.

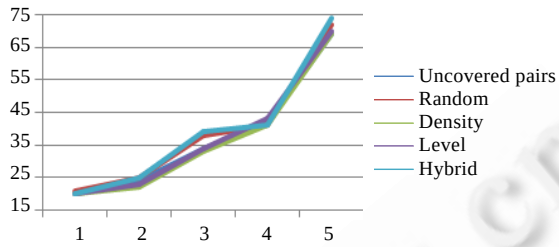

Fig.4 Size performance of the generated covering array by factor ordering selection

图 4 参数排序中各策略生成覆盖表规模的性能比较

(4) 值的选择

配置 B1,B12 和 B13 的值选择策略不同,依次是 uncovered pairs,random 和 density.实验数据如图 5 所示,从图 5 中可以看出,random 效果最差,Pairwise 的实验结果也验证了这一结论.当配置中值选择为 random 时,可以发现算法性能总是非常差.例如,研究实例 6^4,在 P14=(3,1,1,0,0,0)配置下生成的覆盖表规模为 62.另外,两个策略 uncovered pairs 和 density 效果相当,均比 random 效果要好.

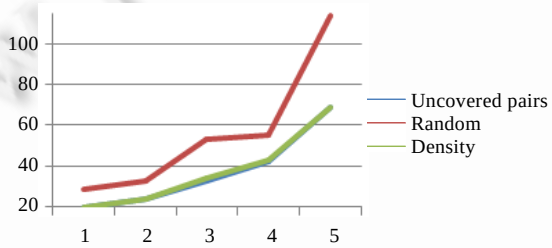

Fig.5 Size of the generated covering arrays by level selection

图 5 值选择的各个策略生成覆盖表规模

(5) 同序参数选择

在配置 B1,B14 和 B15 下,同序参数选择的策略分别是 take first,random 和 uncovered pairs.比较这 3 个配置在生成覆盖表规模上的性能,从图 6 可以看出,这 3 个策略的性能非常接近,相差不大.

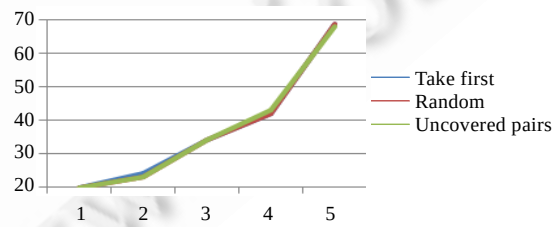

Fig.6 Comparison of factor ordering selection

图 6 同序参数选择的多个策略比较

(6) 同序值选择

在配置 B1,B16,B17 和 B18 中,同序值选择的策略依次为 random,uncovered pairs,take first,least frequently.当策略为 take first 时,所有实例均得出覆盖表,这一点在 Pairwise 实验中已有相关分析.当策略为 least frequently 时,算法偶尔也会运行失败,其原因与 take first 相似,当值选择与同序值选择策略为(uncovered pairs,least

frequently)组合时,算法可能会进入病态循环,建议用户在使用时尽量避免值选择和同序值选择中(uncovered pairs,take first)和(uncovered pairs,least frequently)这样两种组合.除了病态的策略组合,还可以找出性能最优的random,它在这些策略中的表现最为稳定,结果也最为理想.

经过以上观察,可以找出循环次数中的 20,候选用例集数目 20,参数排序中的 density,值选择中的 uncovered pairs 和 density,同序参数选择中的任意策略以及同序值选择中的 random.这些策略都是 6 个决策点中性能最优的.组合这些策略形成新的最优配置(20,20,density,density 或 uncovered pairs,–,random),即(3,3,2,1, –,0)和(3,3,2,2, –,0),其中,"–"表示任意策略.

下面分别考察这些配置的性能,结果见表 10.表 10 中,前 3 列表示值选择是 density,同序参数选择策略任选,其他参数取值不变,即为(20,20,density,density, –,random).结果表明,当值选择是 density 时,这 3 个同序参数选择策略的性能非常接近,可以任意使用.中间 3 列是值选择 uncovered pairs,同序参数选择策略任选,即为(20,20,density,uncovered pairs, –,random).结果显示,在值选择是 uncovered pairs 时,同序参数选择是 random 策略时能够生成较小规模的覆盖表.通过这个比较,我们得出了 Base Choice 实验中的最优配置为(20,20,uncovered pairs,random,random),即(3,3,2,1,0,0).

为了进一步考察初始配置对 Base Choice 实验结果的影响,在第 2 个实验中,我们随机生成一个配置(3,2,4,2,0,3),然后对它进行同样的 Base Choice 实验,得出最优配置(3,3,2,2,2,2),该配置生成覆盖表规模的性能见表 10 最后一列.结果显示,随机配置得出的结果比 Pairwise 实验中最优配置得出的结果略差.

Table 10 Performance comparison of multiple configurations

表 10 多个配置生成覆盖表规模的性能比较

	Density			Uncovered pairs			
	Random	Uncovered	Take first	Random	Uncovered	Take first	随机实验
$5^1 3^8 2^2$	20	20	20	19	20	19	22
$3^4 4^5$	23	23	23	22	22	22	24
$6^1 5^1 4^6 3^8 2^3$	34	34	33	33	33	34	35
6^4	42	42	42	39	41	41	42
$8^2 7^2 6^2 5^2$	69	68	69	67	68	67	71

4.3 爬山实验

爬山法是从另一个角度来探索贪心算法框架中每个决策点对生成覆盖表规模的不同影响,这里我们先以 Pairwise 实验中得出的最佳配置作为初始配置,利用爬山法进行优化.先选定配置 C 为第 4.1 节 Pairwise 实验中得出的最优配置 C=P23=(2,2,1,1,2,0),由于爬山法的配置集是动态生成的,我们将按照以下决策点一步步为它生成配置集.

(1) 循环次数

首先改变 C=(2,2,1,1,2,0)中循环次数这一决策点的取值,直到循环次数的 4 个取值均被覆盖为止,生成的配置为 H1=(0,2,1,1,2,0),H2=(1,2,1,1,2,0),H3=(3,2,1,1,2,0).将 C 和这 3 个配置一起进行实验,结果见表 11.比较这组数据可以看到,H3(循环 20 次)效果最佳,因此选择 H3=(3,2,1,1,2,0)作为循环次数中选定的配置 $C1$.

(2) 候选用例集数目

接着,依次改变 $C1$=(3,2,1,1,2,0)中候选用例集的取值,生成的配置为 H4=(3,0,1,1,2,0),H5=(3,1,1,1,2,0)和 H6=(3,3,1,1,2,0).将 $C1$ 和这 3 个配置一起进行比较,实验结果见表 12.可以明显观察到,H6(候选用例集数目 20)时性能最佳,因此选择 H6=(3,3,1,1,2,0)作为候选用例集中选定的配置 $C2$.

(3) 参数排序

对 $C2$=(3,3,1,1,2,0)改变配置中参数排序的策略,依次生成配置 H7=(3,3,0,1,2,0),H8=(3,3,2,1,2,0),H9=(3,3,3,1,2,0)和 H10=(3,3,4,1,2,0).将 $C2$ 和这 4 个配置一起进行实验,结果见表 13.可以观察到,使用 density 策略对于多个实例均能生成较小的测试用例集,因此选择 H8=(3,3,2,1,2,0)作为参数排序中选定的配置 $C3$.

(4) 值的选择

通过变化 $C3$=(3,3,2,1,2,0)中值选择的取值,生成了配置 H11=(3,3,2,0,2,0)和 H12=(3,3,2,2,2,0),比较这 3 个配置,实验结果见表 14.可以观察到,random 效果最差,uncovered pairs 和 density 这两个策略的性能比较接近,效果均比 random 要好.不过,uncovered pairs 性能比 density 更为理想一些,在实例中它能够得出更小的覆盖表规模,这里选择它作为最优策略,选择 $C3$=(3,3,2,1,2,0)作为值选择中选定的配置 $C4$.

(5) 同序参数选择

改变 $C4$=(3,3,2,1,2,0)中同序参数选择的取值,依次生成配置 H13=(3,3,2,1,0,0)和 H14=(3,3,2,1,1,0),实验结果见表 15.比较同序参数选择中的 3 个策略,发现其性能相似,不过,random 效果更为理想,因此选取 H13=(3,3,2,1,0,0)作为该决策点选定的配置 $C5$.

(6) 同序值选择

依次改变 $C5$=(3,3,2,1,0,0)中同序值选择的取值,生成配置 H15=(3,3,2,1,0,1),H16=(3,3,2,1,0,2)和 H17=(3,3,2,1,0,2),实验结果见表 16.对这些策略进行比较,可以观察到,random 最优,因此选择 $C5$ 作为该部分选定的配置 $C6$.

Table 11 Performance comparison of repetitions on size of the generated covering array

表 11 多个循环次数在生成覆盖表规模上的性能比较

	C	H1	H2	H3
$5^1 3^8 2^2$	20	21	20	19
$3^4 4^5$	24	26	24	23
$6^1 5^1 4^6 3^8 2^3$	34	36	34	34
6^4	42	42	42	41
$8^2 7^2 6^2 5^2$	69	70	69	69

Table 12 Performance comparison of candidates on size of the generated covering array

表 12 多种候选用例集数目生成覆盖表规模的性能比较

	$C1$	H4	H5	H6
$5^1 3^8 2^2$	19	21	20	19
$3^4 4^5$	23	25	23	23
$6^1 5^1 4^6 3^8 2^3$	34	36	34	33
6^4	42	45	42	41
$8^2 7^2 6^2 5^2$	69	75	70	68

Table 13 Performance comparison of factor ordering selection on size of the generated covering array

表 13 多个参数排序策略生成覆盖表规模性能比较

	$C2$	H7	H8	H9	H10
$5^1 3^8 2^2$	20	20	19	20	20
$3^4 4^5$	23	24	22	23	24
$6^1 5^1 4^6 3^8 2^3$	33	38	33	33	36
6^4	41	41	41	42	42
$8^2 7^2 6^2 5^2$	67	71	66	67	70

Table 14 Performance comparison of level ordering selection on size of the generated covering array

表 14 多个值选择策略生成覆盖表规模性能比较

	$C3$	H11	H12
$5^1 3^8 2^2$	19	26	20
$3^4 4^5$	22	31	23
$6^1 5^1 4^6 3^8 2^3$	34	51	34
6^4	41	52	41
$8^2 7^2 6^2 5^2$	67	105	69

Table 15 Performance comparison of factor tie-breaking on size of the generated covering array

表 15 同序参数选择性能比较

	$C4$	H13	H14
$5^1 3^8 2^2$	19	19	20
$3^4 4^5$	22	22	22
$6^1 5^1 4^6 3^8 2^3$	34	33	33
6^4	41	39	41
$8^2 7^2 6^2 5^2$	67	67	68

Table 16 Performance comparison of level tie-breaking on size of the generated covering array

表 16 多个同序值选择策略生成覆盖表规模性能比较

	$C5$	H15	H16	H17
$5^1 3^8 2^2$	19	23	f	21
$3^4 4^5$	22	24	f	24
$6^1 5^1 4^6 3^8 2^3$	33	34	f	36
6^4	42	42	f	43
$8^2 7^2 6^2 5^2$	67	71	f	f

通过这 6 轮实验,我们得出了最优配置 C^*=$C6$=(3,3,2,1,0,0).

为了进一步检验爬山法的优化效果是否取决于初始配置,我们也随机生成一个配置(2,1,3,1,1,0),同样按照以上实验步骤进行了爬山法实验,得出最优配置仍为(3,3,2,1,0,0).初步结果显示,当初始配置为随机配置和 Pairwise 实验中最优配置时,分别进行相同的爬山法实验所得出的结果是一样的.

4.4 验证实验

在 Pairwise 实验中得出最优配置 Bst1=(2,2,1,1,2,0),然后我们先通过 Base Choice 和爬山方法对配置 Bst1 进行优化,Base Choice 对配置 Bst1 进行优化后得出最优配置 Bst2=(3,3,2,1,0,0),爬山法对配置 Bst1 进行优化后

也得出最优配置 Bst2=(3,3,2,1,0,0).为了检查初始配置的不同对 Base Choice 实验和爬山实验的优化结果是否存在很大影响,我们又分别对随机生成的配置进行 Base Choice 实验,得出最优配置 Bst3=(3,3,2,2,2,2),进行爬山实验,得出最优配置 Bst2=(3,3,2,1,0,0),其中,Bst3=(3,3,2,2,2,2)是已有的 DDA 算法.

对于以上通过实验找到的性能较好的配置,我们从以下两个方面验证其有效性:1) 它们对其他实例是否适用,即是否也能生成规模较小的覆盖表;2) 与已有的贪心算法 AETGmTCG 及 DDA 相比,这些配置是否具有优势.另外我们又选择了一些新的实例来对这些配置和已有算法进行实验,结果见表 17,其中,已有贪心算法的数据来自于文献[2,9–12];缺失部分的结果来自文献[12],用斜体表示.表中第 2 列~第 4 列记录的是我们找出的最优配置,后 3 列是已有算法 DDA,AETG 和 TCG.与已有算法相比,我们的最优配置在生成覆盖表规模的性能上是具有竞争力的.Bst2 表现得更为突出,它在一些情况下略优于已有的 3 种算法.例如,在配置 Bst2 下,为 $5^1 4^4 3^{11} 2^5$ 生成的覆盖表规模为 26,而在 3 种已有算法下生成覆盖表的规模分别是 27,30 和 30;又如,在配置 Bst2 下,为 $4^{15} 3^{17} 2^{29}$ 生成覆盖表规模为 34,而在 3 种已有算法下覆盖表的规模分别为 35,41 和 35.

Table 17 Comparison between the prioritized configuration and the existed algorithms

表 17 最优配置和已有算法的比较

	Bst1 (Bst/Avg/Worst)	Bst2 (Bst/Avg/Worst)	DDA (Bst3)	AETG	TCG
3^{13}	19/19.4/20	18/19/20	18	15	20
$5^1 3^8 2^2$	20/20.6/21	19/20.3/22	21	19	20
$6^1 5^1 4^6 3^8 2^3$	34/35/36	33/34.05/37	34	34	33
$5^1 4^4 3^{11} 2^5$	27/28.2/29	26/27.6/29	27	30	30
$4^{15} 3^{17} 2^{29}$	35/36.3/38	34/34.59/36	35	41	35
$7^1 6^1 5^1 4^5 3^8 2^3$	42/43/45	42/42.75/44	43	45	45
4^{40}	45/46.7/48	43/44.5/45	43	42	46

尽管表 17 已显示出优化的贪心算法确实比已有的 DDA,AETG 和 TCG 算法可以产生更好的覆盖表,但我们仍然需要进行更进一步的验证:优化的贪心算法是否可以在更广泛的范围内取得很好的效果?这个效果与最好的结果有多大的差距?

为了回答这两个问题,我们从 Charlie Colbourn 维护的覆盖表专门网站上有代表性地选择了 50 个实例[17],分别利用 Bst1 和 Bst2 配置的贪心算法来为这 50 个实例产生覆盖表,然后与该网站上公布的最优结果进行比较.表 18 中列出了这些实例、目前关于这些实例的最好结果以及本文优化方法产生的结果.

Table 18 50 instances from Charlie Colbourn's website[17]

表 18 在 Charlie Colbourn 网站上选择的 50 个实例[17]

Cases	3^{10}	3^{20}	3^{33}	3^{48}	3^{92}	3^{101}	3^{122}	3^{146}	4^{15}	4^{27}
Best	14	15	18	20	21	22	23	24	26	29
Bst1	16	22	25	28	33	33	34	36	34	40
Bst2	16	20	24	26	30	31	33	34	32	39
Cases	4^{35}	4^{57}	4^{66}	4^{72}	4^{83}	4^{139}	5^9	5^{17}	5^{38}	5^{43}
Best	31	35	36	37	38	41	35	42	48	49
Bst1	44	49	52	52	54	60	42	53	68	70
Bst2	42	48	50	50	52	59	40	52	66	68
Cases	5^{77}	5^{96}	5^{111}	6^8	6^{23}	6^{42}	6^{65}	6^{82}	6^{104}	6^{124}
Best	58	60	64	42	67	74	77	83	90	95
Bst1	82	87	89	56	83	99	111	118	124	130
Bst2	79	83	86	53	81	95	108	114	121	125
Cases	7^{11}	7^{20}	7^{63}	7^{80}	7^{91}	7^{120}	8^{20}	8^{80}	8^{90}	8^{121}
Best	65	87	91	105	111	123	108	120	133	148
Bst1	84	105	147	155	160	172	135	201	208	221
Bst2	81	102	144	152	156	167	131	196	201	216
Cases	9^{10}	9^{30}	9^{100}	9^{130}	10^7	10^{15}	10^{30}	10^{70}	10^{98}	10^{121}
Best	81	148	160	182	113	136	130	202	215	216
Bst1	130	191	266	282	140	184	233	296	322	339
Bst2	124	187	259	274	134	179	228	289	315	332

为了便于在同一张图上更清楚地看出这些结果之间的差异,我们对每个实例 $Case_i$ 的各种结果按照如下公

式进行标准化:

$$ST\text{-}Case_i(X)=Case_i(X)/Case_i(Best).$$

X 可以取 Best,Bst1 和 Bst2.其中,$Case_i(Best)$表示第 i 个实例在目前 Charlie Colbourn 网站上公布的最优的覆盖表规模,$Case_i(Bsti)$表示利用配置 Bst$i$ 产生的覆盖表规模,i=1 或者 i=2.

图 7 显示了优化的贪心算法在 50 个实例上的性能,这里,50 个实例按照表 17 中的顺序进行编号.例如,第 37 个实例就是表 18 中第 4 行第 7 个实例 8^{20},对于该实例,最好的覆盖表规模是 108 条测试用例.使用 Bst1 配置的贪心算法产生 135 条,而使用 Bst2 配置的贪心算法产生 131 条.这两种方法产生的覆盖表都与最好的覆盖表相当接近.从整个图 7 所示的 50 个实例上可以看出,Bst2 配置略好于 Bst1,这两个配置对应的贪心算法产生的覆盖表是最好覆盖表的 1.1~1.8 倍,是最优覆盖表的很好的近似.

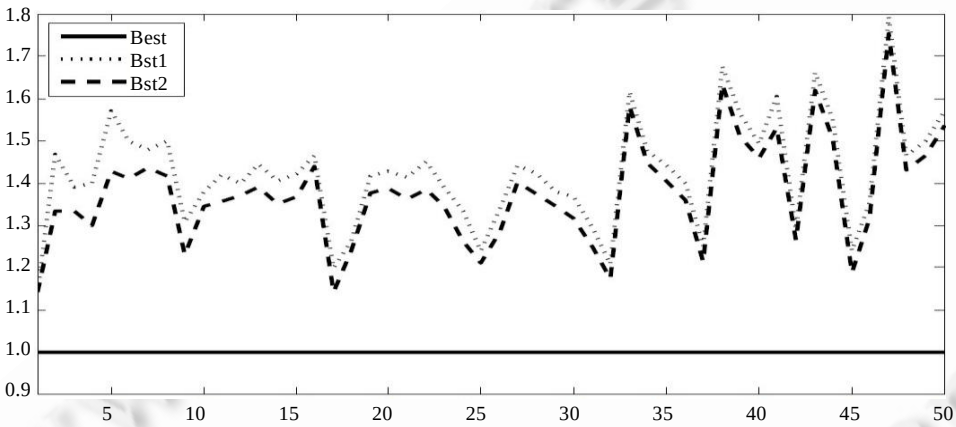

Fig.7 Performance of the prioritized greedy algorithm on 50 instances

图 7 优化的贪心算法在 50 个实例上的性能

4.5 结论以及相关工作比较

Bryce 等人通过统计工具 ANOVA[16]对几千种贪心算法进行实验,得出以下几点结论:

(1) 值选择是最具影响力的,其中 density 效果最佳;

(2) 在值选择固定使用 density 的前提下,参数排序对于测试用例集规模的影响最为明显;

(3) 在测试用例集规模最为关注时,增加循环次数比增加候选用例集数目其效果更为理想.

本文采用 3 种不同的组合设计方法对框架下生成的大量贪心算法进行了合理的抽样,抽取其中一小部分进行研究.实验结果表明,我们的实验虽然数据量较小,但是也和 Bryce 等人得出了一致的结论,例如增加循环次数和候选用例集数目能够对覆盖表规模略有改进、在值选择中密度算法效果较好等等.由于两个实验的研究方法不同,产生的结论侧重点也有所不同.Bryce 等人对框架的各个决策点通过统计工具按照重要程度进行了排序,值选择最为重要,其次是参数选择,循环次数比候选用例集数目更为重要;而本文更为关注决策点中配置的组合对生成覆盖表的改进,找出了框架下一个好的配置(20,20,density,uncovered pairs,random,random).

我们的实验结果是 Bryce 结论的一个补充和验证,得出了以下结论:

(1) 增加循环次数和候选用例集数目能够对覆盖表规模略有改进,但同时会增加时间成本,而且当这两个因素增加到一定程度时,覆盖表规模也不再缩小,因此在使用时,循环次数设在 10~20 次,候选用例集数目在 10~20 个较为合适;

(2) 参数排序中所有策略性能都比较接近,但以 density 最优;

(3) 利用 random 策略给参数赋值时,其生成覆盖表规模较大;

(4) 在同序值选择中,random 性能最佳,但值选择和同序值选择是(uncovered pairs,take first)和(uncovered

pairs,least frequently)组合时易陷入死循环,无法生成覆盖表.

5 总结与展望

本文研究了具有 6 个决策点的覆盖表生成贪心算法框架,任意组合框架下各个决策点可以生成一大类贪心算法.为了有效评估这一类算法在生成覆盖表规模上的性能(回答在实验设计中提出的 4 个问题),我们设计实现了一个可配置贪心算法下的覆盖表自动生成工具作为实验平台,它能够运行任意组合配置的贪心算法.

我们从不同的路线设计了 3 组实验,系统地研究框架中各个决策点以及相互作用对生成覆盖表规模的影响,对 Base Choice 实验和爬山实验采用了指定基准用例和随机基准用例这两种方法,得出以下结论:

(1) 算法框架下,各个决策点的配置对其生成的覆盖表具有明显的影响.例如,当值选择是 random 策略时,会生成规模较大的覆盖表,当值选择和同序值选择是(uncovered pairs,take first)时,会陷入病态循环;

(2) 通过实验,在某些实例中找出了最优配置,它能够生成较小规模的覆盖表,例如(20,20,density,uncovered pairs,random,random)和(20,20,density,density,−,random);

(3) 在验证实验中,通过其他实例对最优配置进行了检测,结果显示,实验中找出的最优配置具有一定的普遍性,对其他实例也能够生成较小规模的覆盖表;

(4) 实验还将最优配置与已有算法 AETG,TCG 及 DDA 进行了比较,结果显示,找出的最优配置是有优势的,为多数实例生成的覆盖表都好于已有算法;

(5) 通过对优化配置贪心算法产生的覆盖表与最优覆盖表规模进行比较后发现,在最优覆盖表无法获得的情况下,优化的贪心算法生成的覆盖表可以作为最优覆盖表很好地近似.

本文现有的工作结果可以提供两个方面的用途:

- 第一,本文获得的最优配置可以作为现有可配置贪心(贪心算法框架)算法的经验配置使用,以用于计算其他覆盖表;
- 第二,在一些刻意追求最小规模覆盖表的应用场景下,本文提供的 3 种实验路线交叉结合使用的方法可以明显改善生成的覆盖表质量.

在下一步工作中,我们还将在更广泛的实验基础上寻求和验证更优的框架配置,具体包括以下几个方面:

(1) 扩大实验规模,增加实验运行次数,深入探索框架中病态循环的内在原因.本文涉及的系统参数个数规模不超过 200,未来我们将尝试研究为几百甚至几千个参数的系统生成覆盖表;

(2) 研究参数排序和值选择的杂交策略,并关注已有部分测试用例(seed)和约束(constraints)这两种情况;

(3) 进一步研究目前优化后的贪心算法产生的近似最优解与实际最优解之间的接近程度,研究优化的贪心算法的适用范围等;

(4) 更为透彻地研究决策点之间的相互关系,为用户提供更为有效的经验配置参数,从而为覆盖表生成的贪心算法的设计和优化提供理论和实践基础;

(5) 研究在可配置贪心算法的经验配置参数不可行的情况下,直接采用本文提出的 3 种实验路线寻找最优覆盖表的可行性,特别是研究爬山法、Base Choice 法与 Pairwise 法分别单独使用时的效果比较,以及它们相互结合后的效果比较,从而研究出效率更高、效果更好的实验路线.

致谢 匿名审稿人给本文提出了非常有帮助的修改意见,为改进本文质量起到了非常重要的作用.同时,研究生吴化尧同学帮助我们进行了大量的实验以进一步验证本文结论,在此一并表示感谢.

References:

[1] Nie CH, Leung H. A survey of combinatorial testing. ACM Computing Survey, 2011,43(2):1−29. [doi: 10.1145/1883612.1883618]

[2] Kuhn D, Reilly M. An investigation of the applicability of design of experiments to software testing. In: Proc. of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop. NASA Goddard Space Flight Center, 2002. 1−5.

[3] Grindal M, Offutt AJ, Andler SF. Combination testing strategies: A survey. Software Testing, Verification, and Reliability, 2005, 15(3):167−199. [doi: 10.1002/stvr.319]

[4] Grindal M, Lindstrom B, Offutt AJ, Andler SF. An evaluation of combination strategies for test case selection. Empirical Software Engineering, 2006,11:583−611. [doi: 10.1007/s10664-006-9024-2]

[5] Yan J, Zhang J. Combinatorial testing: Principle and methods. Ruan Jian Xue Bao/Journal of Software, 2009,20(6):1393−1405 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3497.htm [doi: 10.3724/SP.J.1001.2009.03497]

[6] Williams AW, Prober RL. A practical strategy for testing pair-wise coverage of network interfaces. In: Proc. of the 7th Int'l Symp. on Software Reliability Engineering (ISSRE'96). White Plaints, 1997. 246−254. [doi: 10.1109/ISSRE.1996.558835]

[7] Nurmela KJ. Upper bounds for covering arrays by tabu search. Discrete Applied Mathematics, 2004,138(1-2):143−152. [doi: 10.1016/S0166-218X(03)00291-9]

[8] Cohen MB, Gibbons PB, Mugridge WB, Colbourn CJ. Constructing test suites for interaction testing. In: Proc. of the 25th Int'l Conf. on Software Engineering (ICSE 2003). Portland, 2003. 38−48. http://dx.doi.org/10.1109/ICSE.2003.1201186

[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, 1997,23(7):437−444. [doi: 10.1109/32.605761]

[10] Cohen DM, Dalal SR, Kajla A, Patton GC. The automatic efficient tests generator (AETG) system. In: Proc. of the 5th IEEE Int'l Symp. on Software Reliability Engineering. 1994. 303−309. [doi: 10.1109/ISSRE.1994.341392]

[11] Cohen DM, Dalal SR, Fredman ML, Parelius J, Patton GC. The combinatorial design approach to automatic test generation. IEEE Software, 1996,13(5):83−88. [doi: 10.1109/52.536462]

[12] Tung Y, Aldiwan W. Automating test case generation for the new generation mission software system. In: Proc. of the IEEE Aerospace Conf. 2000. 431−437. [doi: 10.1109/AERO.2000.879426]

[13] Colbourn CJ, Cohen MB, Turban RC. A deterministic density algorithm for pairwise interaction coverage. In: Proc. of the IASTED Int'l Conf. on Software Engineering. 2004. 242−252. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.3990

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

[15] Tai KC, Lei Y. A test generation strategy for pairwise testing. IEEE Trans. on Software Engineering, 2002,28(1):109−111. [doi: 10.1109/32.979992]

[16] Bryce RC, Colbourn CJ, Cohen MB. A framework of greedy methods for constructing interaction test suites. In: Proc. of the 27th Int'l Conf. on Software Engineering (ICSE 2005). New York: ACM Press, 2005. 146−155. [doi: 10.1145/1062455.1062495]

[17] Colbourn CJ. Covering array tables for *t*=2,3,4,5,6. http://www.public.asu.edu/~ccolbou/src/tabby/catable.html

附中文参考文献:

[5] 严俊,张健.组合测试:原理与方法.软件学报,2009,20(6):1393−1405. http://www.jos.org.cn/1000-9825/3497.htm [doi: 10.3724/SP.J.1001.2009.03497]

聂长海(1971－),男,江苏南京人,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.

E-mail: changhainie@nju.edu.cn

蒋静(1987－),女,助理工程师,主要研究领域为软件测试.

E-mail: fannyjj-1987@163.com