用遗传算法的测试用例最小化*)

马雪英[1,2] 盛斌奎[2] 叶澄清[1]

(浙江大学计算机学院 杭州 310027)[1] (浙江财经学院信息学院 杭州 310012)[2]

摘 要 回归测试就是验证对程序的修改是否达到了预期的目的,同时检查修改是否损害了程序原有的正常功能。随着程序新版本的不断推出,测试用例集不断扩大,回归测试成本越来越高。测试用例最小化就是从已有的用例集中,找到一个测试运行代价最小的用例子集用于回归测试,并保持原来的测试覆盖率。本文主要研究用遗传算法解决测试用例最小化问题:基于测试历史数据,设计基因编码并构建初始种群;利用测试覆盖率和测试运行代价设计适应度函数;通过遗传算子完成进化过程并找到最优或近似最优解。最后本文给出了对算法进行实例研究的结果。结果表明,本文提出的用例最小化技术能有效缩减回归测试用例集,大幅度降低回归测试费用。

关键词 回归测试,测试用例集,测试用例最小化,测试覆盖率,测试运行代价

Test-Suite Minimization Using Genetic Algorithms

MA Xue-Ying[1,2] SHENG Bin-Kui[2] YE Cheng-Qing[1]

(Dept. of Computer Science, Zhejiang University, Hangzhou 310027)[1]

(Dept. of Information Management, Zhejiang university of Finance & Economics, Hangzhou 310012)[2]

Abstract Regression testing is an expensive process used to revalidate the modified program. As the software is modified and new test cases are added to the test suite, the test suite grows and the cost of regression testing increases. Regression test-suite minimization techniques attempt to reduce the cost of regression testing by identifying a minimized test-suite that provides the same coverage of the software according to some criterion as the original test-suite. This paper investigates the use of an evolutionary approach, called genetic algorithms, for test-suite minimization. The algorithm designs the gene codes of the individuals and builds the initial population based on the test history, calculates the fitness value of each individual using coverage and cost information, and then selectively breeds the successive generations using genetic operations. This generational process is repeated until a minimized test-suite is founded. Finally, some results of studies of this minimization algorithm are presented. The results show that, genetic algorithms can significantly reduce the size and the cost of the test-suite for regression testing, and achieves good cost-effectiveness.

Keywords Regression test, Test suite, Test suite minimization, Test coverage, Test execution cost

1 引言

回归测试是软件测试过程中的一个重要阶段。当代码修改、软件硬件平台变更或硬件配置改变后,都必须重新测试现有的功能,以便确定修改是否达到了预期的目的,检查修改是否损害了原有的正常功能。同时,还需要补充新的测试用例来测试新的或被修改了的功能。在渐进和快速迭代开发中,新版本的连续发布使回归测试进行得更加频繁,测试用例库不断扩大,回归测试成本也随之急剧增加。

可以通过两种途径解决测试用例集膨胀的问题:一是测试用例最小化,二是进行测试选择。测试用例最小化技术(test-suite minimization techniques),也叫测试用例缩减技术,(test-suite reduction techniques)(如文[1,3,7,8,10,12]),就是在原始用例集中,找到一个最小的测试用例子集,并能够提供跟原始测试用例集一样的测试覆盖率;而测试用例选择(test-suite selection)就是从原始用例集中选择出一个用例子集,能够覆盖所有的修改,但这种方法一般不能提供与原始测试用例集相同的测试覆盖(如文[2]、[9]、[11])。

与上述文献中的最小化方法不同,本文研究使用遗传算法进行测试用例最小化,目的是为了有效缩减回归测试用例集,从而降低回归测试费用。

作为一种自适应搜索算法,遗传算法(GAs)被应用于求解许多 NP-complete 问题。虽然它不能保证找到最优解,但是,在给定的时间里,往往能得到比其他算法更优的近似最优解[14]。遗传算法也被广泛地用于软件测试,尤其是在测试用例生成自动化研究上取得了许多进展和成果[19,20,22,24~26]用于结构测试的测试用例自动生成,文[13,21]用于功能测试的测试用例自动生成)。本文对用遗传算法解决测试用例最小化问题进行了研究和探索,设计和实现了基于测试历史的遗传算法并对其做了实例研究,从最小化测试用例集的用例数、测试运行代价、最小化时间三个方面对算法的性能进行了考察和评价。

本文的以下章节是这样组织的:第 2 节简要介绍与本课题有关的一些相关工作;第 3 节是本文主要的研究内容,给出了用于最小化的遗传算法的详细描述。第 4 节给出了实例研究的结果。最后对本文研究工作做总结和展望。

2 相关工作

2.1 测试用例最小化

测试用例最小化问题可以简单的描述为:给出一个测试用例集 $TS=\{t_1,t_2,\cdots,t_n\}$,每个测试用例 t_i 都有相应的测试代价 $c_i(c_i>0)$,找到一个 TS 的真子集 T,满足:T 的测试代

*)国家自然科学基金资助项目(60073027)、浙江省教育厅科研基金(119034031)、浙江财经学院二〇〇五年度重大科研课题(YJZ0505)。

价 Cost(T)为最小，并且 T 的测试覆盖度 Cov(T)等于 TS 的测试覆盖度 Cov(TS)。这是一个 NP-complete 问题。我们把 TS 称为原始用例集，称 T 为最小化用例集。

实现测试用例最小化，必须具备以下基础：

测试用例库(testing pool)：保存了一个程序在各个生命周期使用过的所有测试用例。

测试覆盖信息(test coverage)：记录了每个测试用例测试了程序的哪些部分(语句或片段)。

测试运行代价信息(test execution cost)：记录了每个测试用例在测试时需要的代价(本文主要考虑运行测试用例的CPU时间，暂时不考虑建立测试环境的代价以及人力代价)。

最小化算法根据用例库中覆盖信息和测试运行代价信息评价用例，剔除冗余用例，降低回归测试费用。

2.2 测试用例最小化技术

可以用以下4个属性评价最小化技术：

第一是充分性(Adequacy)：最小化测试用例集必须能够保持与原始用例集相同的测试覆盖度。

第二是精确性(be precise)：能最大限度剔除冗余用例，缩减用例集大小。

第三是效益(Cost-effectiveness)：用于最小化的费用(即运行最小化算法得到最小化用例集的费用)应该小于由于使用最小化用例集进行回归测试缩减下来的费用，即要求算法花费合理的代价(主要考虑时间效率)得到最小用例集合。

第四是算法的通用性(generality)：使用于不同程序、不同的测试覆盖标准等。

研究测试用例最小化技术，充分性是前提，精确性和效益是关键，通用性是意义。在使用以上评价体系时，我们基于以下假设：测试用例的测试运行代价是一个常量，跟是否使用测试用例缩减技术无关。

2.3 测试历史

为了提高软件测试效率，降低软件测试费用，我们研究开发了面向程序设计语言(如C、C++、Visual Basic)的软件测试自动化工具 Panaroma Tester。该工具实现基于路径覆盖的结构测试，能在单元测试和集成测试的级别上工作。该测试工具提出了一种新的基于块的划分机制，在此基础上，扩展了基于块的覆盖度量标准：SC0、SC1、SC1+和 J-Coverage(详见文[17])。我们把程序理解为块(Block)的序列。块的定义是：可以被看作为一个单元的一组连续的程序语句，或者说，总是在一起执行的最大的程序语句序列。

块有两类：节点(Node)和段(Segment)。节点包括判断(Decision)、连接(Junction)、程序单元的入口点和出口点。段分为可见段和不可见段。

为了提高模块的重用性和增加测试工具的通用性，该测试工具采用了以测试引擎为核心的体系结构(详见文[18])。测试引擎的一个重要部分就是测试历史数据库，用来存放测试历史信息包括程序的静态分析信息和动态测试跟踪信息。

静态分析信息包括：源文件的结构信息、类的定义信息、方法的定义信息、源文件的块划分信息。块划分信息包括：块的类型、块所对应的记录点的序号、块在源程序里的起始行号等信息，每个源文件的所有块用链表依次串成一条链，对应于整个程序。

动态测试跟踪信息只收集与被测软件运行时相关的信息，如程序中各个块、条件等在每次测试中的执行次数；程序中的每个条件、判断在测试执行过程中的取值(true/false)；每个测试用例测试了哪些段、方法、类以及运行时间等。一个简化的基于块的测试覆盖信息如下表1所示。

表1 基于块的测试覆盖信息

	Block 1	Block 2	…	Block k	…
Case 1	1	0	…	1	…
Case 2	0	0	…	1	…
…	…		…		…
Case	0	1	…	0	…
Case n	1	1	…	0	…
…	…	…	…	…	…

表1中，block 1，block 2，…，block k 是被测程序的块的序列，case 1，case 2，…，case n 是 n 个测试用例。表中全部用'0'或'1'数字表示，比如第 i 行第 j 列中的数字如果等于'1'，则表示用例 i 测试覆盖了块 j；否则，如果等于'0'，则表示用例 i 测试没有测试块 j。这样，表中第 i 行中'1'的个数，就表示了用例 case i 运行时测试到的块的个数，即它的覆盖度；同理，表中第 j 列中'1'的个数，则表示在一次测试中，覆盖了块 block j 的一个测试用例子集的用例个数。

3 测试用例最小化算法

3.1 遗传算法

遗传算法(Genetic Algorithm，简称 GA)是由进化论和遗传学机理而产生的一种优化方法。

遗传算法中，种群(population)是个体的集合，个体(individual)是指染色体带有特征的实体，而染色体是遗传物质的主要载体，由多个遗传因子即基因(gene)组成。在最优化问题中，包含个体 x_i 的种群 $P=\langle x_1,\cdots,x_n\rangle$，个体 x_i 代表问题的一个解，群体就是问题的一些解的集合。评价函数 $F(x_i)$，也称作适应度函数，用以评价群体中每个个体的适应度(fitness)，目标是优化该评价函数(搜索该函数的最大值或最小值)以解决给定的问题。

在完成了基因编码以及适应度函数的设计后，遗传算法使用三个遗传算子：选择(再生)、交叉、变异，完成进化过程，找到最优解。

3.2 用于测试用例最小化的遗传算法

下面我们详细介绍用于测试用例最小化的遗传算法模型，包括基因编码设计、适应度函数设计以及上述3个遗传算子等。

基因编码 一般地，在解决优化问题时，遗传算法的个体代表了问题的一个解。对于测试最小化问题，其解是原测试用例集的一个子集，而最优解是其中的一个解，它的测试运行代价最小，但提供了与原测试用例集一样的覆盖率。因此，初始种群的个体基因信息可以从测试历史信息获取。如表1所示的块执行历史图中，第 j 列中'1'的个数，代表了用例集中测试了块 block j 的用例的个数，而这些'1'所对应的用例组成的集合，就是这样一个用例子集：它们在上一次测试中同时覆盖了块 block j。这样，表1中有 k 个列，就表示了 k 个子集，我们把这 k 个用例子集构建成遗传算法的初始种群。如果原测试用例集共有 n 个用例，那么个体的基因编码可以表示成以下方式：

$G_j=[g_{j1},g_{j2},\cdots,g_{ji},\cdots,g_{jn}]$，$g_{ji}\in\{0,1\}$，$1\leqslant j\leqslant k$，$1\leqslant i\leqslant n$

如果 $g_{ji}=1$ 说明用例 t_i 测试执行了块 b_j，也就是 t_i 包含

在 G_j 所表示的测试子集中，我们记为 T_j。否则如果 $g_{ji}=0$ 则说明用例 t_i 没有执行 t_i，即 t_i 不包含在 G_j 所表示的测试子集中 T_j。T_j 中包含的用例的个数设为 $|T_j|$。

适应度函数 适应度函数在测试用例选择问题中，跟测试用例的覆盖度、测试代价以及相关块的权重有关。关于个体 T_j 的适应度按以下公式计算

$$F(T_j)=\mathrm{Cov}(T_j)/\mathrm{Cost}(T_j) \quad (1)$$

这里 $\mathrm{Cov}(T_j)$ 是用例集 T_j 的测试覆盖度，$\mathrm{Cost}(T_j)$ 是用例集 T_j 的测试运行代价。$\mathrm{Cov}(T_j)$ 可以按下式计算：

$$\mathrm{Cov}(T_j)=\sum_{t_i\in T_j}\mathrm{Cov}(t_i) \quad (2)$$

而根据表 1 中的信息，测试用例的覆盖信息由表中的第 i 行给出，第 i 行中有几个'1'，就代表测试用例 t_i 测试了程序中多少个块，这些'1'所在的列所对应的块就是用例 t_i 所测试的程序的块。设向量 $X_i=[x_{i1},x_{i2},\cdots,x_{ij},\cdots,x_{ik}]$ 表示了表 1 中第 i 行的信息，向量 $W=[w_1,w_2,\cdots,w_i,\cdots,w_k]$ 表示了每个块的权重。这样，用例 t_i 的覆盖度可以按如下计算：

$$\mathrm{Cov}(t_i)=\sum_{j=1\sim k}(x_{ij}\times w_j),\ 1\leqslant j\leqslant k \quad (3)$$

但是由于用例之间的覆盖度有交叉，即几个用例可能测试了某些相同的块，因此计算 T_j 的覆盖度的时候，不能直接按式(3)进行，必须消除重复计算。计算用例子集 T_j 覆盖度的 C++程序如图 1 所示。

同样，$\mathrm{Cost}(T_j)$ 按式(4)计算：

```
Double Population::Cov(int j)
{ int *covnum;
  double Cov;
  Cov=0;
  covnum=new int[sizePop];
  for (int j=0;j<sizePop;j++) covnum[j]=0
  for(int i=0;i<sizeCase;i++)
      { if G[j]. code[i]>0 /* G[j] is the individual Gj*/
      for (j=0;j<sizePop;j++)
         {
         covnum[j]+=X[i][j];
            /*X[i] is the row vector Xi*/
         }
      }
  for (j=0;j<sizePop;j++)
     {if (covnum[j] >0)
       Cov+=w[j]; /* w[j]is the weight of block j*/
     }
  Cov=Cov*100/sizePop;
  delete []covnum;
  return Cov;
}
```

图 1　计算用例集 T_j 覆盖度的 C++程序

$$\mathrm{Cost}(T_j)=\sum_{t_i\in T_j}\mathrm{Cost}(t_i) \quad (4)$$

从以上计算方法可以看出，F 值越高，就表示该用例子集测试覆盖度越大而测试运行代价越小。下面我们定义遗传算子。

选择(Selection)　选择过程是一种基于适应度的优胜劣汰的过程。在选择过程中，适应度高的个体被直接复制到下一代群体中。适应度越高的串，产生后代的概率就越高。在交叉过程中，两个串的部分位(称为基因)进行交换从而产生一个新串作为下一代的个体。变异用来随机地改变染色体的部分基因。交叉和变异的使用都有一定的概率，分别称为交叉概率和变异概率。这里我们采用轮盘赌的选择(roulette wheel selection)策略。

交叉(Crossover)　我们采用单点交叉。设 m 是种群中个体的尺寸大小，在两个个体的同一位置处进行交叉重组，形成两个新的个体。图 2 中所示两个个体 ind_1 和 ind_2，经过交叉形成新的个体 ind_3 和 ind_4。在所实现的算法中，我们设交叉率为 0.8。

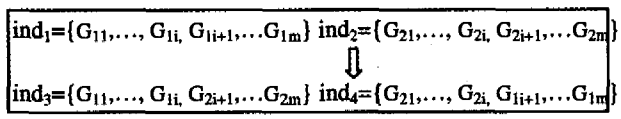

图 2　基因交叉

变异(Mutation)　交叉之后子代经历的变异，实际上是子代基因按小概率扰动产生的变化。基于本问题中基因模型，我们采用二进制变异算法，实现基因码的小概率翻转来达到变异的目的，如下所示：$G_j=[g_{j1},g_{j2},\cdots,g_{ji},\cdots,g_{jm}]\rightarrow Gmut=[g_{j1},g_{j2},\cdots,\boxed{g_{ji}},\cdots,g_{jm}]$。这里，如果 $g_{ji}=1$，那么 $\boxed{g_{ji}}=0$；否则，如果 $g_{ji}=0$ 那么 $\boxed{g_{ji}}=1$。在所实现的算法中，我们设变异率为 0.01。

当决定了基因编码以及以上几个重要的遗传算子，我们就可以如图 3 所述，进行测试用例的选择。

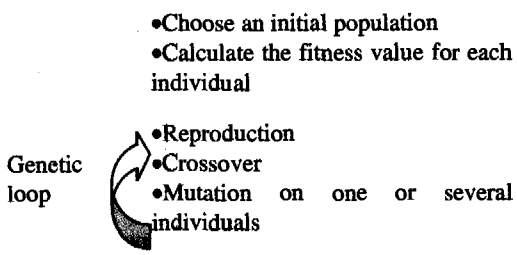

图 3　一种遗传算法

4　实例研究

我们用 C++开发了该测试用例最小化算法的原型(名为 GeA)，并且对两个实际的 VB 程序进行了实例研究：一个是计算器(CALCULATOR)模拟程序，另一个是电视放映模拟程序(LITTLE-TV)。相应的测试用例集是由我们软件测试研究小组的研究人员在研究软件测试工具时，精心设计的。CALCULATOR 程序由 85 块组成，LITTLE-TV 由 241 个块组成，它们的测试历史信息是测试时生成并记录在测试数据库中。CALCULATOR 程序的测试用例库由 174 个用例组成，而 LITTLE-TV 程序的测试用例库包含 238 个测试用例。原始用例集的测试用例都是从用例库随机抽取，CALCULATOR 程序的原始用例集用例数 5 到 80 个，每种用例数的原始用例集都准备 5 组，覆盖率在 65%到 95%之间，平均覆盖率为 90%，测试运行时间最小是 145s，最多是 2398s；同样，LITTLE-TV 程序的原始用例集用例数为 5 到 85 个，每种用例数的用例集也是 5 组，覆盖率在 60%到 98%之间，平均覆盖率为 89%，测试运行时间最少为 87s，最多为 1984s。

为评价算法的精确性和效益，我们从三个方面对最小化算法进行评价：最小化用例集的大小，最小化用例集在测试运行代价(主要是指运行时间)上的降低幅度来评价以及最小化算法的运行时间。

首先，我们将最小化用例集包含的用例数设为原始用例集用例数的函数。实验结果在图 4 和图 5 给出。由于相同用

例数的原始用例集平均有 5 组，所以图中的最小化用例集的用例数是 5 组同样大小的原始用例集最小化后的用例数的平均值。对于 CALCULATOR 程序，GeA 算法得到的最小化用例集用例数平均为 6.49，标准偏差为 1.3；对于 LITTLE-TV 程序，GeA 算法得到的最小化用例集用例数平均为 6.46，标准偏差为 1.2。从结果可以看出，算法能有效缩减用例集的用例数，且用例数稳定。

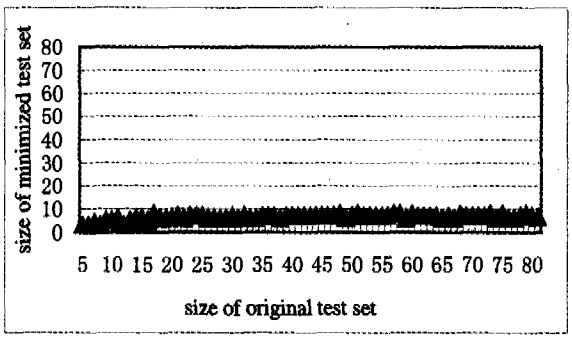

图 4　最小化测试用例数与原始测试用例数的关系

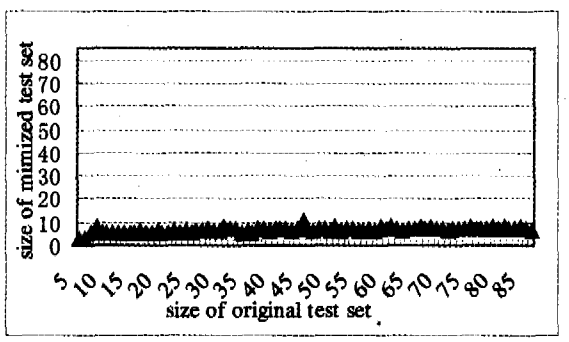

图 5　最小化测试用例数与 LITTLE-TV 原始测试用例数的关系

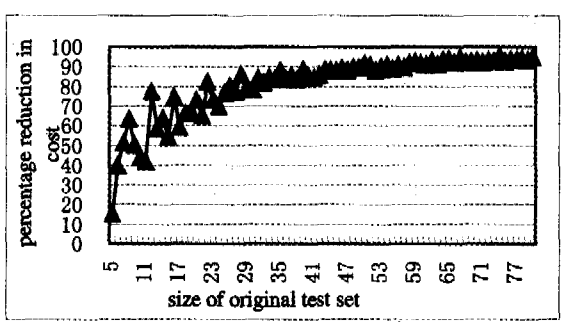

图 6　CALCULATOR 程序运行代价百分比之减少

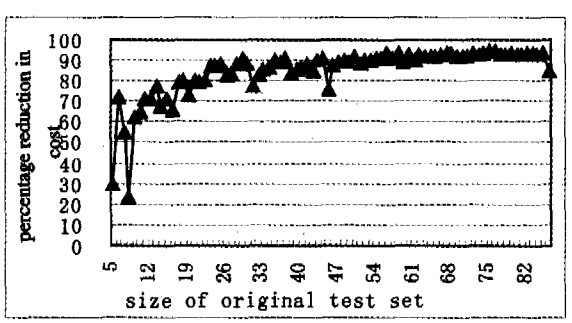

图 7　LITTLE-TV 程序运行代价百分比之减少

然后，我们将测试用例最小化后减少的测试运行代价（时间）占原测试运行时间的百分比，设为原始用例集大小的一个函数，实验结果由图 6 和图 7 给出，同上所述，图中的百分比也是平均值。对于 CALCULATOR，测试运行代价比原始用例集平均降低了 81.09%，标准偏差为 16.25，降低最多是 95.43%，最小是 11.03%；对于 LITTLE-TV，运行代价比原始用例集平均降低了 84.76%，标准偏差为 11.03，降低最多是 94.85%，最小是 23.75%。从减少的测试运行时间来看，对于 CALCULATOR，测试运行时间减少的幅度是 16 ～ 2262s；而对于 LITTLE-TV，测试运行时间减少的幅度是 16～ 1857s。

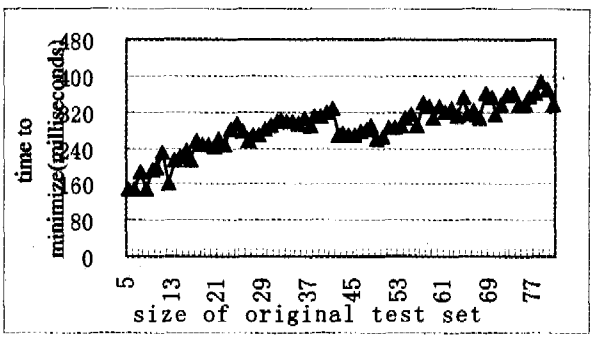

图 8　测试用例最小化的时间（CALCULATOR 程序）

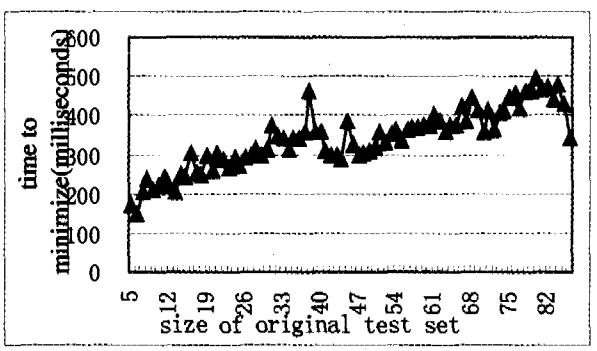

图 9　LITTLE-TV 程序的情况

最后，我们对 GeA 算法得到最小化用例集的运行时间进行了考察，结果由图 8 和图 9 给出。图中 Y 轴表示算法对用例集进行最小化的运行时间，单位是毫秒。对于 CALCULATOR 程序，GeA 算法得到的最小化用例集的运行时间为 148～401ms；对于 LITTLE-TV 程序，GeA 算法得到的最小化用例集的运行时间为 136 ～496ms。

从前面的实验结果，我们可以看出，得到最小化用例集的算法运行时间，远小于测试用例集因最小化而减少的运行时间，具有很好的效益。另外，我们在文[18]中可知，Panaroma Tester 中的测试引擎是该工具具备了通用性，即不同语言（C++，VB 等）通过测试生成的历史数据以及对程序的回归测试的测试历史数据，都以统一的格式存储在测试历史数据库中，所以，基于测试历史设计的最小化算法 GeA 能够对多种语言和修改的测试用例集进行最小化。而且，根据覆盖度量标准，我们只要通过修改 Cov() 函数的计算方法，就能方便地扩展算法以支持不同覆盖标准的测试用例最小化。因此我们的测试用例最小化算法具有很好的通用性。

结论　本文提出了用遗传算法解决测试用例最小化问题，并开发了算法原型，在此基础上进行了实例研究。研究结果表明，我们的测试用例最小化技术能显著减少测试用例集中的用例数和降低测试运行代价，达到了较好的效率和效益，

（下转封面）

(上接第 288 页)

并具备通用性。

虽然,初步的研究结果肯定了我们研究工作的意义,但必须从更多的实际测试最小化应用中,得到更多更充分的实验数据,从而证明我们的最小化技术的实用性和通用性。另外,对于将遗传算法用于测试用例最小化,我们只是做了初步的尝试和实践,还没考虑优化、与其它算法比较等工作。

在未来的工作中,我们将继续研究最小化算法的通用性和实用性,以及进一步深化对遗传算法用于解决测试用例最小化问题的研究,进一步提高最小化效率。

参 考 文 献

1 Chen T Y, Lau M F. Dividing Strategies for the Optimization of a Test Suite [J]. Information Processing Letters, 1996, 60(3): 135~141

2 Chen Y, Rosenblum D, Vo K. TestTube: A System for Selective Regression Testing [A]. In: Proc. 16th Int'l Conf Software Eng, May 1994. 211~222

3 Jones J A, Harrold M J. Test-Suite Reduction and Prioritization for Modified Condition/Decesion Coverage [J]. IEEE Trans on Software Engineering, Mar. 2003, 29(3): 195~209

4 Onoma K, Tsai W-T, Poonawala M, et al. Regression Testing in an Industrial Environment [J]. Communications of the ACM, May 1988, 41(5): 81~86

5 Bäck T. Optimal mutation rates in genetic search [A]. In: Proceedings of the 5th International Conference on Genetic Algorithms (ICGA(93), Morgan Kaufmann. 2~9

6 Baudry B, Fleurey F, Jézéquel J M, et al. Genes and bacteria for automatic test cases optimization in the . net environment [A]. In: Proceedings of ISSRE02 (International Symposium on Software Reliability Engineering), Annapolis, USA, November 2002. 195~206

7 Harrold M J, Gupta R, Soffa M L. A Methodlogy for Controlling the Size of a Test Suite [J]. ACM Trans. Software Eng and Methods, 1993, 2(3): 270~285

8 Offutt J, Pan J, Voas J M. Procedures for Reducing the size of Coverage-Based Test Sets [A]. In: Proc 12th int'l Conf Testing Computer Software. June 1995. 111~123

9 Rothermel G, Harrold M J. A Safe, Efficient Regression Test Selection Techinique [J]. ACM Trans Software Eng and Methods, Apr 1997, 6(2): 173~210

10 Rothermel G, Harrold M J, Ostria J, et al. An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites [A]. In: Proc. Int'l Conf Software Maintenance, Nov. 1998. 34~43

11 Rothermel G, Harrol M J. Selecting Tests and Identifying Test Coverage Requirement for Modified Software [A]. In: Proceedings of the 1994 international symposium on Software testing and analysis, Seattle, Washington, United States, 1994. 169~184

12 Wong W E, Horgan J R, London S, et al. Effect of Test Set Minimization on Fault Detection Effectiveness [J]. Software Practice and Experience, Apr 1998, 28(4): 347~369

13 Berndt D, Fisher J, Johnson L, et al. Breeding Software Test Cases with Genetic Algorithms [A]. Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS 36), 2003

14 Goldberg D E. Genetic Algorithms in Search. Optimization and Machine Learning [M]. Addison-Wesley, 1989

15 Holland JH. Adaptation in Natural and Artificial Systems [M]. Ann Arbor, MI: University of Michigan Press, 1975

16 Rosenblum D S, Weyuker E J. Using Coverage Information to Predict the Cost-Effectiveness of Regression Testing Strategies [J]. IEEE Transaction on Software Eng, 1997, 23(3): 146~156

17 杨建军,陈卫东,叶澄清,等. 面向上下文无关语言的测试工具的设计和实现[J]. 计算机研究和发展,2000,37(11):1375~1382

18 马雪英,姚砺,叶澄清. 面向对象软件测试引擎的设计和实现[J]. 计算机科学,2004,31(7):137~140

19 Roper M. CAST with GAs (Genetic Algorithms) - Automatic Test Data Generation via. Evolutionary Computation. IEE Colloquium on Computer Aided Software Testing Tools, digest no. 96/096, April 1996

20 Pargas R, Harrold M, Peck R. Test data generation using genetic algorithms. Software Testing [J]. Verification & Reliability, 1999, 9(4): 263~282

21 Michael C, McGraw G, Schatz M. Generating Software Test Data by Evolution [J]. IEEE Transactions on Software Engineering, 2001, 27(12): 1085~1110

22 Wegener J, Baresel A, Sthamer H. Evolutionary Test Environment for Automatic Structural Testing [J]. Information and Software Technology, Special Issue devoted to the Application of Metaheuristic Algorithms to Problems in Software Engineering, 2001, 43(spl): 841~854

23 Tracey N, Clark J, Mander K. Automated Program Flaw Finding Using Simulated Annealing [A]. In: Proc. Int'l SympSoftware Testing and Analysis, Software Eng. Notes, 1998. 73~81

24 Watkins A. The Automatic Generation of Software. Test Data using Genetic Algorithms [A]. In: Proceedings of the. Fourth Software Quality Conference, Dundee,. Scotland, July, 1995, 2: 300~309

25 Borgelt K. Software Test Data Generation From a Genetic Algorithm. Industrial Applications of Genetic Algorithms, CRC Press, 1998

26 Lin JC, Yeh PU. Automatic Test Data. Generation for Path Testing using GAs[J]. Information. Sciences, 2001, 131: 47~64

计算机科学

(1974 年 1 月创刊)
第 34 卷第 01 期 (月刊)
2007 年 1 月 25 日出版

国际标准连续出版物号 ISSN 1002-137X
国内统一连续物出版号 CN50—1075/TP

定价: 30.00 元 国外定价: 5 美元
邮发代号: 78—68
发行范围: 国内外公开

主管单位: 国家科学技术部
主办单位: 国家科技部西南信息中心
编辑出版: 《计算机科学》杂志社
重庆市渝中区胜利路 132 号 邮政编码: 400013
电话: (023) 63500828 E-mail: jsjkx@swic.ac.cn
网址: www.jsjkx.com
社 长: 牟炳林
总 编: 彭 丹
主 编: 朱宗元
主编助理: 徐书令
印 刷 者: 重庆科情印务有限公司
总发行处: 重庆市邮政局
订 购 处: 全国各地邮政局
国外总发行: 中国国际图书贸易总公司 (北京 399 信箱)
国外代号: 6210—MO