中国科学: 信息科学 2014 年 第 44 卷 第 10 期: 1345–1360
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS
测试技术专刊 · 论文

# 基于分支限界搜索框架的测试用例自动生成

邢颖[①②*], 宫云战[①], 王雅文[①③], 张旭舟[①]

① 北京邮电大学网络与交换技术国家重点实验室, 北京 100876
② 辽宁工程技术大学电子与信息工程学院, 葫芦岛 125105
③ 中国科学院计算技术研究所, 北京 100190
* 通信作者. E-mail: lovelyjamie@yeah.net

收稿日期: 2014–05–16; 接受日期: 2014–08–18
国家自然科学基金 (批准号: 61202080, 91318301) 和国家高技术研究发展计划 (863)(批准号: 2012AA011201) 资助项目

**摘要** 作为软件测试领域的一个基本问题和热点问题, 面向路径的测试用例自动生成有着特殊的重要意义. 面向路径的测试用例生成本质上是一个约束满足问题, 并通过搜索算法求解. 着眼于提升搜索算法的效率, 本文提出了一种新的智能算法, 将分支限界和爬山法进行了有机的整合, 分支限界作为全局搜索算法, 而爬山法作为局部搜索算法, 发挥各自的优势来对测试用例的解空间进行搜索.

**关键词** 面向路径 测试用例生成 约束满足问题 分支限界 爬山法

## 1 引言

计算机已经应用到国民经济和社会生活的方方面面, 而软件作为计算机的灵魂则起着至关重要的作用. 失效的软件可能会造成巨大的经济损失, 甚至危及到人民的生命安全. 软件质量问题现已成为制约计算机发展的主要因素之一. 许多计算机科学家在展望计算机科学发展方向和策略时都把提高软件质量放在优先于提高软件功能和性能的地位.

软件测试在今后较长时间内仍将是保障软件质量的重要手段. 软件测试在软件开发周期中所占比重很大, 据统计占整个软件开发成本的 50% 以上 [1], 但是其效率很低 [2]. 人工测试以及半自动化测试仍广泛存在, 所以软件测试的自动化变得非常急迫. 而作为软件测试 (包括白盒测试和黑盒测试) 里的一个基本问题, 面向路径的测试用例自动生成则尤为重要, 这是因为白盒测试中的许多诸如控制流测试和数据流测试问题以及黑盒测试中的一些问题都可以归结为面向路径的测试用例生成问题. 其非形式化描述为: 给定一个程序 $P$ 和 $P$ 中一条路径 $W$, 设 $P$ 的输入空间为 $D$, 求 $x{\in}D$, 使得 $P$ 以 $x$ 为输入运行, 所经过的路径为 $W$. 这个问题的实质在于约束系统的建立和求解 [3].

面向路径的测试用例自动生成方法 [4] 目前主要包括基于符号执行的静态方法和基于程序实际执行的动态方法等. 动态方法由于初始值的随机性以及动态执行的不确定性, 经常会导致大量的迭代, 甚至会产生迭代溢出. 而基于符号执行的静态测试用例生成方法不实际运行程序, 通过静态分析和模拟的手段来生成测试用例, 开销较小, 是测试领域的一个重要分支.

正因为面向路径的测试用例生成的本质在于求解约束系统, 研究人员对于基于约束求解的测试用例生成做了大量研究. Demillo 和 Offutt [5] 提出基于约束的测试用例自动生成, 使用代数约束来描述

**引用格式:** 邢颖, 宫云战, 王雅文, 等. 基于分支限界搜索框架的测试用例自动生成. 中国科学: 信息科学, 2014, 44: 1345–1360, doi: 10.1360/N112014-00142

测试用例, 通过变量区间的比较和阈值的设定进行变量区间的缩减, 使用二分搜索方法, 但是缺乏启发信息引导搜索的方向. Gallagher 等 [6] 开发的 Adtest 每次只考虑一个判断谓词或者一个输入变量, 采用迭代的方法求解约束. 这种方法的缺点是效率比较低, 难以处理真实的程序. Gupta 等 [7] 利用迭代逼近法, 求取满足选定路径上所有谓词的输入值. 该算法由于只采用线性函数来进行迭代, 因此只对线性函数有效, 而对于非线性函数则效果较差. Robschink 等 [8] 提出的方法先将整个程序静态地转换成一个静态单一赋值 (static single assignment, SSA) 形式, 将程序切片与路径条件 — 约束求解相结合, 对于系统依赖图中的路径 $W$, 确定 $W$ 执行的必要条件, 简化该路径条件, 然后用约束求解器求解. 由于该方法需要将被测试的程序 (路径上的语句) 转换成 SSA, 故所建立的约束系统都会很大, 可能包括与求解问题无关的变量, 并且对于线性路径约束是不完备的. 斯坦福大学的 Cadar 等 [9] 开发了一个符号执行工具 KLEE, 使用了一系列的约束求解优化算法, 通过分析约束和变量之间的相关性, 将约束表达式划分为相互独立的子集来提高约束求解的效率, 从而达到高覆盖率的目标. 针对复杂数据类型变量的表示和存储问题, 北京邮电大学网络与交换技术国家重点实验室的 (code testing system, CTS) 项目组提出了面向测试用例生成的抽象内存模型 [10] 用来存储动态数据类型约束, 模拟程序实际语义, 可以解决指针的别名、数组的变下标等问题, 并支持链表、树、字符串等动态数据类型.

在 CTS 抽象内存模型平台上, 本文提出了一个静态测试用例生成方法. 本文的贡献可概括为以下几个方面:

(1) 提出了结合分支限界和爬山法的约束求解框架;

(2) 提出了一个高效的变量排序算法来连接分支限界和爬山法;

(3) 通过启发式选取变量初值和目标函数的定义提高爬山法搜索效率.

本文对于爬山法包括启发式初始值选取策略都进行了对比实验, 并同现在比较流行的动静态测试用例生成方法都进行了覆盖率的对比. 结果表明本文方法明显提升了测试用例生成的效率, 在覆盖率和生成时间等方面优于相对比的方法.

## 2 背景介绍

### 2.1 分支限界

分支限界 (branch and bound, BB) 算法在人工智能组合问题求解中占据了很重要的地位, 有效地解决了背包问题、旅行商问题等经典问题. 分支限界法常以广度优先或以最小耗费 (最大效益) 优先的方式搜索问题的解空间树. 分支限界法类似于回溯法, 也是一种在问题的解空间树 T 上搜索问题解的算法. 但在一般情况下, 回溯法的求解目标是找出 T 中满足约束条件的所有解, 而分支限界法的求解目标则是找出满足约束条件的一个解, 或是在满足约束条件的解中找出某种意义下的最优解. 而对于测试用例生成来说, 只要生成满足路径约束的一个解即可, 所以选择分支限界法更为适合.

从活结点表中选择下一扩展结点的不同方式导致了不同的分支限界法, 最常见的有队列式分支限界法和优先队列式分支限界法两种. 由于对于下一个扩展节点的选择关系到算法的效率问题, 所以我们在实际的执行过程中会根据多种启发策略选择最符合测试目标的下一个节点, 这就是我们认为的“最好的” 节点, 即我们采用优先队列式分支限界法. 分支限界算法采用一定的策略空间树进行剪枝, 缩小了搜索范围; 同时它也根据限界策略来自适应地调整搜索方向, 优先搜索可能含有最优解的分支.

### 2.2 爬山法

爬山法 (hill climbing, HC) 是一个著名的局部搜索算法. 爬山法采用逐步试探的方法, 这个过程

类似于在目标函数的曲线上爬山. 在这座山上, 山峰代表局部最优解, 而洼处则代表较差的局部解. 在没有启发策略的爬山法中, 对当前解的邻域是随机进行评估的, 而更好的爬山法则应该通过一定的评估函数来确定下一步的方向. 它首先从一个随机选取的初始值开始进行搜索, 对这个初始值的邻域进行评估, 检查是否有更好的候选解. 如果有, 则用更好的候选解替换当前解. 同时对于新的当前解的邻域进行评估. 如果发现更好的候选解, 则继续进行替换, 直到再无更好的候选解可以替换当前解. 爬山法执行简单并能快速给出结果. 但是使用爬山法的搜索容易陷入局部极值, 而非全局最优解. 这种情况说明搜索结束于并非最优解的一个山峰, 而放弃了其他解空间的搜索. 此时认为邻域内再无比当前解更好的解. 由此可见爬山法对于初始值的依赖很严重.

本文选取爬山法作为每个变量的局部搜索算法, 主要是考虑到其快速收敛的特性, 而且局部极值也可以成为测试用例, 即测试用例生成没有最优解的要求. 同时我们对其初始值选取策略进行了优化, 主要是结合二分法.

### 2.3 区间运算

区间运算是用区间集合代替具体数值的数学方法, 可以更保守地描述一种可能性取值. 区间 (interval) 是一片形如 [min, max] 的连续取值区域, 其中 min 和 max 分别是其上下界. 而区间集 (domain) 是区间的集合. 一个确定值可以表示成 min 和 max 相等的区间, 比如 [5, 5]. 区间运算在区间上定义了一系列运算规则. 它从程序的入口开始, 分析和计算每个变量的可能取值范围, 并为下一步的程序分析提供可靠的区间信息. 令两个区间分别为 $X=[\underline{x}, \overline{x}]$, $Y=[\underline{y}, \overline{y}]$, 以下列举了一些本文中用到的区间运算规则.

(1) $X+Y=[\underline{x}+\underline{y},\overline{x}+\overline{y}]$;

(2) $X-Y=[\underline{x}-\underline{y},\overline{x}-\overline{y}]$;

(3) $X\cap Y=[\max(\underline{x},\underline{y}),\min(\overline{x},\overline{y})]$.

## 3 算法框架

### 3.1 问题定义

许多测试用例生成问题都涉及到待测程序 $P$ 的控制流图 (CFG). 在本文中, 程序 $P$ 的控制流图是一个有向图 $G=(N,E,i,o)$, 其中 $N$ 是语句节点的集合, $E$ 是有向边的集合, $i$ 和 $o$ 分别是唯一的入口和出口节点. 每个节点 $n\in N$ 代表程序中的一条语句, 而每条边 $e=(n_r,n_t)\in E$ 则代表从节点 $n_r$ 到节点 $n_t$ 的控制转移. 对应于判断语句 (比如 if) 的节点为分支节点, 分支节点的出边叫做分支. 我们所说的路径就是 CFG 中的一个节点序列 $W=(n_1,n_2,\ldots,n_q)$, 其中对于 $1\leqslant r<q$, 有 $(n_r,n_{r+1})\in E$.

如果存在一组输入, 当执行这组输入的时候, 所经过的路径为 $W$, 则称路径 $W$ 可达, 否则称路径 $W$ 不可达. 这样, 面向路径的测试数据生成问题可以被定义成一个约束满足问题: $X$ 是输入变量的集合$x_1,x_2,\ldots,x_n$,$D=D_1,D_2,\ldots,D_n$ 是区间集 (即 Domain, 由区间 interval 构成, 本文中为了便于说明将区间集简称为区间) 的集合, 其中 $D_i\in D(i=1,2,\ldots,n)$ 是可能赋给 $x_i$ 的所有值的集合. 该问题的一个解是每个变量都有在其区间内的一个确定值, 表示成 $V=V_1,V_2,\ldots,V_n$, $V_i\in D_i$, 它使得待覆盖路径可达. 采用图 1 的例子来说明这个问题. 图 1 中显示的是一个被测程序 test 及其对应的控制流图, 其中 if_out_6, if_out_7, if_out_8, if_out_9 和 exit_10 是虚节点. 采用分支覆盖, 有 5 条待覆盖路径, 分别是 Path1: 0 → 1 → 9 → 10, Path2: 0 → 1 → 2 → 8 → 9 → 10, Path3: 0 → 1 → 2 → 3 → 7 → 8 → 9 → 10, Path4: 0 → 1 → 2 → 3 → 4 → 6 → 7 → 8 → 9 → 10 和 Path5: 0 → 1 → 2 → 3 → 4 → 5 → 6 →

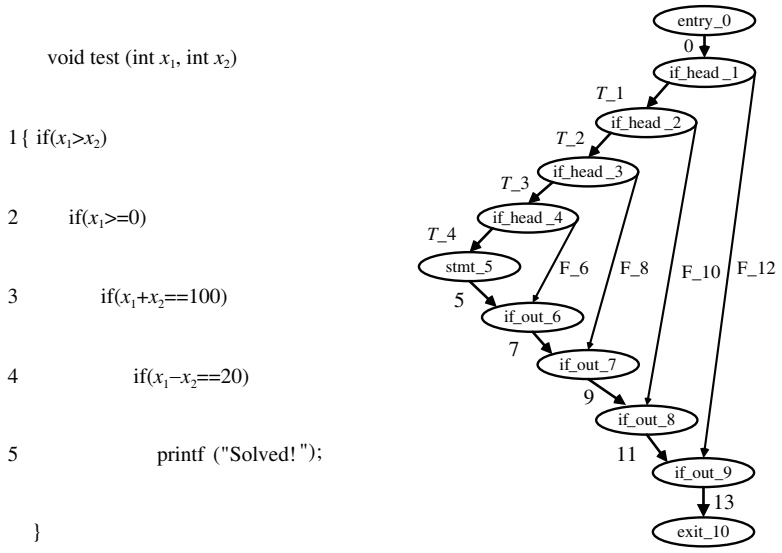



**图 1 程序 test 及其对应的控制流图**

**Figure 1** Program test and its corresponding CFG

7 → 8 → 9 → 10, 路径上的数字代表节点. 假如 Path5 是待覆盖路径如图中加粗显示, 则我们的工作就是从$D_1$, $D_2$中为 $x_1$ 和 $x_2$ 选择 $V$={$V_1$,$V_2$}, 使得以 {$x_1$↦$V_1$,$x_2$↦$V_2$} 为输入运行 test 时, 所覆盖的路径为 Path5. Path5 上包含 4 个分支节点 if_head_1, if_head_2, if_head_3 和 if_head_4, 以及 4 个分支 $T$_1, $T$_2, $T$_3 和 $T$_4 包含着要满足的约束.

本文中提到的变量都是符号变量. 在搜索过程中, 变量被分成三个集合: 已赋值变量 (past variables, 简称 PV)、当前变量 (current variable) 和未赋值变量或自由变量 (future variables, 简称 FV).

如上所述, 面向路径的测试用例生成问题是一类约束满足问题, 这类问题需要通过合适的搜索算法来求解. 一般来说, 搜索算法包括全局搜索算法和局部搜索算法两种. 全局搜索算法能够克服搜索中的局部极值问题, 从而可以发现全局最优解, 当然其代价也较大. 而局部搜索算法比较容易陷入局部极值, 但效率更高、实现更容易. 对于面向路径的测试用例生成问题, 全局搜索可以比局部搜索获得更高的覆盖率, 但是其计算代价更大. Harman 和 McMinn [11] 指出, 局部搜索可能对于某些问题效果更好、效率更高, 但是还有一些问题只有通过全局搜索才能达到高覆盖率. 局部搜索的高性能和采取全局搜索的必要性, 导致了一种折衷策略的产生, 即将全局和局部搜索算法结合起来进行测试用例生成. 由此, 我们提出了一种结合了全局和局部搜索算法的测试用例生成方法 BB-HC. 其中, 分支限界作为全局搜索方法构建搜索框架, 而爬山法则为每个变量进行局部搜索, 两者通过高效的变量排序机制进行连接. 为了便于搜索算法的实施, 搜索的过程使用状态空间搜索进行建模.

### 3.2 分支限界搜索框架

测试用例生成在本文中就是状态空间搜索的过程. 状态空间搜索在 [12] 中进行了详细介绍. 在状态空间中, 我们从初始状态出发, 在每个当前状态下, 通过智能化的方法找到可能的搜索方向, 并找到到达最终状态的解路径. 本文算法的主要思想是扩展当前部分解. 在每个阶段, 从 FV 中选择一个变量赋值来扩展当前解. 爬山法用于判断这样的扩展是否会令搜索向潜在解前进, 并以此对不包含潜在解的分支进行剪枝. 本文中一些算法的描述如表 1 所示. 搜索过程由算法 1 表示.

**表 1 本文中的一些算法及其描述**

**Table 1** Some methods and their description used in this paper

| Name | Description |
|---|---|
| BB-HC | The global search algorithm |
| Dynamic variable ordering (DVO) | To order FV and calculate the next variable to be instantiated |
| Path tendency calculation (PTC) | To calculate the path tendency for all the variables |
| Initial domain calculation (IDC) | To calculate the domain to select the initial value for each variable |
| Hill climbing (HC) | The local search algorithm for each variable |

**算法 1** BFS-BB

**Input:** $p$: the path to be covered.

**Output:** result{Variable↦Value}: the test case that makes $p$ feasible.

1: result←null.
2: call Algorithm 2. Dynamic variable ordering.
3: call Algorithm 3. Path tendency calculation.
4: call Algorithm 4. Initial domain calculation.
5: $V_1$←select($D_1$).
6: initial state←(null, $x_1$, $D_1$, $V_1$, active).
7: $S_{\text{cur}}$←initial state.
8: **while** $x_i \neq$null **do**
9: call Algorithm 5. Hill climbing.
10: **if** $S_{\text{cur}}$=(Pre, $x_i$, $D_i$, $V_i$, inactive) **then**
11: Pre←$S_{\text{cur}}$.
12: $S_{\text{cur}}$←(Pre, $x_i$, $D_i$, $V_i$, active).
13: PV←PV−$\{x_i\}$.
14: **else**
15: result←result$\bigcup\{x_i \mapsto V_i\}$.
16: FV ← FV − $\{x_i\}$.
17: PV ← PV + $\{x_i\}$.
18: call Algorithm 2. Dynamic variable ordering.
19: call Algorithm 4. Initial domain calculation.
20: $V_i$←select($D_i$).
21: $S_{\text{cur}}$←(Pre, $x_i$, $D_i$, $V_i$, active).
22: **end if**
23: **end while**
24: final state ←$S_{\text{cur}}$.
25: **return** result.

待覆盖路径作为 BB-HC 的输入, 包含着要满足的约束. FV 中所有变量通过 DVO 进行排序 (3.3 小节), 并返回第一个待赋值变量 $x_1$. 然后 PTC(4.1 小节) 计算每个变量的路径性质, IDC(4.1 小节) 根据 PTC 的结果对 $D_1$ 进行削减, 在削减后的区间内为 $x_1$ 选择初始值. 以上这些元素构成了初始状态 (null, $x_1$, $D_1$, $V_1$, active), 它也是当前状态 $S_{\text{cur}}$. 然后爬山法 (4.2 小节) 开始对于 $x_1$ 的局部搜索过程. 为了简便起见, 下一段对于爬山法的描述指代对于 FV 中每个变量 $x_i$ 的爬山过程.

爬山法调用区间运算来判断 $x_i$ 的赋值 $V_i$ 是否产生矛盾. 如果无矛盾, 则到达山峰, 目标函数返回 0, $S_{\text{cur}}$ 的类型转变为 extensive, 这意味着对应 $x_i$ 的爬山过程成功, DVO 开始计算下一个待赋值变量. 如果爬山过程中区间运算矛盾, 则计算对应的目标函数, 并依据其返回值对 $x_i$ 的区间 $D_i$ 进行削

减. 在削减后的 $D_i$ 重新选择了一个 $V_i$ 后, 继续调用区间运算判断是否产生矛盾. 归纳起来, 对应于每个变量的爬山法结束的时候有两种可能性. 第一种是发现了对应于 $x_i$ 的山峰, $S_{\text{cur}}$ 的类型转变成 extensive, 意味着爬山成功, DVO 开始计算下一个待赋值变量. 另一种是没有发现对应于 $x_i$ 的局部最优解, 而且再无搜索空间, $S_{\text{cur}}$ 的类型转变成 inactive, 意味着对应 $x_i$ 的爬山过程失败, 回溯不可避免. 当对应于所有变量的爬山过程都成功结束, 再无变量需要排序的时候, BB-HC 的搜索过程结束, 测试用例生成成功.

### 3.3 自由变量动态排序算法

分支限界法将活结点表组织成一个优先队列, 按优先队列中规定的结点优先级选取优先级最高的下一个结点成为当前扩展结点. 本文提供了动态的决策机制确定变量的排序, 这个排序就是变量赋值顺序的优先级. 在很多搜索算法中, 确定这个顺序的原则都是它们的区间大小, 因为这样可以令整个搜索树的节点数目最小化. 但是当不同的变量具有相同的区间大小时, 利用区间大小排序的决策机制就失效了, 我们需要有一种补充策略来解决这种无法打破的僵局.

分支限界调用的区间运算部分从程序入口开始对待覆盖路径进行数据流分析, 如果变量的取值在某个分支节点处导致不可达路径的产生, 则赋值失败, 再向下进行数据流分析. 所以距离程序入口越近的表达式对于区间运算成功与否的影响越大. 因此需要结合表达式出现的顺序对程序中的变量进行分级. 因为变量是存在于控制流图的分支节点上, 在此我们给出定义 1.

**定义 1** 假如一条路径上有 $k$ 个分支 $(n_{qa}, n_{qa+1})(a\in[1,k])$, 则一个分支的级别 $\text{rank}(n_{qa}, n_{qa+1})$ 标志着它在整条路径中的顺序.

第一个分支的级别是 1, 第二个分支的级别是 2, 依次类推. 而出现在分支上的变量拥有和分支相同的级别. 一个分支上可能有多个变量, 即多个变量可能有相同的级别. 而一个变量也可能出现在多个分支上, 所以一个变量也可能拥有多个级别. 对于没有出现在某个分支上的变量, 我们认为它的级别是 infinity. 因此, 如果在分支 $(n_{qa}, n_{qa+1})$ 上比较两个变量的级别, 其中一个变量出现于此分支上而另一个变量没有, 那么前者优先级更高. 这是因为前者拥有级别 $a$ 而后者的级别是 infinity. 比较 $a$ 和 infinity 决定了排序结果. 算法 2 描述了这个过程.

这是一个双关键字 (区间大小 + 变量级别) 排序. 通过快速排序得到根据区间大小的排序结果, 对于相同区间大小的变量会从入口开始沿着整个路径对根据变量级别进行排序, 一旦得出结果则随时退出, 因为只要得到序列的第一个元素即可.

## 4 基于爬山法的局部搜索

### 4.1 启发式选取初始值

变量初始值的选取对于搜索算法能否成功至关重要. 在目前常用的搜索算法中, 初始值主要通过以下两种方式选取: 动态法多随机选取初始值, 这样可以多次执行算法返回不同的测试用例, 保证了测试用例的多样性, 但是这种随机选取的方法没有任何启发式规则的指引, 经常会导致大量的迭代; 很多二分搜索法选取中值作为初始值, 这样会导致有时候多次执行算法返回同一组用例, 无法保证测试用例的多样性. 而我们选取初始值还要考虑以下几个条件:

(1) 要保证测试用例的多样性, 即不能多次执行算法返回一组相同的用例;

(2) 根据统计, C 程序中出现的线性约束占了绝大多数 (大概 90%);

**算法 2** Dynamic variable ordering

**Input:** FV: the set of future variables, $D_i$: the domain of $x_i(x_i \in \text{FV})$, $(n_{qa}, n_{qa+1})(a \in [1,k])$: $k$ branches along the path.
**Output:** $x_i$: the next variable to be instantiated.

```
1: Q_i←quicksort(FV,|D_i|).
2: for i→1:|Q_i| do
3:    if |D_i|≠|D_j| (j>i;x_i,x_j∈Q_i) then
4:       break
5:    else
6:       for (n_qa, n_qa+1)(a∈[1,k]) do
7:          if rank(n_qa, n_qa+1)(x_i)=rank(n_qa, n_qa+1)(x_j) then
8:             a++
9:          else
10:            permutate x_i,x_j by rank(n_qa, n_qa+1).
11:            break
12:         end if
13:      end for
14:   end if
15: end for
16: x_i←head(Q_i).
17: return x_i.
```

(3) 二分法对于顺序存储的数据结构 (区间抽象域在 CTS 内是顺序存储的) 是一种很有效的查找方法.

为了兼顾测试用例的多样性和算法的效率, 在使用启发式规则判定初始值选取范围后随机选取初始值. 通过符号分析和区间运算技术, 提取和每个变量相关的表达式, 并对变量在这些表达式中的性质进行分析. 路径上的变量性质是对于每个表达式上变量性质进行统计计算的结果. 确定了路径上的变量性质之后, 可以对每个变量的初始区间进行削减, 在削减后的区间内为其选取初始值.

基于以上分析, 我们将这两种选取初始值的方式结合起来, 即在启发式确定初始值的范围后, 对初始区间进行二分削减, 在削减后的区间内进行随机选取. 为此我们给出定义 2 和定义 3.

**定义 2** 令 $B$ 为布尔值的集合 {true, false}, $D_i$ 是变量 $x_i$ 的取值区间, 分支函数 $Br(n_{qa}, n_{qa+1})(x_i)$: $D_i \to B$ ($n_{qa}$ 是一个分支节点) 可以定义为如下公式:

$$Br(n_{qa}, n_{qa+1})(x_i) : D_i \to B = (a_i x_i + \sum a_j x_j) \Re \quad c, \quad (j \neq i), \tag{1}$$

其中 $\Re$ 是一个关系运算符, $a_i$, $a_j$ 和 $c$ 是常数, $\sum a_j x_j (j \neq i)$ 是除了 $x_i$ 之外其他变量的线性组合, 我们认为它对于 $x_i$ 是一个常数. 在此基础上, 我们可以对变量初始值的选取提供一个重要依据, 即分支函数和当前变量之间的单调性关系. 函数的单调性是对于某个区间而言的, 它是一个局部概念, 它描述了函数的输出与输入的变化之间的关系. 具体来说就是, 它给出了这样的信息: 输出是与输入的变化同方向还是相反方向. 在将分支函数分解为基本函数后, 如果每个基本函数的单调性可知, 那么根据单调函数的复合仍是单调函数, 则作为合成函数的分支函数的单调性也可以得到, 从而我们可以知道如果令分支函数取 true(单调增加), 输入变量应该怎样取.

**定义 3** 变量路径性质 Path Tendency∈{positive, negative}是变量在一条路径上的性质, 它有利于路径上所有条件的满足, 它对于变量初始区间的选取提供启发信息. 性质为 positive 意味着应选取一个较大的初始值, 性质为 negative 意味着应该选取一个较小的初始值.

计算变量路径性质需要计算变量 $x_i$ 在每个分支 $(n_{qa}, n_{qa+1})(a \in [1,k])$ 的权值 $w_i(n_{qa}, n_{qa+1})$ 以及

**算法 3** Path tendency calculation

**Input:** $X$:the set of variables along the path, $pw_i$: the path weight of $x_i(x_i \in X)$.

**Output:** Path − Tendency ⟨Variable, PathTendency⟩: the table used to store the path tendency of each variable in $X$.

```
Path − Tendency←null.
for x_i∈X do
    if pw_i>0 then
        Path − Tendency←Path − Tendency⋃{⟨x_i,positive⟩}.
    else
        if pw_i<0 then
            Path − Tendency←Path − Tendency⋃{⟨x_i,negative⟩}.
        end if
    end if
end for
return Path − Tendency.
```

**算法 4** Initial domain calculation

**Input:** $D_i$=[min,max]: the input domain of $x_i$, Path − Tendency ⟨Variable, PathTendency⟩: the table used to store the path tendency of each variable in $X$.

**Output:** $D_i$: the domain used to select the initial value for $x_i$.

```
PathTendency(x_i)←retrieve(Path − Tendency).
if PathTendency(x_i)=positive then
    D_i←[(min+max)/2,max].
else
    if PathTendency(x_i)=negative then
        D_i←[min,(min+max)/2].
    end if
end if
return D_i.
```

它的路径权值 $pw_i$, 如公式 (2) 和 (3) 所示. 在得到路径权值后, 就可以通过算法 3 和 4 分别计算路径变量性质和变量初始区间.

$$w_i(n_{qa}, n_{qa+1}) = \begin{cases} \dfrac{|\ a_i\ |}{|\ a_i\ | + \sum |\ a_j\ |}(j \neq i), \text{ if } Br(n_{qa},\ n_{qa+1}) \text{ is monotonically increasing,} \\ -\dfrac{|\ a_i\ |}{|\ a_i\ | + \sum |\ a_j\ |}(j \neq i), \text{ if } Br(n_{qa},\ n_{qa+1}) \text{ is monotonically decreasing.} \end{cases} \tag{2}$$

$$pw_i = \sum_{a=1}^{k} w_i(n_{qa}, n_{qa+1}). \tag{3}$$

### 4.2 爬山过程

爬山法的应用主要是为了解决等式的问题 (约束较强), 同时我们也将其扩展到不等式, 即爬山法可以解决目前出现的常见关系表达式. 爬山法调用区间运算判断当前变量 $x_i$ 的值 $V_i$ 是否会产生矛盾. 换言之, 能够令区间运算不矛盾的 $x_i$ 的一个值 $V_i$ 就是我们要寻找的山峰. 为了更好的说明爬山法的工作原理, 我们对于路径上的分支条件进行了分解, 如图 2 所示.

若路径上有 $k$ 个分支节点, 那么所对应的 $k$ 个分支函数都为 true 才能保证路径可达; 否则少于 $k$ 个分支条件为 true, 那么分支函数取值为 false 的分支需要被定位出来并对于矛盾的情况进行分析,

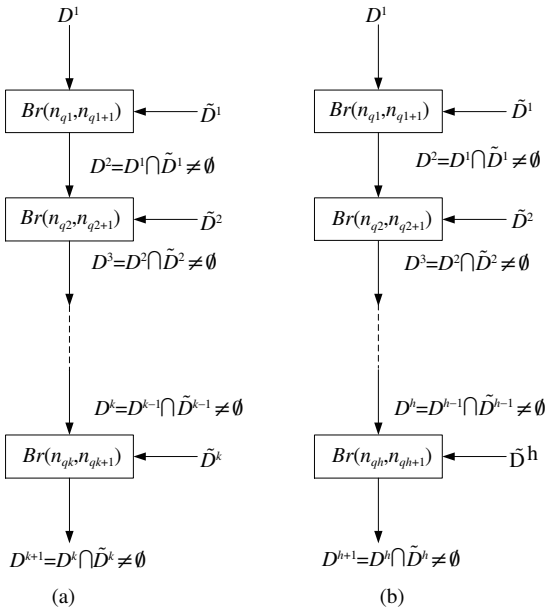



**图 2 区间运算判断矛盾过程**

**Figure 2** The conflict detecting process by interval arithmetic. (a) No conflict; (b) conflict detected

进而对当前变量 $x_i$ 的区间 $D_i$ 进行削减. 对于分支条件 $Br(n_{qa}, n_{qa+1})(a\in[1,k])$ 是否能够为 true 的判断, 取决于两个区间的取值, 即: 进入第 $a$ 个分支的变量区间 $D^a$(满足前面的 $a-1$ 个分支条件) 和满足第 $a$ 个分支条件的变量区间 $\widetilde{D}^a$, 后者是将前者代入 $Br(n_{qa},n_{qa+1})$ 计算的结果. 如果 $D^a\bigcap\widetilde{D}^a\neq\emptyset$, 则可以确定 $D^a\bigcap\widetilde{D}^a$ 既满足前面的 $a-1$ 个分支条件也满足第 $a$ 个分支条件. 则区间运算可以向下继续判断剩下的分支条件. 在这个过程中, 如果 $Br(n_{q1},n_{q1+1})$=false$(1\leqslant h\leqslant k)$, 则在分支 $Br(n_{qh},n_{qh+1})$ 上检测出了矛盾, 需要记录矛盾信息, 计算目标函数值, 并依据这个目标函数值对 $D_i$ 进行削减.

图 2 中, (a) 为一次成功的赋值, $k$ 个分支条件都得到了满足, 在它完成之后就可以进行剩余变量的排序了; (b) 是一次失败的赋值, 第 $h(1\leqslant h\leqslant k)$ 个分支条件没有得到满足, 后续的算法就需要对矛盾的条件进行判断. 具体过程如算法 5 所示.

目标函数的确定对于一个搜索算法的效率和搜索能够成功与否至关重要. 在某种意义上, 一个解比其他候选解更优, 那它就应该具有更好的返回值, 反之, 如果一个解比其他解更差, 则它的返回值也较差. 为此, 依据区间运算的特点和约束求解的需要, 我们给出公式 (4) 作为目标函数的定义.

$$F(V_i)=V_i-\sum_{a=1}^{k}(D^a\bigcap\widetilde{D}^a). \tag{4}$$

$F(V_i)$=0 说明区间运算没有矛盾, 是对应于 $x_i$ 的山峰; 否则就需要根据 $F(V_i)$ 的返回值对 $D_i$ 进行削减, 从削减后的 $D_i$ 中选取新的 $V_i$. 在这个爬山过程中, $F(V_i)$ 的绝对值越来越接近 0, 它就是对应的山峰. 也就是说, 我们的搜索是一个寻找最小值的过程. 其返回值的绝对值越接近 0 的候选解就越优, 我们优先选择这样的候选解作为下一步的赋值. 当区间运算失败时, $F(V_i)$ 的返回值提供了需要削减 $D_i$ 的上界和下界, 这分别由 $F(V_i)$ 的正负号和绝对值决定. 因为对于 $D_i$ 的削减是从两个方向进行的, 即目标函数是双向收敛的, 所以算法的效率得到了很大的提高. 并且通过这种方式, 由分支条件构成的路径约束以越来越精确的方式进行传播.

**算法 5** Hill climbing

**Input:** $D_i$: the current domain of $x_i$, $V_i$: the current value of $x_i$.

**Output:** $S_{\text{cur}}$=(Pre, $x_i$, $D_i$, $V_i$, inactive)when hill climbing fails; $S_{\text{cur}}$=(Pre, $x_i$, $D_i$, $V_a$, active)when hill climbing succeeds.

```
1: while |D_i|>1 do
2:    for i→1:k do
3:       Br(n_qa,n_qa+1)←false;
4:       D̃^a←Br(n_qa,n_qa+1)with D^a.
5:       if D^a⋂D̃^a≠∅ then
6:          Br(n_qa,n_qa+1)←true;
7:          D^a+1←D^a⋂D̃^a.
8:       else
9:             calculate F(V_i)
10:            break
11:      end if
12:   end for
13:   if F(V_i) = 0 then
14:      S_cur←(Pre, x_i, D_i, V_i, extensive).
15:      return S_cur.
16:   else
17:      if F(V_i) < 0 then
18:         D_i←[V_i+1,V_i+F(V_i)].
19:      else
20:         D_i←[V_i−F(V_i),V_i−1].
21:      end if
22:   end if
23:   V_i←select(D_i).
24: end while
25: S_cur←(Pre, x_i, D_i, V_i, inactive).
26: return S_cur.
```

## 5 实例分析

为了更好的说明 BB-HC 的效果, 我们使用图 1 中的例子来说明其执行过程. 待覆盖路径 Path5 是加粗显示的. 我们选择这条路径是因为这条路径上的约束对于两个变量是非常严格的, 其本质相当于解方程组. 可以看出对应的约束满足问题只有 $\{x_1\mapsto 60, x_2\mapsto 40\}$一个解. 为了简便起见, 将两个变量的输入区间都设为 [1,100], 长度是 100. PTC 按照表 2 所示的方式计算每个变量的路径性质. DVO 计算出下一个待赋值变量为 $x_1$, 如表 3 所示. 因此需要从 [1,100] 中为 $x_1$ 选择一个赋值. IDC 检索变量路径性质表得到 $x_1$ 的路径性质为 positive, 这代表着较大的值较有利于路径约束的满足. 假如选择了 70, 就有了图 3 的判断过程. 可以看出 70 导致区间运算失败, 所以 70 并不是 $x_1$ 对应的山峰.

矛盾发生在第四个分支, 通过计算目标函数 $F(70)=20>0$, 可以把 $x_1$ 的区间削减至 [70−20(50), 70−1(69)], 这个区间要远小于 [1,100]. 所以尽管 70 不是最终解的一部分, 但它为下一步的搜索提供了方向. 假如从 [50,69] 中选择了 55, 则通过计算目标函数 $F(55)=-10<0$, 可以把 $x_1$ 的区间削减至 [55+1(56),55+10(65)], 这个区间要远小于 [50,69]. 所以尽管 55 不是最终解的一部分, 但它为下一步的搜索提供了方向. 假如从 [56,65] 中选择了 60, $F(60)=0$ 代表 60 是对应 $x_1$ 的山峰, 其起始点是 70. 由于此时已经计算出仅剩一个值 40 可以给 $x_2$ 选择, 意味着成功解出 $\{x_1\mapsto 60, x_2\mapsto 40\}$.

**表 2 $x_1$ 和 $x_2$ 路径性质计算过程**

**Table 2** PTC process for $x_1$ and $x_2$

| Branching condition | Monotonicity | Weight | Path weight | Path tendency |
|---|---|---|---|---|
| $x_1 - x_2 > 0$ | Br($x_1$):increasing | $w_1 = 0.5$ | | |
| | Br($x_2$):decreasing | $w_2 = -0.5$ | $pw_1 = 1.5$ | $\{\langle x_1$, positive$\rangle$ |
| $x_1 \geqslant 0$ | Br($x_1$):increasing | $w_1 = 1$ | $pw_2 = -0.5$ | $\langle x_2$, negative$\rangle\}$ |
| $x_1 + x_2 = 100$ | – | – | | |
| $x_1 - x_2 = 20$ | – | – | | |

**表 3 $x_1$ 和 $x_2$ 排序过程**

**Table 3** DVO process for $x_1$ and $x_2$

| Ordering rule | Condition of each variable | Tie encountered? | Ordering result |
|---|---|---|---|
| Domain size | $\|D1\|$=69, $\|D2\|$=69 | yes | $x_1 \rightarrow x_2$ |
| Rank1 | Rank1($x_1$)=1, Rank1($x_2$)=1 | yes | |
| Rank2 | Rank1($x_1$)=2, Rank1($x_2$)=$\infty$ | no | |

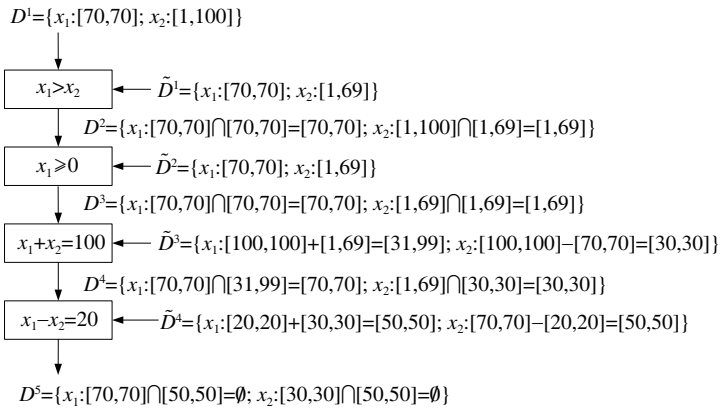



**图 3 $x_1$ 赋值 70 区间运算判断过程**

**Figure 3** The calculating process of 70 for $x_1$

## 6 算法评估与实验

### 6.1 爬山法性能分析

在这一部分中, 我们通过使用变量路径性质确定初始值的爬山法与变量个数和表达式个数之间的关系来实验来分别验证初始值选取策略和爬山过程的效果. 由此, 我们进行了三种方法的对比: 随机初始值和矛盾后不爬山 RI&NHC(random initial value and no hill climbing)、随机初始值和矛盾后爬山 RI&HC(random initial value and hill climbing)、通过使用变量路径性质确定初始值和矛盾后爬山 BB-HC. 因为这几种情况测试用例生成时间差别较大, 我们对代表测试用例生成时间的坐标轴进行了指数化处理. 最后我们通过一个实际工程中的例子对这三种方法的性能进行对比.

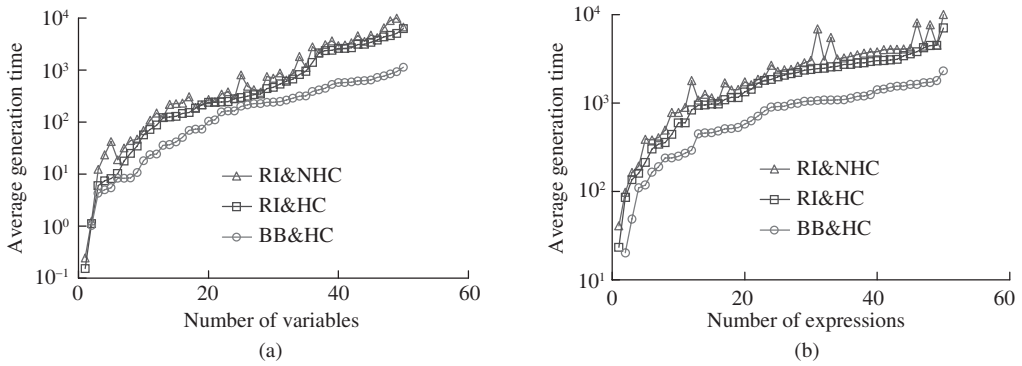



**图 4 BB-HC 性能测试结果**

**Figure 4** Performance test result of BB-HC. (a) Relationship between test case generation methods and the number of variables; (b) relationship between test case generation methods and the number of expressions

### 6.1.1 与变量个数的关系

这一部分通过测试变量个数与三种测试用例生成方法的关系来进行三种方法的对比. 为此, 进行了如下的实验设置: 将被测程序设置为 50 个输入变量 $x_1$,$x_2$,...,$x_n$, 其中 $n$ 从 1 顺次递增到 50. 采用语句覆盖, 在每个被测程序中设置 50 个 if 语句 (相当于 50 个路径约束), 而且只有一条待覆盖路径, 即完全由真分支 (TTTTT) 构成的路径, 从而分支条件与分支谓词完全一致. 每个 if 语句都是 $n$ 个变量的最紧密的线性组合. 并且为了考察三种方法处理等式的能力, 在每个被测程序中要保证有至少一个等式. 对于 $n$ 从 1 到 50 之间的每个值所对应的每个被测程序都使用三种方法进行了 50 次实验, 对每次实验的测试用例生成时间都进行了记录, 并取平均值进行比较. 结果如图 4(a) 所示.

可以看出 BB-HC 的平均生成时间要远小于另两种方法. 而初始值和矛盾后赋值都随机的方法 RI& NHC 用掉了最多的时间. 而且 BB-HC 的平均生成时间随着变量个数的增加呈匀变速增长. 对拟合曲线求导可以得出 $y$=1.06$x$−8.682, 据此可以大致推断当 $n$ 小于 8 时, 测试用例生成时间大致相当; 而当 $n$ 大于 8 时, 测试用例生成时间开始增长.

### 6.1.2 与表达式个数的关系

这一部分通过测试表达式个数与三种测试用例生成方法的关系来进行三种方法的对比. 为此, 进行了如下的实验设置: 将被测程序设置为 50 个输入变量 $x_1$,$x_2$,...,$x_{50}$, 采用语句覆盖, 在每个被测程序中设置 $u$($u$∈[1,50]) 个 if 语句 (相当于 $u$ 个路径约束), 而且只有一条待覆盖路径, 即: 完全由真分支构成的路径, 从而分支条件与分支谓词完全一致. 每个 if 语句都是 $n$ 个变量的最紧密的线性组合. 并且为了考察三种方法处理等式的能力, 在每个被测程序中要保证有至少一个等式. 对于 $n$ 从 1 到 50 之间的每个值所对应的每个被测程序都使用三种方法进行了 50 次实验, 对每次实验的测试用例生成时间都进行了记录, 并取平均值进行比较. 结果如图 4(b) 所示.

可以看出 BB-HC 的平均生成时间要远小于另两种方法. 而初始值和矛盾后赋值都随机的方法 RI& NHC 用掉了最多的时间. 对曲线进行拟合可以得出, 使用 BB-HC 时, 测试用例生成时间和表达式个数呈线性关系. 即当表达式个数增加的时候, BB-HC 的平均生成时间也会随之线性增长.

### 6.1.3 测试实际工程

我们也对来自 Florida State University 的 C 程序库中工程 fem1d_adaptive.c 的一个函数 solvey 使

表 4 测试工程 fem1d_adaptive 结果

**Table 4** The result of testing fem1d_adaptive

| Strategy | Coverage criterion | Time (ms) | Coverage (%) |
|---|---|---|---|
| RI&NHC | Statement | 549 | 96 |
| | Branch | 389 | 100 |
| RI&HC | Statement | 524 | 100 |
| | Branch | 440 | 97 |
| BB-HC | Statement | 445 | 100 |
| | Branch | 378 | 100 |

表 5 BB-HC 与文献 [13] 方法对一些实际工程中的程序测试结果比较

**Table 5** The details of comparison with method in [13] on programs from real projects

| Program | Function | Criterion | Paths | Coverage by [13] (%) | Coverage by BB-HC (%) |
|---|---|---|---|---|---|
| atanl.c | atan2l() | Statement | 4 | 75 | 100 |
| | | Branch | 4 | 72 | 100 |
| | | MCDC | 9 | 71 | 100 |
| sinl.c | sinl() | Statement | 3 | 92 | 92 |
| | | Branch | 5 | 92 | 97 |
| | | MCDC | 5 | 83 | 87 |
| | cosl() | Statement | 3 | 87 | 87 |
| | | Branch | 6 | 91 | 97 |
| | | MCDC | 5 | 82 | 83 |

用以上三种方法进行了测试, 其结果如表 4 所示. 可以看出 BB-HC 不论是在测试用例生成时间还是在覆盖率都明显优于相对比的另两种方法, 并且可以看出初始值选取策略和爬山法的处理分别都取得较理想的效果.

### 6.2 覆盖率分析

#### 6.2.1 与静态方法对比

这一部分中, 对 BB-HC 与文献 [13] 中所使用的方法进行对比. 文献 [13] 所使用的方法对于变量之间弱关联的被测程序效果较好, 但是对于变量之间具有较强关联的情况处理能力稍差. 我们对来自工程 de118i-2 中的几个函数进行了测试, 所选择的的函数都包含至少一个函数调用, 并且变量之间关联较强. 实验使用三种覆盖准则: 语句、分支、MCDC. 对每个被测函数, 我们都进行了 100 次实验, 取平均覆盖率用作比较. 比较结果如表 5 所示.

可以看出 BB-HC 测试实际工程中的被测程序时, 表现要优于文献 [13] 中所使用的方法. BB-HC 更加优异的表现主要是由于前面介绍的搜索策略的应用. 当然 BB-HC 也对一些函数未达到 100% 覆盖率, 这也是我们下一步要解决的问题之一.

#### 6.2.2 与动态方法对比

这一部分将 BB-HC 和两种动态方法进行对比. 被测程序来自文献 [14~16]. 使用分支覆盖, 在

**表 6 BB-HC 与 GA 和 SA 对比实验结果**

**Table 6** The details of comparing BB-HC with GA and SA

| Program | LOC | Paths | AC by GA (%) | AC by SA (%) | AC by BB-HC (%) |
|---|---|---|---|---|---|
| triangleType | 31 | 6 | 95 | 99.88 | 100 |
| isValidDate | 59 | 5 | 99.95 | 98.21 | 100 |
| calDay | 72 | 20 | 96.31 | 99.97 | 100 |
| cal | 53 | 7 | 99.02 | 99.27 | 100 |

4 个被程序上将 BB-HC 与遗传算法 (GA) 和模拟退火 (SA) 进行对比. 其中 GA 和 SA 的采用典型常用的参数设置. 对比实验共进行了 100 次, 取平均覆盖率用作对比, 结果如表 6 所示.

可以看出 BB-HC 对 4 个被测程序都达到了 100% 覆盖并超过了进行对比的两个算法, 这 4 个程序对于 BB-HC 来说相对比较简单. BB-HC 更优的表现主要有两个原因. 第一个原因是 BB-HC 的初始值并非随机选取, 而是基于启发式规则, 所以 BB-HC 在算法执行开始起就会以较快的速度达到较高的覆盖率. 第二个原因是于分支限界是一种确定性算法, 本身没有多次迭代的需要, 所以效率也会相应较高.

## 7 结束语

作为保障软件质量的重要手段, 软件测试在软件开发周期中所占比重很大, 对于软件测试自动化的需求也非常急迫. 而作为软件测试里的一个基本问题, 面向路径的测试用例自动生成则尤为重要. 本文将面向路径的测试用例生成问题定义为约束满足问题, 并通过基于分支限界搜索框架的算法 BB-HC 进行求解. 分支限界作为全局搜索算法具有灵活的回溯机制, 而爬山法作为局部搜索算法为为每个变量寻找满足约束的赋值, 二者通过基于变量级别的动态排序算法进行高效连接. 为了便于搜索算法的实施, 使用状态空间搜索对搜索过程进行建模. 爬山法从启发式选取的初始值开始, 调用区间运算对每次的赋值进行判断, 并依据目标函数的返回值引导后续的搜索过程. 实验结果显示, 初始值选取策略和爬山的过程对于搜索效率的提升效果明显, 而且本文方法和比较流行的动、静态方法相比, 可以取得更高的覆盖率, 这也为其在工程中的应用打下了很好的基础. 我们未来的研究工作主要是处理更大规模的程序和达到更高的覆盖率, 对于文中没有达到 100% 的被测程序我们将完成对其分析和算法改进的过程.

## 参考文献


1 Beizer B. Software Testing Techniques. 2nd ed. New York: Van Nostrand Reinhold, 1990

2 King S, Hammond J, Chapman R, et al. Is proof more cost effective than testing?. IEEE Trans Softw Eng, 2000, 26: 675–686

3 Yan J. Automated software testing based on constraint processing. Dissertation for Ph.D. Degree. Beijing: Institute of software, Chinese Academy of Sciences, 2007 [严俊. 基于约束求解的自动化软件测试研究. 博士学位论文. 北京: 中国科学院软件研究所, 2007]

4 Shan J H. Study on path-wise automatic generation of test data. Dissertation for Ph.D. Degree. Changsha: National University of Defense Technology, 2002 [单锦辉. 面向路径的测试数据自动生成方法研究. 博士学位论文. 长沙: 国防科学技术大学, 2002]

5 DeMilli R A, Offutt A J. Constraint-based automatic test data generation. IEEE Trans Softw Eng, 1991, 17: 900–910

6 Gallagher M J, Narasimhan V L. Adtest: a test data generation suite for ada software systems. IEEE Trans Softw Eng, 1997, 23: 473–484
7 Gupta R, Mathur A P, Soffia M L. UNA based iterative test data generation and its evaluation. In: Proceedings of the IEEE International Conference on Automated Software Engineering, New York, 1999. 224–232
8 Robschink T, Snelting G. Efficient path conditions in dependence graphs. In: Proceedings of the 24th International Conference on Software Engineering, New York, 2002. 478–488
9 Cadar C, Dunbar D, Engler D R. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proceedings of OSDI, 2008. 209–224
10 Tang R, Wang Y W, Gong Y Z. Research on abstract memory model for test case generation. In: Proceedings of the 7th China Test Conference, Hangzhou, 2012. 144–149 [唐容, 王雅文, 宫云战. 面向测试用例生成的抽象内存模型研究. 第七届中国测试学术会议. 杭州, 2012. 144–149]
11 Harman M, McMinn P. A theoretical and empirical study of search-based testing: local, global, and hybrid search. IEEE Trans Softw Eng, 2010, 36: 226–247
12 Xing Y, Gong Y Z, Wang Y W, et al. Path-wise test data generation based on heuristic look-ahead methods. Math Probl Eng, 2014
13 Wang Y W, Gong Y Z, Xiao Q. A method of test case generation based on necessary interval set. J Comput Aid Des Comput Graph, 2013, 25: 550–556 [王雅文, 宫云战, 肖庆. 基于区间必然集的测试用例生成方法. 计算机辅助设计与图形学学报, 2013, 25: 550–556]
14 Mao C Y, Yu X X, Chen J F. Generating test case for structural testing based on ant colony optimization. In: Proceedings of the 12th International Conference on Quality Software, Xi'an, 2012. 98–101
15 Alba E, Chicano F. Observation in using parallel and sequential evolutionary algorithms for automatic software testing. Comput Oper Res, 2008, 35: 3161–3183
16 Ammann P, Offutt J. Introduction to Software Testing. New York: Cambridge University Press, 2008

# Branch and bound framework for automatic test case generation

XING Ying[1,2*], GONG YunZhan[1], WANG YaWen[1,3] & ZHANG XuZhou[1]

1 *State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing* 100876*, China*;
2 *School of Electronic and Information Engineering, Liaoning Technical University, Huludao* 125105*, China*;
3 *Institute of Computing Technology, Chinese Academy of Science, Beijing* 100190*, China*
*E-mail: lovelyjamie@yeah.net

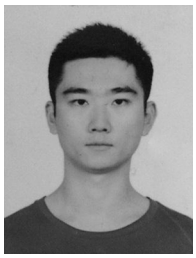
**Abstract** As a basic problem and a hotspot in software testing, the automation of path-wise test case generation is of special importance, which is in essence a constraint satisfaction problem solved by search strategies. Therefore, the constraint processing efficiency of the selected search algorithm is a key factor. Aiming at the increase of search efficiency, a hybrid intelligent algorithm is proposed to efficiently search the solution space of potential test cases by making full use of both global and local search methods. Branch and bound is adopted for global search, and hill climbing is adopted for local search. They are highly integrated to form an efficient search framework.

**Keywords** path-wise, test case generation, constraint satisfaction problem, branch and bound, hill climbing

**XING Ying** was born in 1978. She received the Ph.D. degree from Beijing University of Posts and Telecommunications in 2014. Currently, she is a postdoctor in Beijing University of Posts and Telecommunications. Her research interests include software testing and static analysis.

**WANG YaWen** was born in 1983. She received the Ph.D. degree from Beijing University of Posts and Telecommunications in 2010. Currently, she is a lecturer in the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. Her research interests include static analysis and automatic software testing.

**GONG YunZhan** was born in 1962. He received the Ph.D. degree from Institute of Computing Technology, Chinese Academy of Sciences in 1991. Currently, he is a professor and doctoral student supervisor in the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests include fault tolerant computing and software testing. Dr. Gong is associate director of CCF TCFTC, as well as a member of the standing committee of CCF TCSC.



**ZHANG XuZhou** was born in 1987. He received the B.S. degree from HuaZhong University of Science and Technology in 2010. Currently, he is currently a Ph.D. candidate in the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interest is software testing.