计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200700021

基于支配关系的数据流测试用例生成方法

吉顺慧　张鹏程
河海大学计算机与信息学院　南京 211100

摘　要　程序控制流的设计是为实现正确的数据流服务的,数据流测试是非常重要的。文中将面向 all-uses 数据流准则的测试用例生成问题建模为多目标优化问题,提出了一种基于遗传算法的测试用例生成方法。通过构建待测程序的控制流图进行数据流分析,计算出程序中所有的定义-使用对,得到测试目标,利用面向多测试目标的遗传算法生成满足 all-uses 准则的最优解。遗传算法中定义了一种改进的基于支配关系的适应度函数,在分析测试用例对定义-使用对的覆盖程度时考虑了存在重定义的可能性,且考虑了定义结点和使用结点在执行路径中的先后顺序。实验结果表明,所提方法可以有效地生成满足 all-uses 准则的测试用例,相比其他方法可以有效地提升测试目标的覆盖率,降低生成测试用例所需的迭代次数。

关键词:数据流测试;测试用例生成;遗传算法;适应度函数;支配结点
中图法分类号　TP311

Test Case Generation Approach for Data Flow Based on Dominance Relations

JI Shun-hui and ZHANG Peng-cheng
College of Computer and Information,Hohai University,Nanjing 211100,China

Abstract　The design of control flow in programs serves for realizing correct data flow. Performing the data flow testing is important. With formulating the problem of all-uses data flow criterion oriented test case generation as a many-objectives optimization problem,a genetic algorithm based test case generation approach is proposed. By constructing the control flow graph for to-be-tested program,data flow analysis is performed to compute all the definition-use pairs which are the testing requirements. Then many-objectives oriented genetic algorithm is performed to search the optimal solution for satisfying all-uses criterion. An improved fitness function is defined based on the dominance relations. The existence of killing definition,as well as the sequence of definition node and use node in the execution path,are taken into consideration to analyze the coverage of test case with respect to the definition-use pair. Experimental results show that the proposed approach can effectively generate test cases for satisfying all-uses criterion. And compared with other approaches,it can improve the coverage percentage and reduce the number of generations.

Keywords　Data flow testing,Test case generation,Genetic algorithm,Fitness function,Dominance node

1　引言

软件测试的一项关键工作是针对给定的充分性覆盖准则设计测试用例。尽管 all-paths 准则是最强的覆盖准则,但它并不能检测程序中所有的缺陷,且当程序中存在循环时有可能存在无限多的路径,在这种情况下通过测试覆盖所有路径是很难实现的[1]。程序正确性最基本的要求是,对于任意给定的输入都可以得到期望的输出。输入与输出之间的关联是通过程序中一系列变量的定义和使用关联来实现,可以认为控制流的设计是为了实现正确的数据流[2]。因此,进行数据流测试是非常重要的。然而,针对数据流准则的测试研究还相对较少。

通过定义合适的适应度函数来描述测试用例对测试目标的覆盖程度,测试用例生成问题可以被建模为一个函数优化问题[3]。遗传算法是一种常用的解决方法[4-5],其利用适应度函数指导有效输入域内的最优解搜索。Ghiduk 等提出了基于程序控制流图(Control Flow Graph,CFG)结点支配关系的适应度函数[5]。该函数针对数据流测试目标,计算测试用例对测试目标的接近度,进而比较测试用例的好坏。然而,该函数认为,对于变量的定义-使用对而言,某测试用例的执行路径若覆盖了定义结点和使用结点的所有支配结点,则一定覆盖定义-使用对,忽略了执行路径中存在重定义的可能性。另外,该函数没有考虑定义结点和使用结点在执行路径中的先后顺序,导致针对某些测试目标产生不正确的解决方案,使其

到稿日期:2020-07-03　返修日期:2020-08-16　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61702159);江苏省自然科学基金(BK20170893)
This work was supported by the National Natural Science Foundation of China (61702159) and Natural Science Foundation of Jiangsu Province (BK20170893).
通信作者:吉顺慧(shunhuiji@hhu.edu.cn)

无法在测试中被覆盖。此外，很多遗传算法一次只能为一个测试目标生成测试用例[6]，而生成的测试用例可能同时覆盖其他测试目标，这些测试目标却需要另外的遗传迭代来生成测试用例，从而导致针对多测试目标覆盖的测试用例生成方法比较低效。

针对上述问题，本文提出了一种面向 all-uses 数据流测试准则的测试用例生成方法。all-uses 准则要求测试用例可以执行覆盖程序中变量的每个定义-使用对的至少一条路径[1]，即要求测试覆盖程序中所有的定义-使用对。本文方法包括 3 个步骤：首先，构建 CFG 模型以抽象地描述待测程序，便于实现后续步骤的自动化；其次，基于 CFG 模型分析程序的数据流，计算满足 all-uses 准则的待测试目标；最后，利用遗传算法生成覆盖测试目标的测试用例集。本文的主要贡献包括以下 3 点：

(1)提出了一种改进的基于支配关系的适应度函数，在分析测试用例对定义-使用对的覆盖程度时考虑了存在重定义的可能性，且考虑了定义结点和使用结点在执行路径中的先后顺序。

(2)将面向 all-uses 准则的测试用例生成问题建模为多目标优化问题，以便在遗传算法的一次运行中覆盖多个测试目标，并且随着测试用例的生成，目标数量渐减。

(3)通过一组实验分析了本文方法的有效性，并与已有方法进行比较。实验表明，本文方法可以有效地生成满足 all-uses 准则的测试用例，相比其他方法，可以有效地提升测试目标的覆盖率，降低生成测试用例所需的迭代次数。

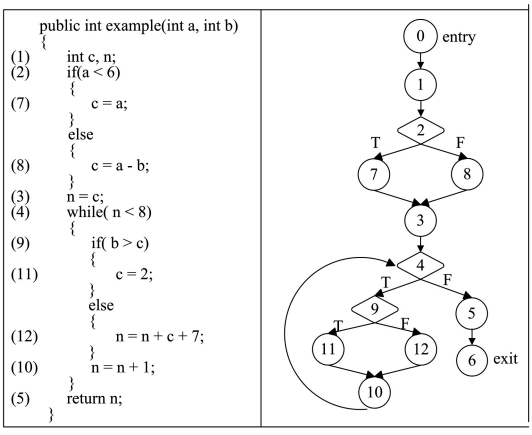


图 1 example 程序及其控制流图

Fig. 1 Program example and its control flow graph

2 背景知识

2.1 数据流测试

数据流测试大多是基于程序的控制流图 CFG 开展的。CFG 是一个有向图(V,E)[5]，其中 V 是结点集，每个结点表示一个程序语句，并且一个控制流图中有唯一的开始结点 n_{entry}和结束结点 n_{exit}，分别表示程序的开始和结束。E 是有向边的集合，每条有向边(n_i,n_j)表示结点 n_i 和 n_j 对应语句之间的顺序关系，n_i 被称为 n_j 的前驱，n_j 被称为 n_i 的后继[1]。若一个结点有两个后继，则表明该结点对应的语句是一个条件语句。图 1 给出了一个程序案例[5]及其对应的 CFG，其中 CFG 结点编号对应程序语句编号。

数据流测试主要关注程序中变量的定义和使用之间的关联。程序中变量的出现可以分为 3 种情况[1]：定义、计算使用和谓词使用，分别记为 def，c-use 和 p-use。变量的计算使用直接影响正在执行的计算，可能间接影响程序的控制流；而谓词使用直接影响程序的控制流，可能间接影响后续执行的计算。语句对变量的 def 或 c-use 存储在对应的 CFG 结点中，对变量的 p-use 存储在对应选择结点的后继结点中。如图 1 所示，结点 7 包含对变量 c 的定义和对变量 a 的计算使用，分别记为 $def(7)=\{c\}$，$cuse(7)=\{a\}$。有向边(2,7)包含对变量 a 的谓词使用，可简化表示为结点 7 对变量 a 的谓词使用，记为 $puse(7)=\{a\}$。

在程序 CFG 中，一条路径是一个结点序列$(n_1,n_2,\cdots,n_m)$，其中任意两个相邻结点之间存在一条有向边，即$(n_i,n_j)\in E(i=1,2,\cdots,m-1)$。若从开始结点到 n_j 的每条路径都经过结点 n_i，则称 n_i 是 n_j 的支配结点[5]。对于图 1 中的结点 12，其支配结点为 0,1,2,3,4,9,12。若路径$(n_1,n_2,\cdots,n_m)$中的结点 $n_2,\cdots,n_{m-1}$都不存在对变量 var 的定义，则称该路径是关于 var 的 def-clear 路径。

根据变量的两种使用类型，程序中的定义-使用对可以分为定义-计算使用对和定义-谓词使用对两种类型。定义-计算使用对 $dcu=(var,n_i,n_j)$表示结点 n_i 包含对变量 var 的定义，且结点 n_j 包含对该定义的计算使用。定义-谓词使用对 $dpu=(var,n_i,n_j)$表示结点 n_i 包含对变量 var 的定义，且边(n_k,n_j)包含对该定义的谓词使用，其中 n_k 作为 n_j 的前驱结点，对应某个条件语句。对于 CFG 的一条完整路径 $p=(n_{\text{entry}},n_1,n_2,\cdots,n_m,n_{\text{exit}})$以及定义-使用对 $du=(var,n_i,n_j)$，当且仅当 p 包含关于变量 var 的 def-clear 路径$(n_i,n_{i+1},\cdots,n_j)$，则称 p 覆盖 du。

2.2 遗传算法

遗传算法通过模拟自然界的进化过程来解决问题，是搜索问题和优化问题的常用解决方法[4]。它通过生成个体的初始种群，利用选择机制以及基于交叉和变异的搜索机制来对初始种群进行演化，以发现问题的最优解。在此过程中，通过适应度函数评估每个个体的好坏程度。具有高适应度的个体被选择到下一代，并通过交叉和变异来产生新的种群。

基本的遗传算法如算法 1 所示，其通过评估每个个体的适应度，可以确定是否找到最优解。该算法选择具有高适应度值的个体到下一代种群，然后通过交换个体的数据片段对两个个体执行交叉操作，最后对种群中的部分个体执行变异操作，从而得到一个新的种群。算法如此循环，直到找出问题的最优解或者到达预设的最大迭代次数。

算法 1 遗传算法

1. Generate an initial population: P_t(t=0);
2. Evaluate the fitness of each individual in P_t;
3. while (termination criterion not reached) do
4. Select P_{t+1} from P_t;
5. Perform crossover to P_{t+1};
6. Perform mutation to P_{t+1};
7. Evaluate the fitness of each individual in P_{t+1};
8. t=t+1;
9. End while

3 本文方法

3.1 问题建模

对于输入域为 X 的待测程序，当且仅当输入 $x \in X$ 的执行路径 $p(x)$ 覆盖定义-使用对 du，才能称 x 覆盖 du。为了在遗传算法中搜索覆盖 du 的最优测试用例，需要定义合适的适应度函数来评估测试用例关于测试需求的好坏。

文献[5]所提出的适应度函数将定义-使用对(var,d,u)的覆盖看作两个目标，分别是定义结点 d 的覆盖和使用结点 u 的覆盖。测试用例 x 对定义-使用对(var,d,u)的适应度是基于支配关系计算的，公式如下：

$$ft(d,u,x)=\frac{1}{2}\times\left(\frac{|cdom(d)|}{|dom(d)|}+\frac{|cdom(u)|}{|dom(u)|}\right) \tag{1}$$

其中，$dom(d)$ 是定义结点 d 的支配结点集，$dom(u)$包括使用结点 u 的支配结点集和定义结点 d，$cdom(d)$和 $cdom(u)$分别是 $dom(d)$和 $dom(u)$中被 $p(x)$覆盖的结点集。此外，若存在两个目标的某支配结点未被 $p(x)$覆盖，则适应度值加上 CL，其公式如下：

$$CL=\frac{1}{2}\times\min\left(0.9,\frac{|fdom(d\vee u)|-|udom(d\vee u)|}{|dom(d)|\times|dom(u)|}\right) \tag{2}$$

其中，$fdom(d\vee u)$是 $dom(d)$和 $dom(u)$中被 $p(x)$覆盖超过 1 次的结点集，$udom(d\vee u)$是 $dom(d)$和 $dom(u)$中未被 $p(x)$覆盖的结点集。

若测试用例 x 的适应度值为 $ft(d,u,x)=1$，则 x 是定义-使用对(var,d,u)的最优解。即若 x 的执行路径 $p(x)$覆盖了 d 和 u 的支配结点，则 x 就是覆盖(var,d,u)的最优解。然而，该适应度函数的定义不适用于以下两种情况：

1)路径 $p(x)$中，在结点 d 和 u 之间存在针对变量 var 的重定义；

2)路径 $p(x)$先到达使用结点 u，后到达定义结点 d。

对于第一种情况，若路径 $p(x)$中，在结点 d 和 u 之间存在另一个定义变量 var 的结点，则该结点会杀死定义结点 d 对 var 的定义，导致 d 的定义无法到达结点 u，从而无法覆盖(var,d,u)。以图 1 中的定义-计算使用对$(c,8,12)$为例，假设测试用例 x 的执行路径为 $p(x)=\{0,1,2,8,3,4,9,11,10,4,9,12,10,4,5,6\}$。由于 $dom(8)=\{0,1,2,8\}$，$dom(12)=\{0,1,2,3,4,9,12,8\}$，由式(1)可得到 $ft(8,12,x)=1$，从而确定 x 是覆盖$(c,8,12)$的最优解。而实际上，由于结点 8 对变量 c 的定义被结点 11 中的定义杀死，$p(x)$并未覆盖$(c,8,12)$。该适应度函数忽略了存在重定义的可能性，会产生不正确的适应度值，从而导致一些测试目标无法被覆盖。

对于第二种情况，若路径 $p(x)$先到达使用结点 u，后到达定义结点 d，虽然 $p(x)$覆盖了 d 和 u 的支配结点，但并未覆盖定义-使用对(var,d,u)。以图 1 中的定义-谓词使用对$(n,10,9)$为例，假设测试用例 x 的执行路径为 $p(x)=\{0,1,2,8,3,4,9,12,10,4,5,6\}$。由于 $dom(9)=dom(10)=\{0,1,2,3,4,9,10\}$，根据式(1)可得到 $ft(10,9,x)=1$，从而确定 x 是覆盖$(n,10,9)$的最优解。而实际上，在路径 $p(x)$中结点 10 的定义并未到达边$(4,9)$，即 $p(x)$并未覆盖$(n,10,9)$。按该适应度函数生成测试用例会导致一些目标定义-使用对被遗漏。

针对以上存在的问题，我们对适应度函数进行了改进。对于目标定义-使用对(var,d,u)，若 d 未被 $p(x)$覆盖，则可以利用式(1)和式(2)计算 x 对(var,d,u)的覆盖度。若 d 被 $p(x)$覆盖了，则 $p(x)$中至少有一个结点 d。计算测试用例 x 覆盖使用结点 u 的适应度，首先需要计算 $p(x)$的子路径，使得 d 中的定义保持活性。对 $p(x)$中出现的每个结点 d，从 d 开始识别下一个对变量 var 进行定义的结点 n_i，若不存在这样的结点，则 n_i 是 $p(x)$中的最后一个结点，得到子路径 $sp(x)=(d,n_1,n_2,\cdots,n_i)$。$p(x)$中从 $entry$ 结点到第一个定义结点 d 之间的子路径则记为 $pre(x)$。此外，为了确保(var,d,u)的覆盖路径中，使用结点 u 应出现在定义结点 d 之后，需要计算 d 与 u 之间 u 的支配结点集 $dom1(u)$。$dom(u)$中除 $dom1(u)$以外的支配结点被记为 $dom2(u)$。d 被 $p(x)$覆盖的情况下，测试用例 x 关于定义-使用对(var,d,u)的适应度定义如下：

$$ft'(d,u,x)=0.5+\frac{1}{2}\times\frac{|cdom1(u)|+|cdom2(u)|}{|dom(u)|} \tag{3}$$

其中，$cdom1(u)$是 $dom1(u)$中被子路径 $sp(x)$覆盖的结点集，$cdom2(u)$是 $dom2(u)$中被子路径 $pre(x)$覆盖的结点集。

若测试用例 x 未覆盖定义-使用对(var,d,u)，即 $ft'(d,u,x)\neq1$，由于定义结点 d 已被覆盖，则将下式定义的 CL'加到适应度值中。

$$CL'=\frac{1}{2}\times\min\left(0.9,\frac{|fdom'(u)|-|udom'(u)|}{|dom(d)|\times|dom(u)|}\right) \tag{4}$$

其中，$fdom'(u)$是 $dom1(u)$中被子路径 $sp(x)$覆盖超过 1 次的结点集，$udom'(u)$是 $dom1(u)$中未被子路径 $sp(x)$覆盖的结点集。

因此，测试用例 x 关于定义-使用对(var,d,u)的适应度函数定义如下：

$$f(d,u,x)=\begin{cases}ft(d,u,x)+CL, & \text{若 } p(x)\text{未覆盖 } d\\ \max(ft'(d,u,x)+CL'), & \text{否则}\end{cases} \tag{5}$$

其中，当 $ft'(d,u,x)$值为 1 时，CL' 值为 0，max 表示对定义 d 保持活性的多条子路径取最大值。

根据改进后的适应度函数定义，对于图 1 中的定义-计算使用对$(c,8,12)$，当测试用例 x 的执行路径为 $p(x)=\{0,1,2,8,3,4,9,11,10,4,9,12,10,4,5,6\}$时，因为定义结点 8 被 $p(x)$覆盖，其适应度值应根据式(3)和式(4)来计算，可找到 $p(x)$的一条子路径 $sp(x)=\{8,3,4,9,11\}$，存在于定义结点 8 和使用结点 12 之间并且属于结点 12 的支配结点为 $dom1(12)=\{3,4,9,12\}$，最终的适应度值 $f(8,12,x)$的计算过程如下：

$$ft'(4,9,x)=0.5+\frac{1}{2}\times\frac{3+4}{8}=0.9375$$

$$CL'=\frac{1}{2}\times\min\left(0.9,\frac{0-1}{4\times8}\right)=-0.015625$$

$$f(4,9,x)=0.9375-0.015625=0.921875$$

根据适应度函数的定义，覆盖目标定义-使用对(var,d,u)的测试用例生成问题，等价于适应度函数 $f(d,u,x)$的最优解搜索问题，即使得 $f(d,u,x)$的值最大化。面向 all-uses 准则的数据流测试需要覆盖所有的定义-使用对，假设程序中有 n 个定义-使用对$(var_1,d_1,u_1),(var_2,d_2,u_2),\cdots,(var_n,d_n,u_n)$，我们的目标就是要找到至少 m 个测试用例 $x_1,x_2,\cdots,$

x_m，使得 $f(d_1,u_1,x),f(d_2,u_2,x),\cdots,f(d_n,u_n,x)$ 最大化，进而在测试中覆盖所有 n 个定义-使用对。面向 all-uses 准则的测试用例生成问题可以建模为一个如下的多目标优化问题[6]：

$$\max(f(d_1,u_1,x),f(d_2,u_2,x),\cdots,f(d_n,u_n,x)) \quad (6)$$

若存在某个输入 $x\in X$ 使得 $f(d_i,u_i,x)=1$，则 x 可以覆盖定义-使用对(var_i,d_i,u_i)，是期望得到的一个测试用例。

3.2 数据流分析

数据流分析的目标是计算出待测程序中所有的定义-使用对，以确定满足 all-uses 准则的测试目标。

数据流分析的基础是构建待测程序的 CFG 模型，将程序中的每一条语句转化为一个 CFG 结点，将语句之间的顺序关系转化为对应 CFG 结点之间的边；同时，还需要将语句中定义和使用的变量信息存储在对应的 CFG 结点中。若待测程序存在输入变量，执行程序时需要输入变量的定义，则将对输入变量的定义存储在程序体中首条语句对应的 CFG 结点中。待测程序到 CFG 的转换可以借助抽象语法树 AST (Abstract Syntax Tree)来实现。

基于控制流图 CFG，传统的可达定义分析方法[7]可以计算在每个程序结点处可到达的变量定义。一个变量定义用(var,n)来标识，表示变量 var 在程序结点 n 处被定义，若程序中存在一条从 n 到 n'的关于 var 的 def-clear 路径，则表明变量定义(var,n)可以达到结点 n'。令 $In(n)$和 $Out(n)$分别表示程序结点 n 的入口处和出口处可达的变量定义集，则可达定义的计算公式如下[7]：

$$Out(n)=Gen(n)\cup(In(n)-Kill(n)) \quad (7)$$

$$In(n)=\bigcup_{p\in pred(n)}Out(p) \quad (8)$$

其中，$Gen(n)$和 $Kill(n)$分别表示程序结点 n 产生的定义集和杀死定义集。若程序结点 n 定义了变量 x，则$(x,n)\in Gen(n)$。若程序中存在另外的结点 m 定义了变量 x，则$(x,m)\in Kill(n)$。另外，$pred(n)$表示 CFG 中结点 n 的前驱结点集合。

式(7)和式(8)的求解可以采用经典的 MFP(Maximal Fixed Point)方法[7]，通过迭代算法计算每个程序点的方程式，直至最终收敛。

基于每个程序结点入口处的可达定义，可以利用下式获取程序中关于变量 var 的所有定义-使用对(var,n,n')：

$$du(var,n)=\{n'\mid(var,n)\in In(n')\wedge var\in use(n')\} \quad (9)$$

其中，$use(n')=cuse(n')\cup puse(n')$。

针对图 1 所示的案例程序，应用数据流分析方法得到程序中所有的定义-使用对，如表 2 所列，共有 29 个，其中 Computation 表示定义-计算使用对，Predicate 表示定义-谓词使用对。

表 1 example 程序的定义-使用对集合

Table 1 def-use pairs in program example

Computation	(a,1,7),(a,1,8),(b,1,8),(c,7,3),(c,8,3),(c,7,12),(c,8,12),(c,11,12),(n,3,12),(n,10,12),(n,3,10),(n,10,10),(n,12,10),(n,3,5),(n,10,5)
Predicate	(a,1,7),(a,1,8),(b,1,11),(b,1,12),(c,8,12),(c,7,11),(c,8,11),(c,11,11),(c,7,12),(c,11,12),(n,3,9),(n,10,9),(n,3,5),(n,10,5)

3.3 测试用例生成

针对数据流分析得到的定义-使用对，本节利用遗传算法搜索覆盖每个定义-使用对的测试用例，使得 all-uses 测试准则得以满足。

(1) 编码

将遗传算法应用到测试用例生成问题，种群中的每个个体代表一个测试用例。本文使用二进制编码方式将个体表示成一个二进制字符串[4-5]，字符串的长度由每个输入变量的取值范围和精度确定。

假设待测程序有 k 个输入变量 $x_1,x_2,\cdots,x_k$，每个变量 x_i 的取值范围为$[a_i,b_i]$，需要的精度为 d_i 个小数位，则 x_i 可用长度为 m_i 位的二进制字符串表示，m_i 取满足$(b_i-a_i)\times 10^{d_i}\leqslant 2^{m_i}-1$ 的最小值。相反地，对一个长度为 m_i 位的二进制字符串计算其实际值 x_i 的解码公式如下：

$$x_i=a_i+x_i'\times\frac{b_i-a_i}{2^{m_i}-1} \quad (10)$$

其中，x_i'是二进制字符串的的十进制值。

若 x_i 是整数型输入变量，则 d_i 设置为 0，解码公式如下：

$$x_i=a_i+int(x_i'\times\frac{b_i-a_i}{2^{m_i}-1}) \quad (11)$$

通过计算每个输入变量的编码长度，可以得到个体的编码长度为 $m=\sum_{i=1}^{k}m_i$。m 位的二进制编码中，前 m_1 位的编码表示输入变量 x_1，接下来 m_2 位的编码表示输入变量 x_2，依次类推，最后 m_k 位的编码表示输入变量 x_k。

以图 1 中的程序为例，令输入变量 a 和 b 的取值范围分别为$[-1,10]$和$[-3,14]$，则 a 需要长度为 4 的二进制编码，b 需要长度为 5 的二进制编码，种群中的个体长度为 9。假设存在个体 101101010，则前 4 位 1011 表示输入变量 a 取值为 7，后 5 位 01010 表示输入变量 b 取值为 2。

(2)适应度评估

对于目标定义-使用对而言，种群中的个体是否为其所期望的测试用例，需要采用式(5)来评估个体对各定义-使用对的适应度，进而用式(6)评估个体对测试目标集的整体适应度。

测试用例 $x\in X$ 关于定义-使用对(var,d,u)的适应度评估包括以下 4 个步骤：

1)在控制流图 CFG 中，计算定义结点 d 和使用结点 u 的支配结点集 $dom(d)$ 和 $dom(u)$，并将结点 d 添加到集合 $dom(u)$中。

2)对待测程序进行插装，以记录程序的执行路径。以测试用例 x 为输入运行待测试程序，获取程序的执行路径 $p(x)$。

3)判定执行路径 $p(x)$中是否包含结点 d，若不包含，则对执行路径 $p(x)$计算 $dom(d)$和 $dom(u)$中被覆盖的结点集 $cdom(d)$和 $cdom(u)$，$dom(d)$和 $dom(u)$中未被覆盖的结点集 $udom(d\vee u)$，以及 $dom(d)$和 $dom(u)$中被覆盖超过 1 次的结点集 $fdom(d\vee u)$。然后利用式(1)和式(2)计算适应度值。

4)若执行路径 $p(x)$中包含结点 d，计算结点 d 和 u 之间属于 $dom(u)$的支配结点 $dom1(u)$，进而得到 $dom2(u)$。计算 $p(x)$中定义 d 保持活性的子路径 $sp(x)$，进而得到 $pre(x)$。计

算 $dom2(u)$中被子路径 $pre(x)$覆盖的结点集 $cdom2(u)$，对每条子路径 $sp(x)$计算 $dom1(u)$中被覆盖的结点集 $cdom1(u)$、未被覆盖的结点集 $udom'(u)$，以及被覆盖超过 1 次的结点集 $fdom'(u)$。然后利用式(3)和式(4)计算适应度值，并对多条子路径 $sp(x)$的计算值取最大值。

本文中控制流图 CFG 各结点的支配结点集采用文献[8]中的方法进行计算。

(3)遗传操作

遗传算法中，若当前种群无法找到最优解，首先需要通过选择操作将当前种群中的优秀个体遗传到下一代，然后，通过交叉和变异这两种遗传操作来生成新的个体，从而实现种群的演化。

选择：选择操作可采用轮盘赌方法从当前种群中选取下一代种群的父集合。其具体实现包括以下几个步骤[4]：

1)针对目标定义-使用对，根据上述方法计算当前种群中每个个体 x_i ($i=1,2,\cdots,pop\_size$)的整体适应度值 $fitness(x_i)$，其中 pop_size 表示种群的大小。

2)计算种群所有个体的适应度值之和 $F=\sum_{i=1}^{pop\_size} fitness(x_i)$，并计算每个个体 x_i 被选中的概率 $p_i=fitness(x_i)/F$。

3)计算每个个体 x_i 的累计概率 $q_i=\sum_{j=1}^{i} p_j$。

4)随机生成一个 0～1 之间的数 r，若 $r\leqslant q_1$，则选择第一个个体 x_1；否则若 $q_{i-1}<r\leqslant q_i$ ($i=2,3,\cdots,pop\_size$)，则选择个体 x_i。

交叉：交叉操作需要根据预先设定的交叉概率 p_c 来实现[4]。对于新种群中的每个个体，随机生成一个 0～1 之间的数 r，若 $r<p_c$，则将该个体加入待交叉的个体集，然后从个体集中随机选择两个个体执行交叉操作。对 m 位二进制码表示的两个个体，随机生成一个整数 pos 来指定交叉点的位置，$1\leqslant pos\leqslant m-1$，对两个个体的 $pos\sim m-1$ 位置的字符进行交叉。

变异：变异操作在交叉操作之后执行，也需要预先设置变异概率 p_m[4]。每个个体中的每个字符位都会根据概率 p_m 进行变异。对每个个体中的每个字符位，随机生成一个 0～1 之间的数 r，若 $r<p_m$，则对该位置的字符进行变异，即将 0 变成 1 或相反。

基于遗传算法的测试用例生成算法的输入除了包括遗传算法必须的种群大小 pop_size、交叉概率 p_c、变异概率 p_m、最大迭代次数 max_gen 之外，还包括待测的插装程序 P'、待测的定义-使用对集合 du-pairs、输入变量的取值范围和精度，算法的输出是一组测试用例集 TC。该算法的一次迭代中，种群面临的目标是所有未覆盖的定义-使用对，若某些定义-使用对被该种群中的个体覆盖，则将它们从待覆盖的定义-使用对集合中删除，如此随着测试用例的生成，待覆盖的目标数量渐减。

针对图 1 所示的案例程序，应用遗传算法得到的一组测试用例如表 2 所列。其中，第 1 列表示遗传算法中的个体，即测试用例的二进制表示，第 2 列表示测试用例的实际数值，第 3 和第 4 列表示测试用例覆盖的定义使用对。

表 2 针对 example 程序生成的测试用例集

Table 2 Test cases generated for program example

Individual	Test case	Covered Def-use pairs	
		Computation	Predicate
000010000	$a=-1$, $b=5$	$(c,7,3)$,$(a,1,7)$, $(n,10,5)$,$(n,10,10)$, $(n,3,10)$	$(c,11,11)$,$(c,7,11)$, $(a,1,7)$,$(n,10,9)$, $(n,10,5)$,$(n,3,9)$, $(b,1,11)$
110011001	$a=7$, $b=10$	$(a,1,8)$,$(b,1,8)$, $(c,8,3)$	$(a,1,8)$,$(c,8,11)$
110100110	$a=8,b=0$	$(n,3,5)$	$(n,3,5)$
110101110	$a=8,b=4$	$(n,3,12)$,$(c,8,12)$, $(n,12,10)$	$(b,1,12)$,$(c,8,12)$
010000110	$a=1,b=0$	$(c,7,12)$	$(c,7,12)$
010001010	$a=1,b=2$	$(c,11,12)$,$(n,10,12)$	$(c,11,12)$

4 实验分析

我们根据本文方法开发了原型工具 DFOTCG(Data Flow Oriented Test Case Generation)，来实现程序的测试用例生成，本节将讨论如何通过一些程序案例来表明方法的有效性和性能。

4.1 实验设计

本文的测试用例生成方法面向 all-uses 测试准则，提出了改进的适应度函数，且在遗传搜索过程的一次运行中以多个定义-使用对为覆盖目标。为了证明本文方法的有效性和性能，设计了如下两个研究问题。

RQ1：本文方法生成的测试用例在测试目标的覆盖度方面的有效性如何？

RQ2：本文方法为了生成满足 all-uses 准则的最优解，在遗传迭代次数方面的性能表现如何？

实验使用了 4 个 Java 程序进行应用，以回答上述两个问题。研究对象包括图 1 中的案例 Example (prog1)，Triangle classifier (prog2)，Middle Value (prog3)和 Power x^y (prog4)，它们都是在测试用例生成的研究中常用的案例程序[4,5,9]。表 3 展示了这些程序的相关信息，其中第 2 列表示程序的代码行数，第 3 列表示程序中存在的定义-使用对个数，其中不可行的定义-使用对已通过分析从测试目标集中删除，最后一列给出了程序的简要描述。

表 3 对象程序

Table 3 Subject programs

Program	LOC	du_pairs	Description
prog1	26	29	Shown in Fig. 1
prog2	27	29	Find the triangle type
prog3	23	28	Find the middle value
prog4	19	18	Find the value of x^y

此外，遗传算法的参数设置为：pop_size=10，$p_c=0.8$，$p_m=0.15$，max_gen=1 000。关于交叉概率和变异概率的设定，许多基于遗传算法的数据流测试用例生成研究中都采用该概率值[4,5,9]。种群大小和最大迭代次数原本按照文献[5]进行设置，但在种群大小为 10 的基础上设置最大迭代次数为 200 时，其中两个测试对象的测试用例生成无法覆盖所有的定义-使用对，因此本文将最大迭代次数设置为 1 000，使得算法能生成满足 all-uses 测试准则的测试用例集。

4.2 实验结果

为了体现本文方法的有效性和性能，本文采用随机生成法，以及基于 Ghiduk 等[5]提出的适应度函数的遗传搜索，来与本文方法进行比较。为了比较的公平性，随机生成法每次迭代生成的测试用例数量设定为与 pop_size 相同的值。考虑到遗传算法中初始种群的随机性以及遗传操作的概率性，我们对每个对象程序执行 10 次实验，以观测如下实验结果。

(1)平均覆盖率

我们用下式定义的覆盖率来衡量测试用例生成方法对测试目标的覆盖度。

$$meanCR=\sum(\frac{|coveredPairs|}{|du-pairs|}\times 100\%)/10$$

其中，$du-pairs$ 是程序中的所有定义-使用对集合，$coveredPairs$ 是在一次实验中被测试用例覆盖的定义-使用对集合。

(2)平均迭代次数

在最大迭代次数的限制下，达到最高覆盖率所需的平均迭代次数可以用下式衡量：

$$meanGens=\sum gen/10$$

其中，gen 是在一次实验中生成测试用例所需的迭代次数。

表 4 列出了针对 4 个研究对象、使用 3 种方法得到的平均覆盖率和平均迭代次数。

表 4 实验结果

Table 4 Experimental results

Program	Mean coverage percentage/%			Mean number of generations		
	Our approach	Ghiduk et al.[5]	Random Generation	Our approach	Ghiduk et al.[5]	Random Generation
prog1	100	92.07	96.9	535	517	623
prog2	100	100	95.86	388	470	616
prog3	100	97.14	100	2	2	3
prog4	100	88.89	100	38	28	42

针对研究问题 RQ1，如图 2 所示，本文方法对所有案例的定义-使用对都实现了 100%的覆盖率，设计的测试用例能够满足 all-uses 数据流测试准则。对于 prog1，prog3 和 prog4，本文方法的覆盖率明显高于采用 Ghiduk 等提出的适应度函数得到的覆盖率。对于 prog1 和 prog2，本文方法的覆盖率优于随机生成法的覆盖率。

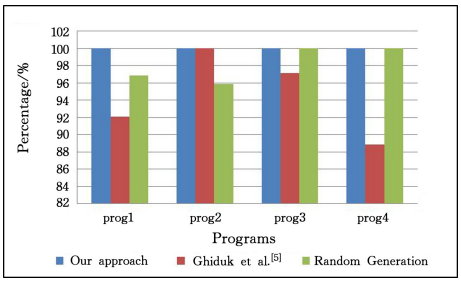


图 2 Mean coverage percentage

Fig. 2 平均覆盖率

针对研究问题 RQ2，如图 3 所示，对于所有案例的测试用例生成，本文方法的评价迭代次数都低于随机生成法，在 prog1 和 prog2 两个案例上的优势更明显。由于基于 Ghiduk 等所提适应度函数的测试用例生成采用的遗传算法与本文相同，因此其迭代次数与本文方法相比没有明显差别。

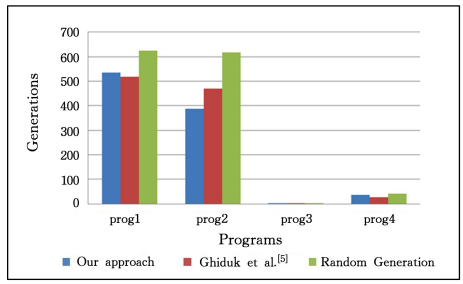


图 3 Mean number of generations

Fig. 3 平均迭代次数

尽管本文实验选取的研究对象规模不大，但这些案例程序中包含了大规模程序中常用的语句结构，因此本文方法在面向实际的大规模测试对象时也是可行的。

5 相关工作

目前已有很多研究基于搜索方法进行测试用例生成，但针对数据流准则的研究还相对较少。

Girgis 首次提出面向 all-uses 数据流准则的基于遗传算法的测试用例生成[4]，测试用例的适应度函数定义为被覆盖的数据流路径数与待覆盖的数据流路径数的比率。Andreou 等面向 All-DU-Paths 数据流准则采用遗传算法来生成测试用例[10]，提出的适应度函数在考虑定义使用路径覆盖率的同时，还考虑了未被覆盖路径和难以覆盖路径的覆盖率。Deng 等针对面向对象的程序考虑了类的测试用例生成[11]，通过定义-使用对的覆盖率来指导遗传算法的解搜索。对于两个测试用例，若它们覆盖的目标数相同或者没有覆盖任何目标，则基于覆盖率的适应度函数计算出的适应度值是相同的，无法比较哪一个测试用例对测试目标来说更好，如此更优的测试用例可能会被忽略。

Jaffari 等[12]设计了一种新的适应度函数来指导面向定义-使用对覆盖的遗传算法，该函数不仅包括未覆盖定义-使用对比率的计算，还包括测试用例与守卫条件之间距离远近的评估。Vivanti 等针对 Java 类中方法内、方法间和类间的定义-使用对，提出了基于遗传算法的测试用例生成方法[13]，定义了基于层接近度和分支距离的适应度函数来分析对定义结点和使用结点的覆盖，考虑了定义结点的重定义结点排除。Chen 等[14]利用遗产算法生成覆盖定义-使用对的测试用例，基于接近度和分支距离分别计算测试用例到定义结点和使用结点的距离，同时加上惩罚值考虑了定义结点的重定义问题。Jiang 等对层接近度和分支距离的传统适应度函数进行了改进[15]，基于测试用例的执行路径和定义-使用对的 def-clear 路径的匹配度来考虑适应度。层接近度和分支距离是定义适应度函数的常用方法，但当分别考虑定义结点和使用结点的覆盖时，若忽略定义结点和使用结点的先后覆盖顺序，则可能导致定义-使用对的覆盖遗漏。

Ghiduk 等为面向 all-uses 准则的数据流测试用例生成提出了基于结点支配关系的适应度函数[5]，但该适应度函数忽略了存在定义结点的重定义的可能性。Varshney 等[9]基于此提出了改进的适应度函数，若测试用例的执行路径包含定义结点的重定义结点，直接将其适应度赋值为 0，并用分支距

离取代 CL 以考虑与未覆盖的支配结点间的距离。然而，简单地将包含重定义结点的测试用例的适应度值设为 0，将导致这类测试用例的适应度值可能低于未覆盖定义结点和使用结点，但覆盖了一些支配结点的测试用例，因此可能遗漏一些较优的个体。

本文提出的基于结点支配关系的适应度函数，不仅从定义结点活跃子路径的角度解决了重定义结点的问题，还考虑了定义结点和使用结点在执行路径中被覆盖的先后顺序问题，有效地提高了测试目标的覆盖率。

在基于搜索方法的测试用例生成研究中，除了遗传算法外，蚁群算法[16]、粒子群优化算法[17]、加速粒子群优化算法[18]、基于遗传算法和粒子群优化算法组合的方法[19]以及人工蜂群算法[20]都被用来解决面向数据流准则的测试用例生成问题。

结束语 本文将面向 all-uses 准则的测试用例生成问题建模为多目标优化问题，提出了一种改进的基于支配关系的适应度函数，在分析测试用例对定义-使用对的覆盖程度中考虑了存在重定义的可能性，且考虑了定义结点和使用结点在执行路径中的先后顺序。基于提出的适应度函数，指导面向多测试目标的遗传算法搜索满足 all-uses 准则的测试用例集。最后通过一组实验分析表明本文方法可以有效地生成满足 all-uses 准则的测试用例，相比其他方法可以有效地提升测试目标的覆盖率，降低生成测试用例所需的迭代次数。

下一步工作包括：1) 将本文方法应用到更大规模的程序对象，检测方法的有效性和性能，并加以完善；2) 针对面向对象程序，考虑方法之间以及类之间的定义-使用对的测试覆盖问题。

参 考 文 献

[1] RAPPS S, WEYUKER E J. Selecting Software Test Data Using Data Flow Information[J]. IEEE Transactions on Software Engineering, 1985, SE-11(4): 367-375.

[2] JI S H, LI B X, ZHANG P C. Test Case Selection for All-Uses Criterion-Based Regression Testing of Composite Service[J]. IEEE Access, 2019, 7: 174438-174464.

[3] AHMED M A, HERMADI I. GA-based Multiple Paths Test Data Generator[J]. Computer & Operations Research, 2008, 35(10): 3107-3124.

[4] GIRGIS M R. Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm[J]. Journal of Universal Computer Science, 2005, 11(6): 898-915.

[5] GHIDUK A S, HARROLD M J, GIRGIS M R. Using Genetic Algorithms to Aid Test-Data Generation for Data-Flow Coverage[C]// 14th Asia-Pacific Software Engineering Conference, 2007: 41-48.

[6] GONG D, ZHANG W, YAO X. Evolutionary Generation of Test Data for Manly Paths Coverage Based on Grouping[J]. The Journal of Systems and Software, 2011, 84(12): 2222-2233.

[7] AHO A V, LAM M S, SETHI R, et al. Compilers: Principles, Techniques, & Tools[M]// New York: Addison-Wesley, 2006: 597-632.

[8] LENGAUER T, TARJAN R E. A Fast Algorithm for Finding Dominators in a Flowgraph[J]. ACM Transactions on Programming Languages and Systems, 1979, 1(1): 121-141.

[9] VARSHNEY S, MEHROTRA M. Search-based Test Data Generator for Data-Flow Dependencies Using Dominance Concepts, Branch Distance and Elitism[J]. Arabian Journal for Science and Engineering, 2016, 41: 853-881.

[10] ANDREOU A S, ECONOMIDES K A, SOFOKLEOUS A A. An Automatic Software Test-data Generation Schema Based on Data Flow Criteria and Genetic Algorithms[C]// Seventh International Conference on Computer and Information Technology, 2007: 867-872.

[11] DENG M J, CHEN R, DU Z J. Automatic Test Data Generation Model by Combining Dataflow Analysis with Genetic Algorithm[C]// Joint Conference on Pervasive Computing, 2009: 429-433.

[12] JAFFARI A, YOO C J, LEE J. Automatic Test Data Generation Using the Activity Diagram and Search-Based Technique[J]. Applied Sciences, 2020, 10(10): 1-21.

[13] VIVANTI M, GORLA A M, FRASER G. Search-based Data-flow Test Generation[C]// IEEE 24th International Symposium on Software Reliability Engineering (ISSRE). 2013: 370-379.

[14] CHEN J Q, JIANG S J, ZHANG Z G. Approach for Test Case Generation Based on Data Flow Criterion[J]. Computer Science, 2017, 44(2): 107-111.

[15] JIANG S, CHEN J, ZHANG Y, QIAN J, WANG R, XUE M. Evolutionary Approach to Generating Test Data for Data Flow Test[J]. IET Software, 2018, 12(4): 318-323.

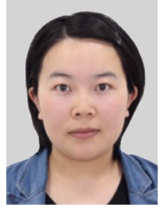
[16] GHIDUK A S. A New Software Data-Flow Testing Approach via Ant Colony Algorithms[J]. Universal Journal of Computer Science and Engineering Technology, 2010, 1(1): 64-72.

[17] NAYAK N, MOHAPATRA D P. Automatic Test Data Generation for Data Flow Testing Using Particle Swarm Optimization[C]// International Conference on Contemporary Computing, 2010: 1-12.

[18] KUMAR S, YADAV D K, KHAN D A. An Accelerating PSO Algorithm Based Test Data Generator for Data-flow Dependencies Using Dominance Concepts[J]. International Journal of System Assurance Engineering and Management, 2017, 8(2): S1534-S1552.

[19] KUMAR S, YADAV D K, KHAN D A. A Novel Approach to Automate Test Data Generation for Data Flow Testing Based on Hybrid Adaptive PSO-GA Algorithm[J]. International Journal of Advanced Intelligence Paradigms, 2018, 9(2/3): 278-312.

[20] SHEORAN S, MITTAL N, GELBUKH A. Artificial Bee Colony Algorithm in Data Flow Testing for Optimal Test Suite Generation[J]. International Journal of System Assurance Engineering and Management, 2020, 11(2): 340-349.



JI Shun-hui, born in 1987, Ph.D, lecturer, is a member of China Computer Federation. Her main research interests include software modeling, analysis, testing and verification.