文章编号：0253-374X(2013)01-0135-07 DOI:10.3969/j.issn.0253-374x.2013.01.023

基路径与业务逻辑结合的有效测试路径集算法

杜庆峰，张秋华
(同济大学 软件学院，上海 201804)

摘要：在分析基路径覆盖测试技术及相关最新研究成果基础上，以被测程序及其程序图为依据，通过对变量依赖、非关联路径等的定义，推导出有效路径数量公式 V_E，进而提出了一种解决无效路径问题的算法模型. 通过对算法模型的理论推导和验证，证明该算法模型是有效的.

关键词：基路径；无效路径；算法；测试用例
中图分类号：TP311.5 **文献标志码**：A

An Effective Path Set Algorithm Based on a Combination of Basis Paths and Business Logic

DU Qingfeng, ZHANG Qiuhua
(College of Software Engineering, Tongji University, Shanghai 201804, China)

Abstract: Based on an analysis of the basis path test methods and the latest relevant research results, this paper defines variable dependence and non-affiliated path according to the tested program and its program graph. Subsequently, the calculation formula V_E of the set of effective path is devived. Furthermore, an algorithm model for solving infeasible paths is established. A theoretical derivation and an application prove the algorithm to be feasible and efficient.

Key words: basis paths; infeasible path; algorithm; test cases

软件测试按照类别一般可划分为黑盒测试(功能性测试)、白盒测试(结构性测试)和非功能性测试. 白盒测试主要用于组件(单元)级别的测试，其逻辑覆盖主要有语句覆盖、判定覆盖、条件覆盖、判定-条件覆盖、条件组合覆盖及组件基路径覆盖[1]. 其中，组件基路径覆盖测试是一种重要的路径覆盖测试技术，由于其在测试用例设计时会出现业务逻辑不合理的无效路径问题，故人工干预多、测试效率低[1-2].

由于组件基路径覆盖测试的思想是以图论及向量空间等为依据，得出的独立路径集合中的路径没有考虑其业务逻辑的合理存在，因此在大多数情况下会出现部分路径从图论和几何学的角度去分析是存在的，但这些路径所对应的业务逻辑是不可行的，即路径是无效的，其结果导致这些路径对应的测试用例是不可执行的[1-2]. 这里列举一个简单的判定三角形类型程序组件[3-4](以C语言实现，与用Java等面向对象语言实现情况类似)来说明路径从图论和几何学的角度去分析是存在的，但该路径不具有业务逻辑的合理性. 图1为组件程序图.

节点划分	//程序主体
	Void Triangle (int *a*, int *b*, int *c*)
	{
	Bool isTriangle;
1	If ((*a* < *b* + *c*) && (*b* < *a* + *c*) && (*c* < *a* + *b*))
2	IsTriangle = true;
3	Else
	IsTriangle = false;
4	If (isTriangle)
	{
5	If (*a* == *b* && *b* == *c*)
6	{Print ("Equilateral\n");}
7	Else
	If (*a* ! = *b* && *b* ! = *c* && *a* ! = *c*)
8	{Print ("Ordinary triangle \n"); }
9	Else
	Print ("Isosceles\n");
10	}
11	Else
	Print ("Not a triangle\n");
12	}

图1 判定三角形类型程序

Fig.1 Triangle judgment procedure

收稿日期：2011-12-20
基金项目：国家自然科学基金(41171303)
第一作者：杜庆峰(1968—)，男，教授，博士生导师，工学博士，主要研究方向为软件测试. E-mail:du_cloud@tongji.edu.cn

图1左边部分是程序的节点划分情况,这里将顺序执行的且不会引起程序控制转移的语句整合在一个节点中,并不影响后面对问题的分析.对应的组件程序图[1]如图2所示.

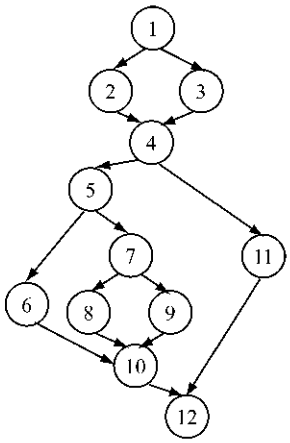

图2 判定三角形形状的组件程序图

Fig.2 Component program graph of triangle judgment procedure

基路径覆盖测试的独立路径集合不唯一,图1独立路径中包含路径:1→3→4→5→7→8→10→12,从图2可以看出这条路径是存在的,但是该路径所对应的业务逻辑是不合理的,因为节点3对应的语句是判定为非三角形的结果,所以从业务逻辑上不可能再经过5→7→8→10→12节点,故路径1→3→4→5→7→8→10→12是业务逻辑上不合理的无效路径,其对应的测试用例是不可执行的.这样的路径在这个例子中不止一条,这里仅说明问题的存在,不一一分析.

1982年,McCabe将图论、向量空间理论引入路径测试领域[2],提出了强连通的圈复杂度,即线性独立环路数量的相关概念,将被测单元组件转换为程序图,进而提出了一种基路径算法.该算法得出的独立路径集合没有考虑路径和业务逻辑的结合,也没有考虑路径中变量依赖.McCabe也意识到会出现无效路径,并给出了以无效路径为对象用人工干预的方法使其转化为另一条独立的业务逻辑合理的其他路径,但这种方法可操作性差,尤其对于比较复杂或复杂的单元组件.

1993年,Nakajo等[3]提出了组件基路径覆盖测试的路径中变量的定义-使用链(define-use chains)概念,研究了路径中变量定义和使用的关系,但没能提出一种具体的方法或模型来解决基路径中的无效路径问题.2008年,Yan等[4]提出了一种程序中变量约束限制和交互的基路径覆盖测试思想.这种方法是基于C语言的,是在McCabe方法基础上的改进,但也没能有效解决基路径中的无效路径问题.2009年,Yates等[5]提出了JJ-Paths模型,通过分析JJ-Paths及其路径序列来分析程序组件中语句和分支的合理性,绕过无效路径问题,其目的是变相达到合理的基路径覆盖,但仍有待于进一步研究并验证.2010年,Zolotov等[6]提出了一种基于路径的高速结构测试模型,该模型提出了统计路径跟踪,为基路径测试中的路径相关性度量提供了依据,如能够反映基路径中无效路径可能的占比关系等,为解决无效路径问题提供帮助,但未能对基路径集合中出现的无效路径提出根本的处理策略.2011年,笔者首次提出通过将路径业务逻辑与组件程序图结合构建优化程序图的思想,为解决无效路径问题提出了一种新思路[7].

综合以上分析,目前基路径覆盖测试技术的独立路径集合中存在无效路径问题,即路径在业务逻辑上不可行的瓶颈仍没有得到解决,这直接导致基路径覆盖测试技术使用过程中人工干预多、效率低.本文在分析基路径覆盖测试技术和逻辑覆盖测试技术的基础上,将两者结合提出了组件基路径覆盖测试有效路径集生成模型,解决了组件基路径覆盖测试中存在的不足和瓶颈.该模型生成的独立路径集合中避免了出现从图论及集合学方面理解存在的路径,但这些路径不符合业务逻辑的情况,进而解决了无效路径问题.这样得出的路径集不但满足基路径覆盖测试的原理和思想,同时路径集合中的所有路径均具有业务逻辑的合理性.

1 算法相关定义

结合被测程序的程序图和业务逻辑,为了便于本文第3节中算法的分析,在此对算法涉及到的相关概念进行了定义.

定义1 非关联路径

在被测程序对应的程序图中,如果存在一些路径且这些路径与被测程序的业务逻辑没有直接关联,则称之为非关联路径.下面以例子加以说明.

在一个给500个员工代发工资组件的单元系统中,根据一个工资单文件(fileA)中的工资金额来给系统中的用户发工资,其伪代码和节点划分及程序图[1]分别如图3和4所示.

该组件程序中的非关联路径为1→2→3→4→7,即当打开或者读取文件失败(如fileA不存在或者格

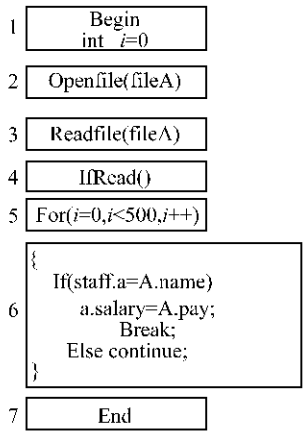

图 3　代发工资伪代码及节点划分

Fig. 3　Pseudo-code and node division of salary processing system

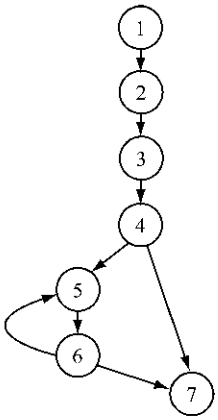

图 4　代发工资程序图

Fig. 4　Program graph of salary processing system

式错误等)的路径，该路径与发工资具体的业务逻辑没有关系.

定义 2　变量依赖

在被测程序对应的程序图中，如果存在节点 N_i 与节点 N_m，从节点 N_i 到节点 N_m 存在路径 P，且在节点 N_i 处对特定变量 V 进行定义并初始化赋值，则存在以下两种情况：

(1) 如果在节点 N_m 处引用变量 V 并且在路径 P 的其他节点处没有对 V 进行重新定义或赋值，那么节点 N_m 变量依赖于节点 N_i.

(2) 如果在节点 N_m 处对 V 进行重新赋值或计算，从而改变 V 的值，那么节点 N_m 变量也依赖于节点 N_i，且 N_m 与 N_i 关于变量 V 相互依赖.

定义 3　有效路径集

如果被测程序对应的程序图中存在一组路径集合 $P=\{P_1,P_2,\cdots,P_m\}$，当且仅当集合中的路径同时满足下面三个条件时，则称路径集合 P 是程序图的有效路径集.

(1) P 中的每一条路径 $\{P_1,P_2,\cdots,P_m\}$ 都是可执行的.

(2) P 中的每一条路径 $\{P_1,P_2,\cdots,P_m\}$ 分别代表不同的业务逻辑(线性无关性).

(3) P 中的所有路径覆盖了基本的业务逻辑.

2　有效路径集算法思想及逻辑实现

2.1　算法思想

本算法是基于组件级别的测试算法，其核心思想主要分为两部分：

第一步，对 McCabe 覆盖中产生不可执行路径的那些路径和节点进行处理，使不可执行路径不会出现在有效路径集里[8-9]. 通过定义非关联路径，将导致出现不可执行路径的节点通过变量依赖隔离出来，并在非关联路径找出之后，将有变量依赖关系且入度和出度均为 1 的节点从程序图中删除，这样在第二步中就不会出现不可执行路径. 具体算法逻辑将在下面的算法逻辑步骤 2 中作详细说明.

第二步，采用自底向上遍历程序图的方法，从汇节点[1]出发寻找有效路径. 这种从结果出发逐步向上遍历程序图的各种分支和判定结构的方法，既保证了基路径覆盖又保证了程序逻辑覆盖的完整性. 具体算法逻辑将在下面算法逻辑步骤 3 中作详细说明.

具体算法逻辑如下：

步骤 1　依据被测程序，画出程序图.

步骤 2　找出程序图中的非关联路径，将其中入度和出度均为 1 的节点从程序图中隔离并删除(源节点和汇接点除外)，保留重复的节点，得到优化后的程序图.

步骤 3　在优化后的程序图中，运用公式 $V_E=\sum_{i=0}^{m}(I_i-C)+D$ 计算有效路径集元素的个数并找出有效路径. 其中，V_E 是有效路径集的元素的个数；I_i 是从汇节点向上遍历的第一个入度不小于 2 的节点的入度；C=Count($I_{i-1}>2$)，是 i 节点的直接前置节点中入度不小于 2 的节点的个数；D 是非关联路径的个数；m 是从汇节点开始向上遍历到某个节点时遍历器的计数. 该节点同时满足以下两个条件：① 该节点是前面某一个入度不小于 2 的节点的一个直接前置节点；② 该节点的所有直接前置节点入度小于 2.

不难看出，该公式是一个递归公式，递归过程如

下：

在程序图中，从汇节点开始向上遍历找到第一个入度不小于2的节点 i，其入度记为 I_i，需要注意的是，对于存在循环的时候，忽略进入循环的那个节点.

从节点 i 开始向上遍历，设节点 i 的直接前置节点中入度不小于2的节点个数为 C，则节点 i 处得到的有效路径个数为 (I_i-C). 用同样的方法计算节点 i 的每一个直接前置节点处的有效路径个数，并不断向上遍历.

递归结束条件：通过上面的递归遍历到的某个入度大于或者等于2的节点，其所有直接前置节点的入度都小于2，则遍历结束.

遍历结束后，将得到的所有 (I_i-C) 相加，再加上非关联路径数目就可以得出总的有效路径的个数.

步骤4 依据步骤3得出的有效路径集的路径个数，在优化后的程序图中取路径. 采用自底向上的方法，从汇节点出发，找到第一个入度不小于2的节点，然后沿着它的各个直接前置节点分别取一次，对于它的直接节点如果入度也不小于2，则按照同样的方法取节点，直到取尽各种情况.

用以上计算方法得到有效路径集中的路径个数和具体路径，需要考虑程序内部各种语句结构. 各种常见程序的基本语句结构有：顺序执行的语句、条件语句(if 语句)、循环语句(for 语句、while 语句)、选择语句(switch(case)语句)等.

下面就算法中可能出现的各种语句结构[10]进行举例说明并就有效路径的计算进行分析.

(1) 对于顺序执行语句，有效路径个数 $V_E=1$，如图5所示.

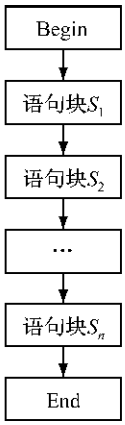

图5 顺序执行的语句

Fig.5 Sequential statement

(2) 对于包含 If 条件语句的情况，如图6所示，$V_E=4-0=4$.

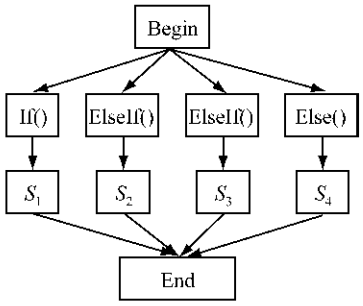

图6 条件语句

Fig.6 Conditional statement

(3) 对于包含循环语句的情况，如图7所示，$V_E=2$.

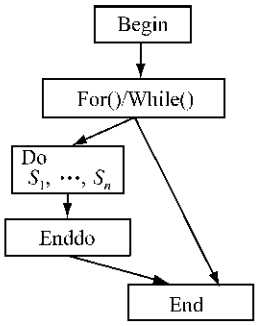

图7 循环语句

Fig.7 Loop statement

(4) 对于包含 Switch(case)语句的情况，如图8所示，$V_E=n+1$.

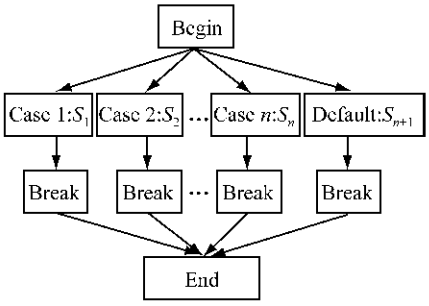

图8 Switch(case)语句

Fig.8 Switch(case) sentence

如果在程序图中出现同时包含(1)～(4)情况的复合结构，进行综合处理即可，算法同样适用.

2.2 算法实现

由于本算法是研究基于组件级别的测试算法，一般情况下单个组件程序图中划分的节点数是有限的. 对于程序图 $G=(V,E)$，其中 V 是程序图中节点的集合，E 是边(节点关系)的集合. 那么，E 可以通

过定义一个 $n\times n$ 的邻接矩阵 R(其中 n 是节点的个数),当节点 j 是节点 i 的直接后继节点时,令 $R[i][j]=1$,否则令它为 0. 由于程序图是个有向图,所以邻接矩阵中第 i 行非零元素的个数为节点 i 的出度,第 i 列非零元素的个数为节点 i 的入度. 当 n 特别大时,为了节省存储空间,可以通过邻接表的方式来实现,即:定义 n 个长度为 n 的向量(vector),每个向量里面的内容是该节点所能指向的节点的集合(即存储的是该节点的直接后继节点). 两种实现方法不影响算法的分析,下面就以邻接矩阵的方式对该算法的两个关键步骤的实现进行详细的说明.

第一步　生成优化后的程序图子算法. 依据变量依赖和非关联路径隔离并删除导致产生不可执行路径的非重复的节点和边,这里的非重复可以理解为节点入度和出度均为 1 的情况. 得出优化后的程序图的子算法(伪代码)如图 9 所示.

```
1  Procedure Build_New_Graph(V,E)
2    Input node[i], v[i],v[j], R[i][j]
3    Output all new R[i][j]
4  Begin int i, j, n    array R[i][j]
5    Get n,R[i][j] from input;
6  /* Linear array for nodes */
7    For(i = 0;i < n; i++)
8      node=get (node [i]);
9  /* Search to see if i1 and i2 have variable dependency */
10   For(i = 0;i < n; i++)
11     If v def in node [i1] and v in node [i2]
12       v[i1] = v[i2];
13 /* Double dimensional array for relation of nodes */
14   For(i = 0;i < n;i++)
15     For(j = 0;j < n;j++)
16     Get R[i][j];
17   Get no-relpaths{P1,P2,…,Pk};
18 /* Delete non-repetitive nodes in no-relation paths */
19   Do up-down traverse
20 If v[i1]=v[i2] and i1 in no-rel path and i2 in no-rel path
21      If in-degree and out-degree of i1=1 delete node[i1]
22     If in-degree and out-degree of i2=1 delete node[i2]
23   End do
   End
```

图 9　生成优化后的程序图的子算法

Fig.9　Sub-algorithm of program graph optimization

第二步　生成有效路径集子算法. 依据优化后的程序图,采用遍历分别取各种分支的情况来生成有效路径集. 需要注意的是,由于生成路径时,采用的是自底向上的遍历法,所以可以先将原来表示节点关系的邻接矩阵进行转置,这样就可以通过转置后的矩阵采用自底向上遍历法来获取路径.

该子算法的具体实现(伪代码)如图 10 所示.

```
1  Procedure Find_Effective_Paths
2    Input node[i],R[i][j]
3    Output set of paths
4  Begin int p,q,i,c,in_deg[i]=0 out_deg[i]=0;
5  /*Find set of pre-nodes of node i and in-degree of node i*/
6    For(p=0,p<n,p++)
7      in-deg[i] += R[p][i];
8      If (R[p][i]=1)
9      Add p to pre-node of node [i];
10     c = c+1;
11 / * Transposition of R[i][j] * /
12   Void Trans R[i][j]
13   Set R[j][i] =R[i][j]
14 / * Traverse bottom-up to generate paths * /
15   Traverse-bottom-up(graph G)
16   Initialize paths{P(k+1),…,Pm},trans R[i][j]
17   For (i=0,i<n,i++) if in-deg[i] >= 2
18     I0= in_deg[i], m=i break;
19   For(i=m+1,i<n,i++)   if(Ii−Oi!=0)
20   VE= I0+|Ii−Oi| add i to Pm
21   End traverse
22 Add no-relpaths to paths
23 Output paths{P1,P2,P3,…,Pm}
24 End
```

图 10　生成有效路径集的子算法

Fig.10　Sub-algorithm of effective-path set generation

2.3　算法复杂度分析

(1) 时间复杂度

假设程序图中节点数为 n,根据第 2.2 中的算法实现步骤,第一步中初始化节点信息的时间复杂度为 $O(n)$,遍历一维数组找出变量依赖关系的时间复杂度也为 $O(n)$. 二维数组初始化并检查节点是否有直接相邻关系的时间复杂度是 $O(n^2)$,遍历并删除非关联路径中的非重复节点需要的时间复杂度是 $O(n)$. 所以第一步的总时间复杂度是 $3O(n)+O(n^2)\approx O(n^2)$. 第二步中找出前置节点集合的时间复杂度为 $O(n)$,将二维数组构成的矩阵转置时需要的时间复杂度是 $O(n^2)$,自底向上遍历找出有效路径时的时间复杂度是 $O(n)$,所以第二步中总的时间复杂度是 $O(n^2)+2O(n)\approx O(n^2)$. 因此整个算法的最大时间复杂度是 $O(n^2)$.

(2) 空间复杂度

同样假设程序图中节点数为 n,则创建节点的一维数组空间复杂度是 $O(n)$,邻接矩阵的二维数组空间复杂度是 $O(n^2)$,存储节点变量依赖关系的一维

数组空间复杂度为 $O(n)$，自底向上生成有效路径集时对邻接矩阵进行转置保存其副本的空间复杂度为 $O(n^2)$. 所以总的空间复杂度是：$O(n)+O(n^2)+O(n)+O(n^2)\approx O(n^2)$.

3 算法验证

根据第 2.2 节算法实现可知，本算法可以通过程序自动实现，这里不再赘述，仅以一个具体的组件例子对算法逻辑进行验证.

组件需求描述：判定三角形类型，根据用户输入的三个边数据，输出三角形的类型. 若三个边数据不满足构成三角形的条件(两边之和大于第三边)则输出“非三角形”，反之输出其具体的三角形类型(普通三角形、等腰三角形、等边三角形)，其具体的程序及程序见图 1 和 2.

(1) 根据组件程序画出程序图(见图 2).

(2) 依据组件需求描述中的业务逻辑，在程序图中找出非关联路径，将其路径从程序图中隔离并删除，保留重复的节点和循环中的路径，得到优化后的程序图. 故可以得到一条非关联路径，即为三角形类型判定失败的业务逻辑：P_1. 因条件语句在节点 4 判定失败未能进入具体类型判定的路径为 1→3→4→11→12.

依据变量依赖的定义，节点 4 变量依赖于节点 3，在路径 P_1 中，节点 3 和节点 11 的入度和出度均为 1，故可以从程序图中去掉. 由此得到优化后的程序图如图 11.

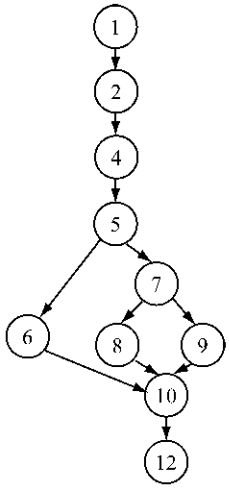

图 11 三角形类型判定组件优化后的程序图

Fig.11 Program graph of triangle judgment procedure after optimization

(3) 在图 11 的基础上，运用公式 $V_E=\sum_{i=0}^{m}(I_i-C)+1$ 计算有效路径集元素个数并找出除非关联路径外的有效路径. 由于第一个入度不小于 2 的节点是节点 10，其直接前置节点 6 和节点 8、节点 9 的入度都小于 2，所以得到

$$V_E=\sum_{i=0}^{m}(I_i-C)+1=(3-0)+0+1=4$$

即除了非关联路径外余下的有效路径有三条，结合图 11，得到余下的有效路径分别如下：

P_2：在节点 10 的直接前置节点中选节点 6 得到路径 1→2→4→5→6→10→12.

P_3：在节点 10 的直接前置节点中选节点 8 得到路径 1→2→4→5→7→8→10→12.

P_4：在节点 10 的直接前置节点中选节点 9 得到路径 1→2→4→5→7→9→10→12.

(4) 依据算法得到以上四条路径，经过分析可以知道这四条路径都是可执行的，所以这四条路径构成一个有效的测试路径集.

依据本算法可得到有效路径集并转换为相应的测试用例，可以发现四个测试用例代表的业务逻辑分别是：输入的三个数据无法构成三角形、能构成等腰三角形、能构成等边三角形、只能构成普通三角形四种情形. 由此可见，本算法得出的测试用例不仅满足 McCabe 基路径覆盖(线性无关)，且得出的有效路径集合中的路径都是可执行的，同时最大限度满足了业务逻辑覆盖.

4 结论

(1) 将业务逻辑与基路径相结合进行基路径分析，这样就保证了得到的测试用例是可执行的且符合业务逻辑的.

(2) 采用自底向上遍历程序图的方法，从汇节点出发遍历并分别取各种分支情况，在遍历过程中将白盒测试中逻辑覆盖的思想融合在其中，在最大程度上保证了测试用例符合基路径覆盖和基本的业务逻辑.

(3) 算法时间复杂度比较低，为 $O(n^2)$.

(4) 算法得出的有效路径集转化为相应的测试用例时，由于有效路径集中的路径是相互独立的，所以相应的业务逻辑也不会相互重复，避免了测试用例的冗余.

(5) 根据本算法的推导和分析过程可知，该算法也同样适用于组件间的集成测试和系统测试，可以理解为，在集成测试中节点可以对应于单个组件，

在系统测试中节点可以对应于子系统,故算法具有良好的可扩展性和应用前景.

参考文献:

[1] 杜庆峰.高级软件测试技术[M]. 北京:清华大学出版社,2011.
DU Qingfeng. Senior software engineering[M]. Beijing: Tsinghua University Press, 2011.

[2] Jorgensen P C. Software testing—a craftman's approach[M]. Boca Raton: CRC Press LLC, 1995.

[3] Nakajo T, Yamaguchi I, Kume H. A test-path determination method based on define-use chains: test conditions and program fault overlooks[J]. Systems and Computers in Japan, 1993,24(5):14.

[4] YAN Jun, ZHANG Jian. An efficient method to generate feasible paths for basis path testing[J]. Information Processing Letters,2008,107(3-4):87.

[5] Yates D F, Malevris N. Inclusion, subsumption, JJ-paths, and structured path testing: a redress[J]. Software Testing, Verification and Reliability, 2009,19(3):199.

[6] Zolotov V, XIONG Jinjun, Fatemi H. Statistical path selection for at-speed test[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2010,29(5):749.

[7] DU Qingfeng, DONG Xiao. An improved algorithm for basis path testing[C]// 2011 International Conference on Business Management and Electronic Information, BMEI 2011. Piscataway:IEEE, 2011:175-178.

[8] ZHANG Zhonglin, MEI Lingxia. An improved method of acquiring basis path for software testing[C]// 5th International Conference on Computer Science and Education, ICCSE 2010. Piscataway: IEEE Computer Society, 2010:1891-1894.

[9] 李慧贤,刘坚.过程内控制依赖的计算[J].计算机工程与应用,2003(2):78.
LI Huixian, LIU Jian. Computation of intraprocedural control dependence[J]. Software Engineering Institute,2003(2):78.

[10] 杜庆峰,李娜.白盒测试基路径算法[J].计算机工程,2009,35(15):100.
DU Qingfeng, LI Na. Basis paths algorithm of white-box testing[J]. Computer Engineering, 2009, 35(15):100.

(上接第 52 页)

[6] Hankin R K, Britter R E. TWODEE: the health and safety laboratory's shallow layer model for heavy gas dispersion Part 1. Mathematical basis and physical assumptions [J]. Journal of Hazardous Materials, 1999, 66 (3): 211.

[7] Venetsanos A, Bartzis J, Wurtz J, et al. DISPLAY-2: a two-dimensional shallow layer model for dense gas dispersion including complex features [J]. Journal of Hazardous Materials, 2003, 99 (2): 111.

[8] Allwine K, Shinn J, Steit G, et al. Overview of URBAN 2000: a multiscale field study of dispersion through an urban environment [J]. Bulletin of America Meteorological Society, 2002, 83(4): 521.

[9] Martin D, Nickless G, Price C S, et al. Urban tracer dispersion experiment in London (DAPPLE) 2003: field study and comparison with empirical prediction [J]. Atmospheric Science Letters, 2010, 11(4): 241.

[10] Hanna S R, Hansen O R, Dharmavaram S. FLACS air quality CFD model performance evaluation with Kit Fox, MUST, Prairie Grass, and EMU observations [J]. Atmospheric Environment, 2004, 38(28): 4675.

[11] Hanna S R, Brown M J, Camelli F E, et al. Detailed simulations of atmospheric flow and dispersion in downtown Manhattan: An application of five computational fluid dynamics models [J]. Bulletin of the American Meteorological Society, 2006, 87(12): 1713.

[12] Pullen J, Boris J P, Young T, et al. A comparison of contaminant plume statistics from a Gaussian puff and urban CFD model for two large cities [J]. Atmospheric Environment, 2005, 39(6): 1049.

[13] Moussafir J, Oldrini O, Tinareli G, et al. A new operational approach to deal with dispersion around obstacles: the MSS (micro swift spray) software suite [C] // Proceedings of 9th International Conference on Harmonisation Within Atmospheric Dispersion Modelling for Regulatory Purposes. Garmisch-Partenkirchen: Institute for Meteorology and Climate Research (IMK - IFU), 2004:114-118.

[14] Xie Z, Castro I P. Large-eddy simulation for flow and dispersion in urban streets [J]. Atmospheric Environment, 2009, 43(13): 2174.

[15] 刘国梁,宣捷,杜可,等. 重烟羽扩散的风洞模拟实验研究[J]. 安全与环境学报, 2004, 4(3): 27.
LIU Guoliang, XUAN Jie, DU Ke, et al. Wind tunnel experiments on dense gas plume dispersion [J]. Journal of Safety and Technology, 2006, 4(3): 27.

[16] 席学军,邓云峰. 城市地区毒气扩散事故数值模拟[J]. 中国安全生产科学技术, 2006, 2(6): 35.
XI Xuejun, DENG Yunfeng. Numerical simulation on poison gas pollutant dispersion in urban area [J]. Journal of Safety Science and Environment, 2006, 2(6): 35.

[17] 尤学一,李莉,刘伟. 城市街道内污染物扩散的数值模拟[J]. 天津大学学报, 2007, 40(9): 1077.
YOU Xueyi, LI Li, LIU Wei. Numerical simulation of pollutant dispersion in urban streets [J]. Journal of Tianjin University, 2007, 40(9): 1077.

[18] ZHENG Maohui, GUO Yuerong, FENG Xuezhi, et al. Numerical simulation and analysis of hazardous gas dispersion in urban sub-domain [C] // Proceedings of 19th International Conference on Geoinformatics. [S. l.]: IEEE Computer Society Press, 2011: 2305-2309.

[19] Chan T L, Dong G, Leung C W, et al. Validation of a two dimensional pollutant dispersion model in an isolated street canyon [J]. Atmospheric Environment, 2002, 36(5): 861.

[20] Nielsen M, Ott S. A collection of data from dense gas experiments, Risø Laboratory Report: Risø-R-845 [R]. Roskilde: Risø National Laboratory, 1996.