Classified Index: TP311

Dissertation for the Master Degree in Engineering

Research on Automatic Software Test Case Generation Based on Genetic Algorithms

Candidate: Liu Shuangyue

Supervisor: Wang Peidong

Academic Degree Applied for: Master of Engineering

Speciality: Computer Applied Technology

Date of Oral Examination: March, 2010

University: Harbin University of Science and Technology

哈尔滨理工大学硕士学位论文原创性声明

本人郑重声明：此处所提交的硕士学位论文《基于遗传算法的软件测试用例自动生成技术研究》，是本人在导师指导下，在哈尔滨理工大学攻读硕士学位期间独立进行研究工作所取得的成果。据本人所知，论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文研究工作做出贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签名：刘双悦　　日期：2010年4月6日

哈尔滨理工大学硕士学位论文使用授权书

《基于遗传算法的软件测试用例自动生成技术研究》系本人在哈尔滨理工大学攻读硕士学位期间在导师指导下完成的硕士学位论文。本论文的研究成果归哈尔滨理工大学所有，本论文的研究内容不得以其它单位的名义发表。本人完全了解哈尔滨理工大学关于保存、使用学位论文的规定，同意学校保留并向有关部门提交论文和电子版本，允许论文被查阅和借阅。本人授权哈尔滨理工大学可以采用影印、缩印或其他复制手段保存论文，可以公布论文的全部或部分内容。

本学位论文属于

保密，□在　年解密后适用授权书。

不保密 ☑。

（请在以上相应方框内打√）

作者签名：刘双悦　　日期：2010年4月6日

导师签名：[illegible]　　日期：2010年4月6日

基于遗传算法的软件测试用例自动生成技术研究

摘　要

软件测试已经变得比以往任何时候都复杂和困难。软件测试作为保证软件质量和可靠性的重要手段已经成为国内外软件行业研究的重点方向之一。

研究测试用例的自动生成，可以降低手工测试的高额成本，将测试人员从繁重的劳动中解脱出来，同时提高测试过程的可信赖程度。测试用例自动生成方法的研究，对促进软件测试过程自动化程度，有着重要的现实意义。

本文主要针对测试用例自动生成这一问题进行深入的研究和设计。首先，介绍软件测试的基本理论，分析比较测试用例自动生成的方法：随机法、符号执行法、程序直接执行法和遗传算法，最终确定使用遗传算法作为实现路径测试用例自动生成的核心算法。

随后，基于对遗传算法及其适应度函数的特点进行分析和研究，重点介绍面向路径测试的适应度函数计算方法，对适应度函数性能进行评估。在适应度函数及适应值选择策略方面提出相应的改进。新的选择策略把群体分为若干组，以组为单位进行轮盘赌选择，在选中的组中，由该组内的个体综合作用产生新的个体。

最后使用三角形分类程序作为例子，生成该程序的测试数据，针对四种常用的适应度函数和两种改进的适应度函数进行评估验证，并对实验数据进行分析。实验结果表明，能够产生较少的测试用例覆盖给定路径的方法是分支谓词方法和逆路径概率方法，在基于路径测试的遗传算法中，使用这两种适应度函数计算方法是最有效的。

关键词 软件测试；路径测试；遗传算法；适应度函数

Research on Automatic Software Test Case Generation Based on Genetic Algorithms

Abstract

Software testing has become more complex and difficult than ever. Being an important measure to assure quality and reliability of software,software testing becomes one of the most important aspects in domestic and abroad software researches.

Studies of automatic test case generation can reduce the high cost of manual software testing,free tester from heavy labor and at the same time increase its reliability.So studies of automatic test case generation is of great significance to increase the realization of automatic software testing.

The paper takes the research and design for the automated generation of test cases deeply.First,this paper introduces the basic theory of software testing, The genetic algorithm is used as the core algorithm of automatic test case generation based on analysis and comparison of the methods such as random algorithm,symbol executing algorithm, target oriented algorithm,path oriented algorithm and genetic algorithm.

Second,based on the study of basic principle of genetic algorithms,this paper focuses on several fitness function for path testing,accesses the performance of fitness functions and improves the fitness function and selection strategy of fitness value. New selection strategy divides population into several groups,uses roulette options in each group unit and generate new individual by the combined effect of individual in the selected group.

At last,as an example,we generate testing case for the Program of Triangle Classifier. The results demonstrate that some fitness functions provide better results than others,generating fewer test cases to exercise a given program path.In these studies,the branch predicate and inverse path probability approaches were the best performers.

Keywords software test , path test , genetic algorithms , fitness function

目 录

第1章 绪论

1.1 研究背景

尽管软件开发技术在过去的几十年来有着显著的提高，软件开发仍然依靠开发人员沉重的劳动。由于人类不能够完美的交流和工作，错误往往会发生，结果导致软件缺陷。软件中的错误仍旧是昂贵的，棘手的问题，每年全球组织大约要开销数十亿美元。美国政府标准技术研究院(NIST)的一项研究发现，美国用户仅就此项每年遭受的经济损失超过 595 亿美圆。更重要的是，NIST's 发现从软件开发者的角度，大约三分之一的损失可以通过前期的测试得到避免。

1.1.1 路径测试的研究意义

软件测试存在于软件的整个生命周期，确保软件的质量，让软件的安全性得到提高是关系全局，十分困难的课题。测试要经过需求分析阶段和开发阶段，开发阶段包括单元测试、集成测试、确认测试、系统测试，如图 1-1 表示软件开发阶段的四个过程。

软件开发阶段错误发现的越早越好，因为在初期的修改涉及的面相对少，所受的经济损失小。但在开发中期的时候，软件配置的大多成份已经完成，某一模块的变动就要对所有已完成的配置成分都做相应的改动，工作量大，而且逻辑上的修改变得越发复杂，因此付出的代价急速变大；如果软件完成时再引入变动，就必需要付出比中期还要高的代价。

软件测试工作量大，实现穷举测试是不可能的，为了节省时间和资源，必须精心编写测试用例，在数目庞大的测试用例集合中精心选取适量的测试数据，对程序单元中关键的执行路径加以测试，以达到某种覆盖率要求的测试效果，保障软件的质量。这种路径测试问题的研究具有了很重要的意义。

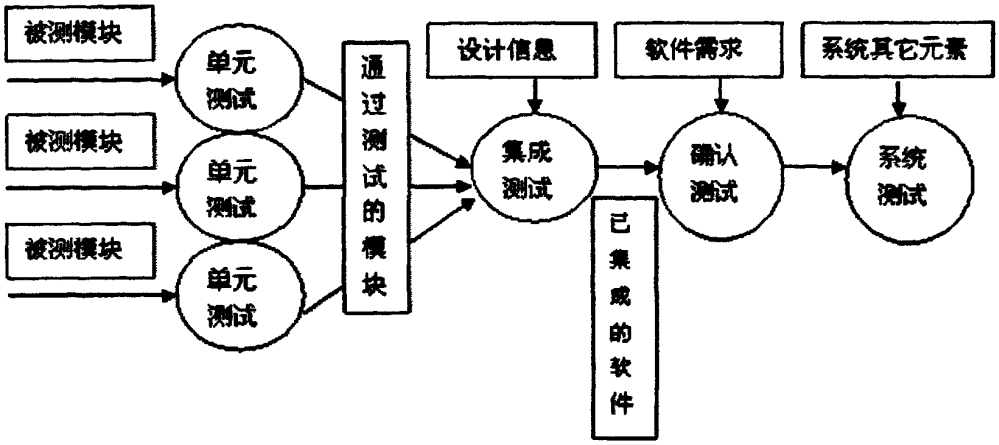


图1-1 软件测试的过程

Fig. 1-1 Process of software testing

1.1.2 软件测试用例自动生成技术的研究意义

针对软件加以测试需要使用不同的软件测试策略，选取任何一种测试策略，测试用例生成都是测试阶段最重要的技术难点。

当前的测试人员通常是依据工程经验，使用向前核查法和逆向回溯法手工为给定的程序路径生成测试数据。向前核查法与逆向回溯法不能够覆盖到一些复杂路径，这使其不能胜任复杂路径的测试。软件测试过程繁琐，需要开销很多的人力、物力与时间以实现测试用例的生成工作。导致测试成本居高不下，测试效率无法提高，质量安全得不到保障。测试用例自动生成技术可以避免开发人员对部分数据抱有的成见，改变了以往测试人员根据经验测试的传统做法，大大提高的软件的测试效率和软件质量。因此测试用例自动生成方法的研究，对实现软件测试过程自动化，有着十分重要的现实意义[1]。

1.2 国内外研究现状

世界各国的专家在测试用例的自动生成技术方面做了深入的探索，提出了许多的策略。主要包括基于规格说明的测试数据生成方法与基于程序结构的测试数据生成方法。

文献[2]提出的从关系代数查询表示的规格说明中自动生成测试数据的方法，还有文献[3]提出的从布尔规格说明自动生成测试数据的方法，它根据软件规范设计测试数据，因而正确性决定于规范说明的正确性。但实际上我们不能确定规范说明一定正确。所以，依照被测程序的逻辑构成选择测试数据是十分重要的。

D.Bird[4]等使用随机法自动生成测试数据，能够无约束的快速随机生成很多的测试数据。这是一个巨大的测试数据集合，例如随机法要使三角形分类程序达到分支覆盖标准，需产生二十多万组测试数据。C.V.Ramanmoorthy等人[5]，R.A.DeVlillo等人[6]，L Clarke等人[7]都对符号执行法进行了研究。被测程序并没有真正运行，只是按照执行的顺序将对应的变量用符号表达式替换。因此随着程序规模的越来越大，符号表达式变得越来越长，几乎无法求解。该方法的可用性也就会变差。

研究有关程序执行的测试数据生成方法比较多。Bogdan Korel[8]提出的是采用步进的方式执行程序，一次只前进一个分支谓词。并且提出了谓词函数的概念，用来度量分支谓词的接近满足程度。Bogdan Korel用该方法为Pascal语言的结构化子集设计的程序生成测试数据。奚红宇等人将该方法用于Ada软件的测试数据生成[9]。

M.Gallagher等人[10]采用对程序插桩强制程序用输入空间D中随机一组数据为入口来执行路径w，且用插桩语句向测试数据生成器返回每个变量与分支谓词的情况。

Neelam Gupta[11]等人提出的迭代松弛法使用程序谓词切片的思想，在输入域中随机选取一组输入数据，对路径上的各谓词分支，明确它的谓词切片与输入依赖集，推算出每个谓词函数有关入口参数的线性约束，构建入口参数的增量的线性方程系统，取高斯消元法求解后得到各入口参数的增量，进而取得一组新的输入。如果路径上每个谓词分支全是入口参数的线性函数，此方法要么经过一次迭代找到解，要么确保路径不可行。若谓词函数里包含入口参数的非线性函数时，由于此方法是将线性函数用作非线性函数的近似，因此若想找到解需循环多次，所以此方法针对非线性路径约束是不完备的。GuPta等人在研究分支覆盖的测试数据自动生成中采用了此思想。

单锦辉博士把迭代松弛法改进后，把它应用到测试用例的自动生成[12]。改进了Gupta方法，去掉该方法里构造谓词切片与输入依赖集的过程，直接计算各谓词函数的线性算术表示，构建输入变量的线性方程系统，求解后得到一组新的输入。

近些年，国内外的研究人员把遗传算法应用到面向路径的测试数据自动生成。Borgelt[13]和Lin[14]等人将遗传算法用于面向路径的测试。B.F.Jones [15] 等人和Joachim Wegener [16] 等人的实验结果发现，基于遗传算法的测试数据生成在三角形分类等程序中所生成的数据比随机法要低一到两个数量级。汪浩等人给出了遗传算法的形式化的表示并基于该方法的系统建立了模型 [17]；荚伟等人将遗传算法应用于基于路径覆盖的Ada软件结构测试数据的自动生成，并得到遗传算法比爬山法和随机法生成测试数据的效率高的结论[18]。此外，景志远[19]从数学的角度分析了将MGA和遗传K均值等改进的算法应用于测试用例的自动生成。傅博[20]提出了基于模拟退火遗传算法的测试数据自动生成算法， Praveen Ranjan[21]和Jose Carlos[22]等人提出用遗传算法只产生可行测试数据的方法。

1.3 本文的主要内容

本文首先分析了路径测试数据自动生成系统的3个部分，重点研究测试数据自动生成方法。并在此基础之上提出一种基于遗传算法的面向路径的测试数据自动生成算法并解决了哪种适应度函数有利于面向路径测试的问题。本论文主要的目标和主要工作如下：

(1) 阅读大量的文献资料，深入研究软件测试问题，提出目前软件测试迫切需要解决的问题是测试费用高昂，而测试费用大多用在测试用例的设计上。因此改善自动化测试中软件测试用例自动生成技术是提升测试效率、减少测试费用的有效途径。

(2) 研究分析几种常见的面向路径适应度函数计算方法，并对适应度函数进行改进。

(3) 学习遗传算法，针对测试数据自动生成系统的特点，提出以遗传算法为核心算法的面向路径的测试用例自动生成系统的算法GenerateData。

(4) 将算法用于面向路径的测试用例的自动生成，改进适应度的选择策略，以评估适应度函数的性能。针对四种适应度函数和两种改进的适应度函数进行性能评估，哪种适应度函数的性能最好即能够以较少的测试用例覆盖给定路径。

针对目前遗传算法应用于路径测试中使用的四种常见的适应度函数和两种改进的适应度函数进行分析评估，在分析了遗传算法的特点之后，本文提出了基于遗传算法的路径测试数据自动生成算法。并使用三角形分类程序做为被测程序，演示该程序的测试用例生成的过程，并对实验数据进行分析。

本文共包含5章，每章主要内容如下：

第1章，简单介绍了本课题的研究背景，国内外研究现状以及本课题的研究内容。

第2章，介绍归纳软件测试的理论，并对本文自动生成测试用例关键问题进行了阐述。详细介绍测试数据生成系统的三个组成部分，并给出每个模块阶段的工作所使用的关键方法。

第3章，介绍遗传算法及适应度函数的构造，指出遗传算法的缺陷及改进方向，在详细分析了四种面向路径的适应度函数计算方法后，提出两种改进的适应度函数。提出对适应度的性能进行评估。

第4章，提出以遗传算法为核心算法的测试数据自动生成算法GenerateData，详细介绍该算法的设计实现及主要特点。

第5章，针对实际的被测程序TRITYP以及第三章介绍的六种适应度函数，使用本文提出的方法为指定的路径生成测试数据，针对不同适应度函数实验结果进行比较和分析。

最后，对本课题的工作进行总结。

第2章 软件测试及测试数据自动生成方法

2.1 软件测试

2.1.1 软件测试的定义

1983年，IEEE计算机协会软件工程技术委员会提出的“软件工程标准术语”中，软件测试被定义为：使用人工或自动手段来运行或测定某个软件系统的过程，其目的在于检验该被测软件是否满足规定的需求或是衡量预期结果和实际结果之间的差别。

2.1.2 软件测试的目的

软件测试的目的是发现系统中的缺陷（并叫别人改正这些缺陷）。测试由软件产品（或服务）公司内的一些人完成，其目标和宗旨是在产品到达客户手中之前发现产品中的缺陷。软件测试的目的不是证明产品没有缺陷，而是要发现软件产品中的缺陷。

2.1.3 软件测试的原则

软件测试目的是检查软件的错误和缺陷，预测和改善软件质量，软件测试学者们根据已往的经验总结了很多测试原则，总结如下：

(1) Good enough原则。

Good enough原则是指测试的投入与产出要适当权衡，测试进行的不够充分是对质量不负责任的表现，但是投入过多的测试，则是资源浪费的表现。随着软件测试的投入，测试的产出随之增加；但是当测试投入增加到一定的比例后，其测试效果并不会明显地增加。零缺限是理想的追求，而Good enough则是现实的追求。不能盲目追求最佳的测试效果而投入过多的测试资源，应该根据项目实际要求和产品的质量要求来考虑测试的投入。

(2) 防止测试的盲目性。

对软件进行盲目的测试会导致很多问题，例如，测试用例没有周期记录，无法进行重复性测试；对测试中出现的问题不易确定程序的错误；不容易定位

找到的错误原因，更糟糕的是不能再现错误，浪费时间因而效率低下。

(3) 明确预定输出结果是测试工作绝对需要的一部分。

测试之前如果不确定预期输出结果，就很难认定缺陷是否存在。

(4) 测试用例不是用来证明程序的正确性。

我们要选取那些能够使软件失效的测试用例来测试程序。

(5) Pareto原则应用于软件测试。

Pareto原则，也叫80-20原则，在软件测试中的80-20原则是指80%的bug在分析、设计、评审阶段就能被发现和修正，剩下的16%则需要由系统的软件测试来发现，最后剩下的4%左右的bug只有在用户长时间的使用过程中才能暴露出来。

(6) 所有的测试都应追溯到用户需求。

(7) 程序员防止验证自己的程序。

这个原则的原义是指测试工作是一项需要避免主观思维的工作，程序员先天具有爱惜自己程序的特性，其实任何人都具有爱惜自己工作成果的潜意识，但是不能因为这样而免去开发人员测试的义务。因为开发人员是最清楚自己程序的人，所以对自己的程序进行更深入的检查。解决这一矛盾的方法是交叉测试、同行评审、结对编程等。

2.1.4 软件测试技术分类

针对软件测试技术能够从不同的方面进行区分（如图2-1所示）：

(1) 从是否需要执行被测试软件方面，可以分成静态测试与动态测试。

(2) 从在动态测试中是否需要分析程序结构的方面，可以分成白盒测试与黑盒测试。

(3) 从在静态测试中是否需要分析程序语法的方面，可以分成语法测试与语义测试。

(4) 从怎样选取测试数据的方面，可分为功能测试、结构测试与随机测试。

2.1.4.1 静态测试 静态测试是又称为静态分析技术，其基本特征是不执行被测试软件，而对需求分析说明书、软件设计说明书、源程序做结构检查、流程图分析、符号执行等找出软件错误。

2.1.4.2 动态测试 动态测试的基本特征是执行被测试程序，通过执行结果分析软件可能出现的错误。可以人工设计程序测试用例，也可以由动态分析测试工

具做检查与分析。通过执行设计好的相关测试用例，检查输入与输出关系是否正确。

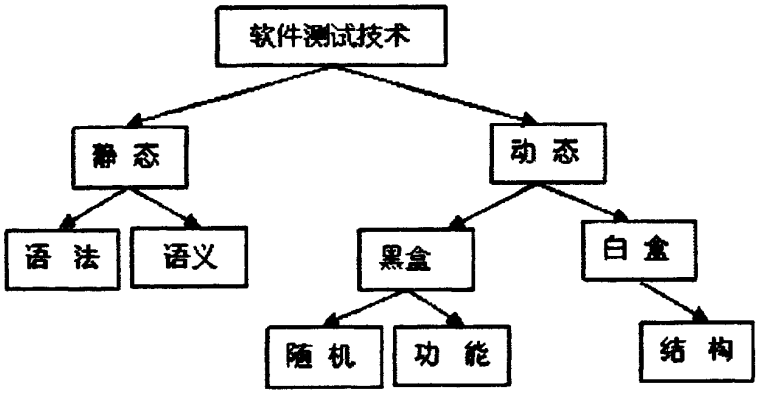


图2-1 软件技术分类

Fig.2-1 Classification of software testing

2.1.4.3 **黑盒测试** 黑盒测试技术就是对被测软件S，设S的功能空间是F，选取或者生成F的一个子集T属于F，T称为测试用例。各种黑盒测试技术所不同的是选择T的方式不同。

2.1.4.4 **白盒测试** 白盒测试要知道系统逻辑结构的工作过程，编写测试用例检验程序结构运行以确定是否与规格说明书规定的相符，检测程序中的各条路径是否都能按预期要求正常工作[23]。目前，比较成熟的白盒测试技术方法有静态白盒法、侵入式法、控制流图法、基路径法、数据定义使用法、程序片法。

2.1.5 软件测试过程

按软件测试过程中的先后顺序可以分为5个步骤：单元测试、集成测试、确认测试、系统测试，验收测试。

1．**单元测试** 软件的最小单元可能是一个具体的函数(function 或 procedure)、一个模块、一个类、或一个类的方法(method)，它应该具有一些基本属性，如明确的功能或规格定义、与其他部分明确的接口定义。因此，如果一个单元可以清晰地与同一程序的其他单元划分开来，就可以把它作为软件一个单元，进行单元测试。

2．**集成测试** 在单元测试基础之上，将所有模块按照概要设计要求组装成

为子系统或系统，进行测试。集成测试的目的是确保各单元组合在一起后能够按既定意图协作运行，并确保增量的行为正确。集成测试的内容包括单元间的接口以及集成后的功能。使用黑盒测试方法测试集成的功能，并且对以前的集成进行回归测试。

3．**确认测试** 确认测试主要是检验软件的功能与性能等特征与客户的需求是否相符，该过程也要在近似实际场景下执行。除了检验功能需求外，确认测试还要检验系统的非功能需求。确认测试的测试用例如果在客户现场没有通过，就会导致客户拒绝产品，会意味着经济损失或耗费人力物力对产品进行返工。

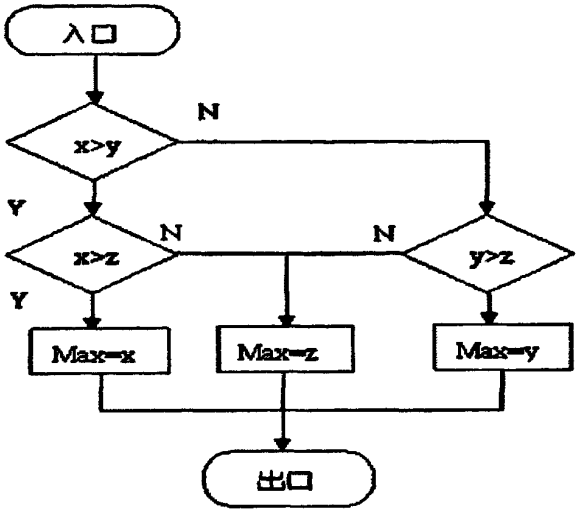
4．**系统测试** 系统测试是经过和系统需求规格相对比，检测软件与系统需求规格不一致或相互矛盾的地方。它将经过确认测试的软件，作为整个基于计算机系统的一个元素，与计算机硬件、外设、一些支持软件、数据和人员等其它系统元素相互结合，在实际运行（使用）环境下，对计算机系统进行一系列的集成测试和确认测试。

5．**验收测试** 验收测试是主要是根据用户的需求而建立，是整个测试过程中的最后一个阶段。在执行这类测试时，最终用户要参与之中。验收测试计划过程应该在需求确定之后尽快开始进行。验收测试和系统测试的主要差别是测试的主体，也就是说谁在进行测试工作。当用户在系统测试中起到了十分积极的作用时，而且测试环境的其他部分足够真实，那么验收测试和系统测试合并在一起是有意义的。

2.2 路径测试问题的描述

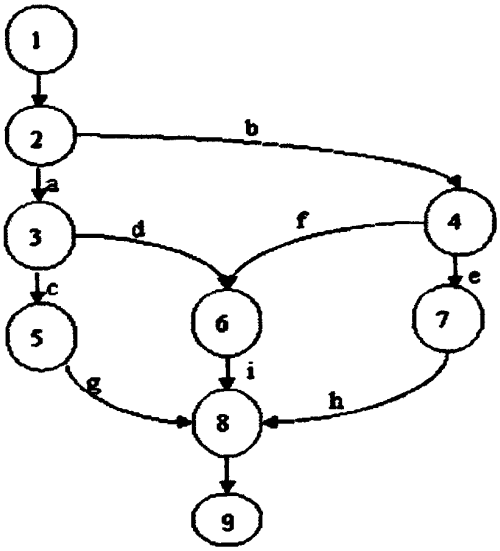
路径测试数据生成问题在软件测试中可定义为：对于一个程序P和P中的一条路径W，设P的取值范围是D，求 $\overline{X} \subset D$，使得P以 $\overline{X}$ 为输入运行，所经过的路径为W[24]。

在结构测试中，一个程序P的结构可以由控制流程图(CFG)表示，G＝(N,E,s,e)，N是结点的集合，E是边的集合，s是唯一的入口结点，e是唯一的出口结点，一个结点是一起被执行的语句集合，边(n_i,n_j)对应结点n_i和n_j的控制转换。如果n_i的最后一条语句是一个选择语句或迭代语句，边(n_i,n_j)被称做一个分支。一个分支谓词可以赋给一个分支。一条路径是一个结点序列$(n_1,n_2,\ldots n_k)$，$K \geq 2$， 这样的边存在于n_i到n_{i+1}，$1 \leq i \leq (k-1)$, $n_1=s$并且$n_k=e$。一个程序P的输入变量是出现在输入语句中的变量，一个输入参数或者是P中使用的全局变



a) 程序流程

a) Flow chart of program



b) 控制流图

b) Control flow chart

图2-2 取三个最大值程序的程序流程图和控制流图

Fig.2-2 The flow chart and control flow chart of an exmple program

量。输入变量的类型可以是编程语言中的任何一种类型。输入变量x_i的取值范围D_{xi}是x_i 可取的所有值的集合，程序P的输入域D的输入空间D是n个输入变量取值范围的笛卡尔积$D=D_{x1}\times D_{x2}\times\cdots\times D_{xn}$。输入数据集合id是一组赋给输入变量的值构成的集合，id是n维输入空间D中的一个点。

图2-2是求三个数最大值程序的程序流程图与控制流图。假设入口的3个参数是x=4，y=3，z=5，那么程序就经过路径Pathl:1-2-3-6-8-9，还可用<adi>表示。这是该程序所有可能路径中的一条。复杂的程序路径的组合可能是非常庞大的，要在测试中覆盖所有的路径往往是不太现实的。要解决路径覆盖这一难题就要把覆盖的路径数目缩减到一定限度。因此，路径测试的目标就是从测试用例中挑选一些测试用例使得其所经过的路径能够达到某种覆盖标准。

2.3 测试用例生成系统结构

一个典型的测试用例生成系统由程序分析模块、路径选择模块、数据生成模块三个部分组成。如图2-3所示。

经过静态分析代码模块，获得路径选择所需的相关数据，举个例子，程序的控制流图等；通过解析上个模块传来的控制流图等信息，由路径选择模块来发现合适的测试路径集合，也就是说，通过对这个集合里的路径进行测试，人们能够获得对该模块的一定程度的路径覆盖，数据生成模块接收测试路径集合，它产生可以经过集合中路径的输入数据。

2.3.1 程序分析

程序员熟悉通过静态分析程序初始生成被测程序的程序流程图。还可通过一些工具如AutoFlowchart来生成流程图，然后再按照要求生成控制流图。这里的控制流图就是对程序流程图的简化，相当于把程序流程图中的每个处理符号都简化成一个点，原先连结两个处理符号的箭头，变为连结两个结点的有向弧，这样得到的有向图称为控制流图。一般称控制流图中开始结点之后那个结点为入口结点，称终止结点之前的那个结点为出口结点。控制流图单体现程序内部的控制流程，不体现对数据的具体操作以及分支或循环的具体条件。

控制流图的关键是它可以明确将该程序的执行对应于从入口结点到出口结点的路径。并且为程序员分析和处理程序中隐藏的繁多的可执行路径[25]提供较好的理论上可预见的方式。

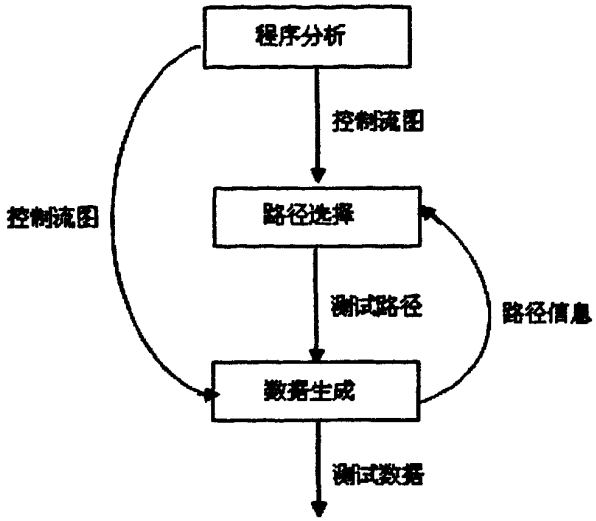


图2-3 测试用例生成系统结构

Fig.2-3 Architecture of test case generation system

2.3.2 路径选择

此阶段所要做的是基于一定的策略或者说是测试覆盖标准，来选取一组测试路径集合。该标准不仅可以判断一组测试用例的好坏，还规定了测试结束的条件。

2.3.2.1 基本路径测试法 此方法开始在控制流图上分析环路复杂性，获得基本可执行路径集合，称为基本路径集，接着编写测试用例，确保在测试中程序的各条可执行语句至少执行一次。

2.3.2.2 逻辑覆盖法 逻辑覆盖考虑测试用例对程序内部逻辑覆盖的程度。最完全的覆盖是覆盖程序里的各个路径，可是因为程序中会包括循环，路径的数目庞大，所以每一条路径都能被执行到是不现实的。所以，我们尽量做到使覆盖的程度高一些。最常用的逻辑覆盖技术有[26]：

1. **语句覆盖** 选取充足的测试用例，让程序中各条语句至少被执行一次。
2. **判定覆盖** 判定覆盖的含义是，让程序的各个分支至少执行一次，让程序里各个判定至少都可取得一次“真”值与“假”值。
3. **条件覆盖** 条件覆盖要求程序判定里的每个条件全部取到各种可能的结

果

4．判定/条件覆盖 因为判定覆盖与条件覆盖的包含关系是不确定的，所以都满足两种覆盖标准的逻辑覆盖，叫做判定条件覆盖，也就是说，选取测试用例，让程序判定里各个条件取到所有可能的值，且让各个判定取到所有可能的结果。

5．条件组合覆盖 在条件组合覆盖中考虑了判定中每个条件的所有可能结果，但并没考虑条件的组合情况。条件组合路径覆盖要编写充足的测试用例，使得当以这些测试用例做为程序输入运行程序时，各判定中条件结果的全部可能组合最少出现一次。

2.3.3 测试数据生成

自动生成测试数据的方法不断出现。一般包括基于规格说明的测试数据生成与基于程序结构的测试数据生成两种方法。本文深入探讨的是基于程序结构的测试用例生成方法，总体包括以下几种：

2.3.3.1 符号执行法 在一条给定的逻辑路径上执行该路径上的语句。程序的输入数据可以是确切的数值，还可以包含符号值。创建约束系统的方法总体分为从前向后替换与从后向前替换。

2.3.3.2 随机法 随机法对输入空间D进行随机选择，生成一个测试数据的开销较小，简便易行是其主要特点。它能自由的迅速生成很多的测试数据，而不受输入空间的类型的限制。

D.Bird等用随机法优化PL/1，验证编译程序生成能全部执行并拥有自检功能的PL/1程序。但是该方法为满足特定的测试需求，会生成很多的冗余测试数据，这是十分不可取的。并且对于特殊的测试要求也不容易被达到。举个例子，要使测试基准程序三角形程序达到分支覆盖会产生二十多万组测试数据。

2.3.3.3 程序直接执行法 Korel研究从输入空间中随机选一组数据，按步进的方式运行程序P，即一次前进一个分支谓词。

因为使用了步进方式，所以这种方法可以尽快发现不可行路径。然而，由于它运用回溯技术且一次只考虑一个分支谓词和一个输入变量，所以即使指定路径上的所有谓词函数都是输入变量的线性函数，也要进行多次迭代。对于非线性路径约束，此方法在非线性路径约束问题上仅可以发现局部极小值，如果谓词函数包含大量局部极小值时不易发现问题的解。因此程序直接执行法对于非线性路径约束是不完备的。Bogdan Korel为Pascal语言的结构化子集设计的程

序生成测试数据，奚红宇研究Ada软件的测试数据生成都采用了该方法。

M.Gallagher等人研究了程序插装，迫使程序以D中随机一组数据为输入来执行路径w，并用程序运行时由插装语获得的相关信息获知各变量和分支谓词的状态。对P中每个判定语句插装，使插装语句插在分支谓词语句之前。

迭代松弛法使用程序切片思想，在D里随机取一组输入观察W上各分支谓词，经过静态、动态数据流分析明确各谓词函数对于入口参数的依赖关系，构建谓词片与动态切片，构建这些谓词函数有关当前输入的线性算术表示，接着构建入口参数的增量的线性约束系统，进一步构建入口参数的增量的线性方程系统，求解后取得每个入口参数的增量，进而取得一组新的输入。

2.3.3.4 遗传算法的应用 因为发现的软件缺陷部分依赖搜索的强度，一些研究者提出使用启发式技术自动生成测试用例，目标是以合理的代价找到产生更穷尽的测试数据的一种方法。最具前景的启发式算法包括遗传算法，一种模仿生物进化模式的搜索算法。在遗传算法中，数据被编码为染色体字串，使用类似生物遗传学中的选择、交叉、变异等运算对问题进行求解。其中交叉和变异运算的目的是为种群赋予多样性，以避免算法停止在局部极值点上。遗传算法的每一次迭代求解都是以种群为单位的，这一特性可以通过并行运算来进行。最近一段时间，有人提出使用遗传算法进行软件测试有例生成，在实验中获得了可观的结果。北航软件所的一些研究人员进行了使用遗传算法生成测试数据方面的研究，编写了相应的软件，并且发现基于遗传算法的测试数据生成技术具有比爬山算法和随机法更好的性能；汪浩等人为基于该方法的系统建立了模型，此外，他们也对用于该目的的 TCAG 软件测试用例生成工具进行了介绍。景志远研究了用于生成测试用例的改进遗传算法，如 MGA 和 K 均值等。

除了测试用例生成，Roper等人使用遗传算法进行语句覆盖和分支覆盖方面的研究；Borgelt和Lin等则进行了该算法在路径测试方面的研究；Jones和Joachim等人的研究提出，基于遗传算法的测试数据生成在三角形分类等程序中所生成的数据比随机法要低一至二个数量级。文献[27]提出模拟退火和遗传算法的混合算法，使两者取长补短。

GAs由一个适应度函数引导，被设计用来为特定的代码产生输入数据，例如覆盖一个程序的所有语句，分支或者可行路径或者在特定环境条件下测试软件性能。因为一个构造得好的适应度函数能够显著的增大成功的可能性，用较少的迭代覆盖目标代码，近来的研究已经集中在确定高性能的适应度函数去引导这样的结构测试。为了评估给定的适应度函数的性能，以前的研究者通过把遗传算法产生的结果与随机测试数据生成进行比较，虽然这种方法有效的演示

了进化技术的作用，但是何种适应度具有较高的性能还没有明确结论。在对不同程序进行测试时，应采用适当的遗传方法。如何提高遗传算法的性能，需要进一步的研究。

2.4 本章小结

本章介绍了有关软件测试技术的基本理论和方法，并对常用的几种典型测试方法进行了分析研究，将遗传算法应用于测试技术，大多是对特定测试目标进行优化，这些方法较其它测试方式普遍有更好的性能。在此基础上，提出了面向路径覆盖的测试用例生成方法，将遗传算法作为核心算法，为待测程序生成满足目标路径的最优解。

第3章 遗传算法及适应度函数研究

遗传算法(Genetic Algorithms，GA)是基于遗传和自然选择等生物进化机制的启发式随机搜索自法，和其它许多搜索算法一样，也具有迭代形式。GA由美国密执安大学John Holland及其研究团队提出并发展起来的，他提出这一算法的最初目的是研究自然系统的自适应行为[28]。Holland的学生Goldberg在其博士论文中首次提出“遗传算法”这一名词。之后，他进行归纳总结，形成了遗传算法的基本框架[29]。遗传算法在实际应用过程中得到进一步发展和完善。

3.1 遗传算法用于软件测试

遗传算法使用进化方法去识别和评估复杂问题的可能解，这种问题的最优解也许是不可获得的。遗传算法的名称意味着传递这样一个概念，一个特定问题的候选解在结构上与一个染色体相似，描述GAs的语言反映了这种类比。像其它的自适应搜索算法，GAs并不确保找到最优解，但是它们被证明在时间和系统资源有限的条件下发现优质解是有效的。因为GAs尤其适用于搜索空间巨大，复杂或不易理解的问题。

在软件测试中，目标是找到满足特定测试标准的测试用例，为了做到这一点，遗传算法开始于随机产生的种群，该种群是一些被称作染色体的可能的测试用例。每个染色体被用作被测代码的输入。基于结果，一个适应值被计算，用来评价该染色体对于求解问题的优劣。使用这个适应值，产生一个新的染色体种群，适应值高的染色体存活率就高。染色体的多样性通过交叉和变异运算被赋予，重新产生下一代，选择和交叉直到达到了停止条件。

遗传算法的主要执行步骤如下（如图3-1）：

(1) 算法初始化：确定编码方法、适应度函数、群体规模、选择算子、交叉算子、突变算子、交叉概率、突变概率等算法参数；

(2) 随机产生解的初始种群；

(3) 计算群体中每个个体的适应度，并判断是否符合优化准则，若符合，输出最佳个体及其代表的最优解，并结束计算；否则转向第 4 步；

(4) 选择，按适应度高的原则，从旧群体中选出个体组成群体 A，群体 A 的规模与旧群体相当，旧群体中的个体可能在群体 A 中多次出现，也可能不出现；

(5) 按照一定的交叉概率和交叉方法，生成新的个体；

(6) 按照一定的变异概率和变异方法，生成新的个体；

(7) 由交叉和变异产生新一代的种群，如果群体性能已满足要求，或者已达到最大迭代次数，则算法停止，输出群体中适应度最高的个体作为结果；否则返回到(3)。

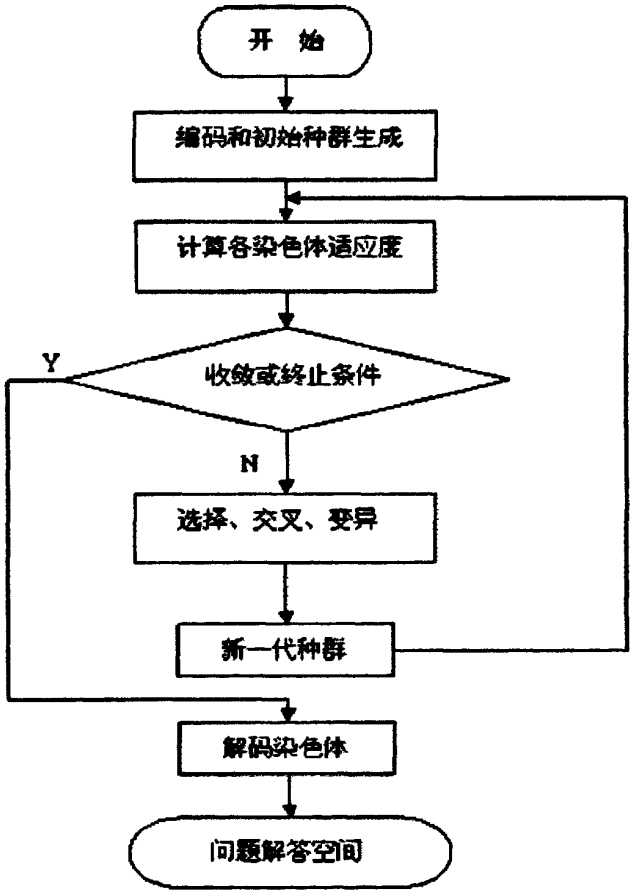


图 3-1 遗传算法工作流程图

Fig.3-1 The flow chart of genetic algorithm

3.2 遗传算法的构成要素

3.2.1 染色体编码

染色体编码是把实际问题的解空间用计算机能够识别的符号串来表示。使用生物的基因位串来表示实际所求问题的候选解。编码方法是否合适对问题求解的质量与速度关系密切。DeJong 提出了编码的两条基本原理[30]。

编码原则一（最小字符集编码原则）：应使用能使问题得到自然表示或描述的最小编码字符集的编码方案。

编码原则二（有意义积木块编码原则）：所选编码方式应能易于产生与求解问题相关的且具有低阶、短定义长度模式的编码方案。

原则一为实际编码工作指出了方向，根据原则一，由于实际问题中往往采用十进数，用二进制数字串编码时，需要把实际问题对应的十进数变换为二进数，使其数字长度扩大约 3.3 倍，因而对问题的描述更加细致，加大了搜索范围，有效避免局部寻优。而且二进制编码的字符集小，它优于非二进制编码；从另外一个角度看，进行突变操作时运算速度快。

但是，缺点也存在于二进制编码当中，如：使用二进制编码效率不高，还会让遗传算法的性能变差。而且编码时需进行十进制数到二进制数变换，输出结果时又要解码，进行二进制数到十进制数的变换；如果二进制数串十分长，交叉操作计算量就大。所以，就有人使用十进制数编码与符号（字符串）编码等。因为数字串长度较二进制数短，交叉运算计算量较二进制少；但变异运算较二进制编码繁琐；十进制串编码方式为人们所熟悉，不需要进行十进制和二进制数相互的变换。

3.2.2 适应度函数

遗传算法只用适应度函数来评估染色体的好坏，在此基础上使用各种遗传操作，基本不用搜索空间的知识，所以适应度函数选择是否恰当对算法的性能优劣有十分重要的影响，要依据所求解问题的特点具体确定。基本遗传算法通常选取下面两种方法之一将目标函数值 f(X)转换为个体的适应度 F(X)，其目的是使个体适应度非负。

方法一：针对求目标函数最大值的优化问题，变换方法为：

$$F(x)=\begin{cases} f(X)+C_{min} & if\ f(X)+C_{min}>0 \\ 0 & if\ f(X)+C_{min}\le 0 \end{cases} \tag{3-1}$$

方法二：针对求目标函数最小值的优化问题，变换方法为：

$$F(x)=\begin{cases} C_{max}-f(X) & if\ f(X)<C_{max} \\ 0 & if\ f(X)\ge C_{max} \end{cases} \tag{3-2}$$

再者，在较小的种群规模下，如果遗传算法运行的初期某些染色体的适应度非常高，那么在传统的选择方法下，它们就会被赋予一个非常高的选择概率。这会导致这些染色体大批的繁殖，进而，种群的绝大部分为其后代，这便降低了种群的多样性，算法在局部极值点处就开始收敛，无法达到全局最优解。如果在算法运行即将结束的时候，大部分的染色体都有一个高适应度值，那么这时种群的最高适应度和平均适应度彼此接近，遗传算法会以几乎相同的概率选择最高适应度的染色体和选择平均适应度附近的染色体，于是该算法实际上是以近似随机的方式来选择染色体，染色体的适应度不再起作用，算法也就不具有效用。基于此，有必要对使用的适应度函数进行数学变换，使得在保证遗传算法的可行性的同时，确保种群中染色体具有多样性，并使之具有适当分散的适应度值，使得种群中染色体的适应度有一定的区别，同时又不至于相去甚远，保证染色体相互的健康竞争，以及遗传算法的优良效果。下面列出了一些较为常用的变换方法。

(1) 线性尺度变换：将优化目标函数转换为适应度函数的线性函数

$$F'=aF+b \tag{3-3}$$

式中：a，b 为系数，能依据求解问题的特性与期望的适应度值的分散程度于算法初始时明确或在每一代过程里重新计算。

(2) 乘幂尺度变换：将优化目标函数转换成适应度函数的幂函数：

$$F'=F^k \tag{3-4}$$

(3) 指数尺度变换：将优化目标函数转换成适应度函数的指数函数：

$$F'=\exp(-\beta F) \tag{3-5}$$

式中 β 决定着选择的强制性，β 越小选取该个体的强制性就越大。

3.2.3 遗传操作

3.2.3.1 选择操作 选择算子用于从群体中按个体的适应值选择出较适应环境的个体，作为待繁殖的父代个体。选择中使适应度高的个体繁殖下一代的数目较

多，适应度低的个体繁殖下一代的数目较少，甚至被淘汰。直观的说，适应度高的一个个体可能被反复多次选择成为多个待繁殖的父代个体，而适应度低的一个个体被选择的次数很少，甚至可能为 0。显然，选择算子是对生物进化中自然选择的模拟，保证了迭代过程中“适者生存”的群体进化现象。选择算子是影响遗传算法收敛效果和速度的主要因素。

针对给定的规模是 n 的种群 $P=\{a_1,\ldots,a_n\}$，个体 a_j 的适应度是 $f(a_j)$，其选择概率是：

$$p_s(a_j)=\frac{f(a_j)}{\sum_{i=1}^{n}f(a_i)},j=1,2,\ldots,n \tag{3-6}$$

公式确定后代种群中个体的概率分布。通过选择操作产生用来繁殖的交配池，在父辈种群里各染色体存活的期望数目是：

$$p(a_j)=np_s(a_j),\ j=1,2,\ldots,n \tag{3-7}$$

3.2.3.2 **交叉操作** 交叉操作采用来源于不同符号串的基因通过交换混合来获得新符号串的概念。根据交叉点的不同，交叉运算又有单点交叉、多点交叉、均匀交叉等多种不同的交叉方法。

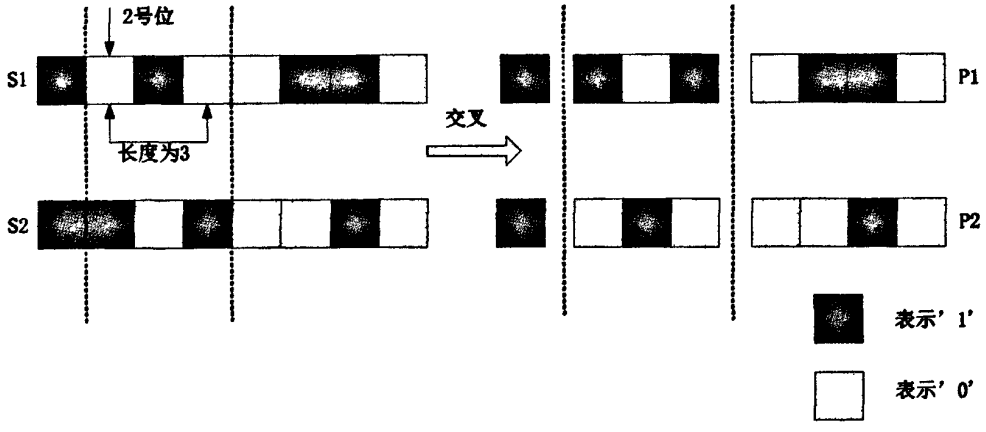


图 3-2 染色体交叉示意图

Fig. 3-2 picture of chromosomes crossover

我们用两点交叉来具体说明．两点交叉操作的简单方式是把被选取来的两个个体 S1 和 S2 当作父母两个体，把两者的某些基因采取交换。例如

S1=10100110，S2=11010010，获得一个在 1 到 8 之间的随机数 c1（代表交叉的起始点）和 c2（代表交叉基因的长度），假设产生的为 2 和 3，并设其一个后代个体是 P1，另一个是个体 P2．它们互换后结果 P1=11010110，P2=10100010。如图 3-2 所示。

3.2.3.3 **突变操作** 变异算子用于染色体上位串产生随机的变化，得到新的个体。能遗传的突变是生物进化的重要手段，对于保证群体多样性具有不可替代的作用。

3.3 遗传算法的问题和改进

遗传算法虽然是一种性能较好的算法，但在实际求解问题时，会产生早熟收敛、局部寻优能力弱以及中后期搜索效率差等问题。比如，进化种群里少数个体的适应值明显优于其它个体，导致经过少数几次迭代后，这些个体就占满整个种群，进化过程就过早结束了。在实际研究应用中，人们对遗传算法进行了许多改进和变形，具体体现在三个方面，一是在编码方法上，除了二进制编码以外，现已提出顺序编码、实数编码、整数编码等多种编码方法。对应不同的编码方法，有相应的交叉和突变算子。二是提出了许多不同的选择策略，如截断选择、顺序选择、正比选择、局部选择、锦标赛选择等。除了选择、交叉、突变三种主要遗传操作之外，人们还根据已知的生物进化和遗传机理提出在遗传算法中应用其它高级遗传操作，或称其为次要遗传算子。包括倒立操作、显性操作、生态操作、迁移操作、性别区分等。另外，针对不同问题还出现了分布式遗传算法、并行遗传算法等等[31]。

适应值选择的好坏直接关系到所得的下一代群体的质量，传统的方式是对个体进行“优胜劣汰”的选择，但同时也会存在着抽样误差，因而导致求解的精度不够[32, 33]。本文针对 SGA 仅仅依靠变异产生新的基因值而导致局部搜索能力较弱的特点，提出使用改进适应度选择策略的遗传算法，它可以有效克服这个问题，提高适应值选择的精度，并应用于面向路径的测试用例自动生成问题的求解上，评估六种适应度函数的性能。

3.4 面向路径的适应度函数设计

适应度函数在遗传算法的成功搜索中扮演着重要的角色，构造得好的适应度函数不仅提高了找到可行解的可能性，而且产生更好的全面的代码覆盖，同

时消耗较少的系统资源。表3-1描述了路径测试研究中被测的适应度函数，适应度函数分为逼近程度(approximation level)，距离计算(distance calculation)或者两者兼有。逼近程度指示实际执行路径与将要到达的局部目标的接近程序。例如，测试用例所经过的正确的结点数或者该路径被产生的频度。从正确结点数的情况看，有较高逼近程度的测试用例被认为比有较低的逼近程度的测试用例更合适。距离计算检查路径实际在哪个分支结点与期望路径偏离，它的目标是衡量特定的测试用例完成到期望路径的谓词条件的接近程度。

表 3-1 被测适应度函数

Table 3-1 Fitness function tested

	Distance calculation	Approximation level
NEHD	no	yes
BP1	yes	yes
BP2	yes	no
IPP	no	yes
DO-NEHD	no	yes
DO	no	yes

针对具体的遗传算法的应用，如何定义适应度函数以找出满足所需路径覆盖的测试用例是用遗传算法解决问题的关键[34]。遗传算法采用了适应度进行优化，有好的测试用例生成时，其性状能较好得到遗传，在随后连续进化的子代中，仍能保证该类测试用例有较大的产生概率[35]。

下面的适应度函使用图 3-3 的程序来描述。五个可行的路径在表 3-2 中。

表 3-2 实例程序路径

Table 3-2 Paths through instance program

Number	Path	Result triangle
1	1T2T3T4A	Equilateral triangle
2	1T4F	Neither
3	2T4F	Neither
4	3T4F	Neither
5	4T5T	Scalene

```
Void triTest(int i,int j,int k){
1.                int tri=0;
2.                String type;
3.                if  (i==j)   tri+=1;          1T
4.                if  (i==k)   tri+=2;          2T
5.                if  (j==k)   tri+=3;          3T
6.                if  (tri==0)  {               4T
7.                  if  ((i+j)>=k) || (j+k>=i) || (i+k>=j)) {
                     type="scalene";            5T
8.                  }
9.                }
10.             else if (tri>3)                 4A
11.               type="equilateral";
12.             else                            4F
13.               type="neither";
14. }
```

图 3-3 实例程序

Fig.3-3 instance program

3.4.1 广义哈明距离

广义哈明距离 (Normalized extended Hamming distance)，在路径测试中，e 采用哈明距离衡量被覆盖分支与所选择分支之间的距离，它仅能够衡量两个没有特定次序的对象之间的距离。但是针对路径，两个不同的路径也许包含相同的分支却存在不同的次序，因此简单的哈明距离不再适用。NEHD 是在哈明距离的基础上所做的改进，能够很好的衡量两条给定路径之间的距离[36]。举一个简单的例子解释这一过程。

使用简单的程序如图 3-3，给定的路径是'1T2T3T4A'用 P_L 表示，当第一个测试用例运行的时候，它找到路径'2T4F'，用 P_F 表示。为了决定这个测试用例的适应度函数，第一步是计算这两个路径之间的距离和相似性。第一阶的距离(first-order distance)是由只在两个路径其中之一出现的分支数目决定，因此有四个分支 {1T，3T，4A，4F}。相似性(similarity)是两个路径的并，在这个例子中，包含 5 个分支 {1T，2T，3T，4A，4F}。第一阶的值是 1 减去距离集的大小除以相似性集，即 1-4/5=1/5 或 0.20. 第二阶是所有连续分支的不同对，例如 P_L 集是 {1T2T，2T3T，3T4A}，P_F 是 {2T4F}，产生相同的相似性和距离集{1T2T，2T3T，3T4A，2T4F}。因此，第二阶的值是零，最终集合停止，假如这个值大于零，适应值计算就继续以三个分支为一组，然后四个，直到阶值

为零。在计算实际的适应值时，权重因素被应用到阶值中。第一阶的权重是1，随后的权重是前一阶的权重乘以目标覆盖路径的组合路径数目。例如，P_L有四个单一路径的组合，三个两两组合。这个测试用例的适应值=(1*1/5)+(0*4)=0.20。

适应度函数 NEHD 释放路径的特点，具有良好的潜力，因为它避免抑制一些能够找到与搜索路径相似的路径的测试用例。例如，两个路径{abcd}和{ebcd}因为初始结点不同，所以看起来没有什么关系，然而内部却有很大的相似处。它的缺点是在路径比较长的情况花费计算时间，因为要计算每个阶段的组合数目。在寻找相同路径的情况下，做为输入的测试数据在适应值上没有差别，因此可以记录特定路径的适应值，减少计算时间。

3.4.2 分支谓词法

在路径测试中，不同的测试用例满足不同的分支。因此，种群成员需要按满足特定分支程度和接近整个路径的程度分等。下面介绍两个不同的基于谓词的分支适应度函数。

1．**分支谓词 1(BP1)** 使用相似推理和 Hamming 距离，以确定是否测试用例产生一条与被优化路径相似的路径[37]。适应度函数由两部分组成：路径的相似度和罚函数。相似度是分支 P_L 和 P_F 从输入结点到偏离点的相同点的数目。例如，P_L='1T2T3T4A'，P_F='1T4F'，相似度为 2，因为偏差点在第二个结点。如果 P_F='1T2T3T4A'，那么相似点为 5(匹配的结点数加 1)，因为所有的结点都匹配。匹配的结点数越多，测试用例越接近目标状态。罚函数由分析偏差结点的分支谓词计算得来。对一个给定的测试用例{1，1，8}，要优化的路径P_L='1T2T3T4A'，偏差点是在语句(i==k)为假时。惩罚是使语句为假和使语句为真的差，即|i-k|或者 7。结果除以所有测试用例共同的惩罚值。例如两个测试用例在都在第一个结点处偏离，它们的偏差值是 1，如果第一个测试用例的惩罚值是 1，第二个是 2，那么两个都要除以 2，并且从偏差值里减去。如下：

case1=1-1/2=0.5

case2=1-2/2=0.0

每一个结果加 1，使测试用例的适应值不会为零。布尔条件由常量来处理，尽管实际的常量并不是由最初时给定，其它的研究对分支测试使用一个相似技术，对真值和假值条件语句使用常量 1000 做为惩罚值[38]。

2. 分支谓词 2(BP2) 仅使用上述公式的罚函数部分，取罚函数的倒数，使得测试用例一旦接近满足条件语句，就会有较高的适应值。该方法的弱点是它不能区别测试用例是否完成一条路径覆盖。也就是说它不能区分在不同点偏离的，具有相同适应值的两条路径。例如，P_L＝'1T2T3T4A'，输入测试用例是(1，1，8)。在结点 2 处偏离，这个染色体的适应值是 1/|(1-8)|=0.142。但是，假如这个测试用例是(1，5，3)，在结点 1 处偏离，它的适应值就会是 0.25。尽管第一个测试用例执行的路径比第二个更接近目标。使用这种方法与模拟退火和 Korel 算法比较，尽管被测程序不包含布尔语句，模拟退火和 GA 都要优于 Korel 算法[39]。

3.4.3 逆路径概率

逆路径概率 inverse path probability(IPP)[40]记录当前已找到的每条路径的次数，次数的倒数即为当前测试用例的适应值。因此，当首次找到某一路径时，它的适应值为 1，随后其适应值将逐渐减少，其公式为

$$Fitness=1.0/sum\_path_i*population\_size \tag{3-8}$$

其中 sum_pathi 是测试用例覆盖的此路径的次数。例如，有一个种群为 20，path2 已经被发现了 5 次，新产生的测试数据集为{3，4，4}，它的适应值为 1/5*20 或者 4.0。该方法的优点是它不鼓励与 P_L 相似的容易找到的路径，例如一个占用了很大搜索空间的路径。缺点是鼓励新找到的实际对搜索并没有帮助的（不相似的）路径。

3.4.4 广义哈明距离偏移

Deviation only plus NEHD(DO-NEHD)是一个新的适应度函数，只使用适应函数 NEHD 的偏移部分，减少了路径匹配的计算时间。在 NEHD 中，适应函数有两个元素，第一个是 P_L 和 P_F 之间的偏移点，第二个是测定两条路径最长连续部分的共有路径，例如，假如 P_L='1T4F'，P_F='2T4F'，偏移点是 1，因为它们在第一个结点处不同，最长的共有部分也是 1（它们都有 4F），公式

$$Fitness=DO+subP^2 \tag{3-9}$$

DO 是偏移点，subP 是两条路径共有的最长子路径的长度，所以此时适应度为 2。该方法的优点是它的计算速度和考虑路径当中的两个不同的因素。鼓励那

些有相同开始结点要优化的路径，还有那些在早期偏移的结点，但是有共同子路径的路径。

3.4.5 唯一偏移

唯一偏移(Deviation Only)，适应度函数 DO 是 DO-NEHD 的简化版，用来描述与适应度函数 BP1 罚函数的相关性以及当 NEHD 被添加到 DO-NEHD 函数中结果的差异度。仅使用从开始结点到偏离点的长度，使用相同的例子，P_F 的适应值将是 1.0，该方法的优点是偏移点计算速度快，缺点是偏移早的路径被抑制，尽管它可能包含一些好的元素。

3.5 本章小结

介绍遗传算法一些基本概念，详细分析了遗传算法及其特点，以及六种面向路径测试的适应度函数的计算方法。在这些适应度函数中，有 4 种适应度函数的计算方法在以往的基于遗传算法的面向路径的测试数据生成中被用到，通过与随机生成测试用例比较来评估其性能。虽然这些方法有效的体现了进化测试技术的作用，但是针对遗传算法所采用的不同的适应度函数哪种具有更高的性能没有任何建议。下面将引进基于遗传算法的自动测试用例自动生成系统评估适应度函数的性能。

第4章 使用遗传算法实现测试用例自动生成

遗传算法在减少测试冗余数据方面是有前景的，它可以很好地应用于面向路径的自动测试过程中[41, 42]，对测试用例所占用的空间和性能进行优化。在测试用例自动生成的研究中，程序在不同测试用例的驱动下，有不同的执行路径，它们被执行的几率也不相同。对于容易执行到的路径，往往有较多的冗余测试用例生成，而对难以执行到的控制路径，却只有很少的测试用例。遗传算法可用于指导测试用例的自动生成，针对较易执行的程序路径，算法可以抑制这类测试用例的生成，相反对难以出现的执行路径，算法将尽可能鼓励这类测试用例的生成。因此，可以提高测试用例生成的有效性，保证测试的充分性。

本章详细介绍测试数据生成模块的主要工作内容和工作成果，并重点介绍Generatedata算法包的算法实现。

4.1 系统框架结构

通过静态分析被测程序，得到程序的控制依赖图，根据某种覆盖标准或测试需求确定测试路径集合。对于较易执行的路径使用随机测试工具生成测试数据，但是对于使用随机方法很难执行到的路径，使用测试数据生成模块来有针对性的生成测试数据。测试数据生成模块系统框架如图 4-1，其中，GenerateData是测试数据生成模块的核心模块。

4.2 测试环境构造

在测试数据生成模块，首先需要进行测试环境构造。测试环境构造是测试数据生成模块工作的基础。它的主要工作包括：被测程序插装、驱动程序生成、I/O 处理和外部变量处理。对被测程序进行程序插装，采用的插装策略是对分支谓词的插装，有两种形式：一种是对 if 分支结构的插装，一种是对while 循环结构的插装。对 if 分支结构的插装是将插装的语句紧挨分支谓词放置就可以了。这些插装语句把程序动态执行时显示的相关信息给测试者。例如一段取三个数最大值的程序代码如图 4-2 以及插桩后的程序如图 4-3。

被测程序有时不是估立的，它会其它的模块有关系，要模拟这一情况，在进行测试时需要设置两种辅助测试模块：一个驱动模块(driver)和(或)若干个装

模块(Stub)。驱动模块(driver)常被称为“主程序”，作为被测模块的上级调用模块，它接收测试数据并将这些数据传递给被测试模块，驱动被测程序运行并输出相应的信息。

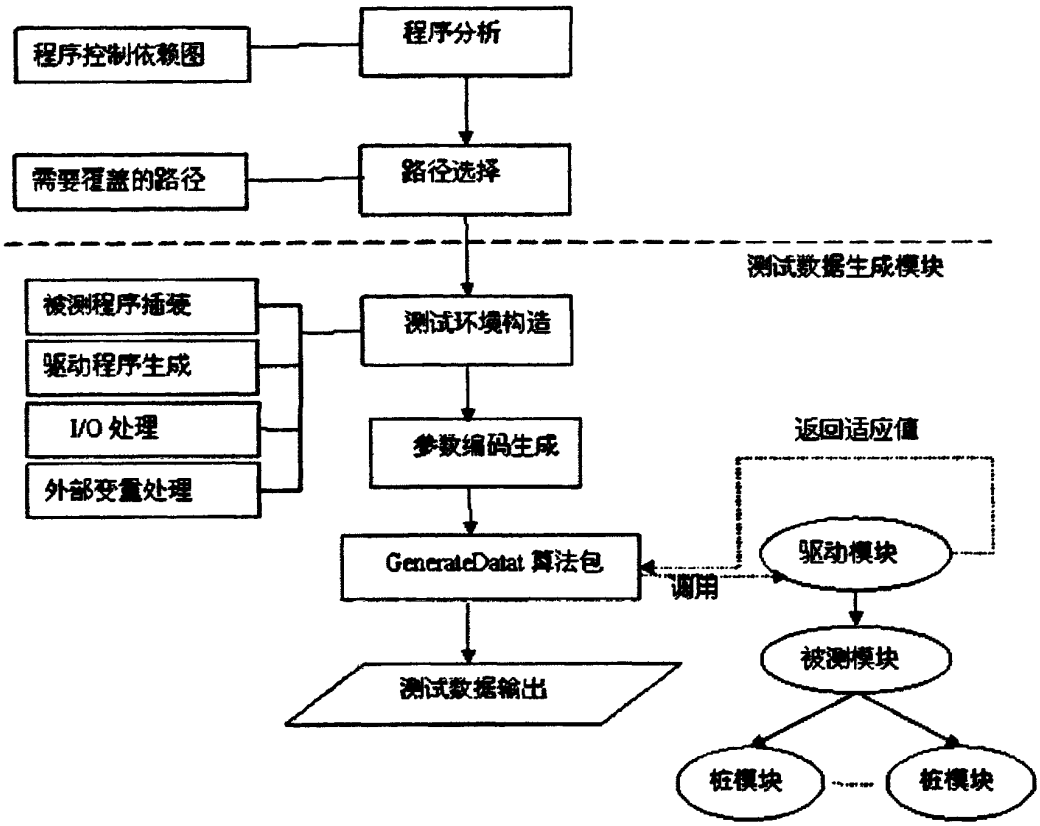


图 4-1 测试用例生成系统结构

Fig.4-1 Architecture of a test case generator system

```
function[themax]=max(x,y,z);
if(x>y)                                    %a
      if(x>z)                              %c
       themax=x;                           %g
      else                                 %d
       themax=z;
      end
else                                       %b
      if(y>z)                              %e
       themax=y;
      else                                 %f
       themax=z;
      end;
end;
```

图 4-2 程序实例

Fig.4-2 Program instance

```
function[themax]=max(x,y,z);
if(x>y)                                    %a
    concatpath('a');                       %插装语句
  if(x>z)                                  %c
    concatPath('c');                       %插装语句
    themax=x;                              %g
  else                                     %d
    concatPath('d');                       %插装语句
    themax=z;
  end
else                                       %b
    concatPath('b');                       %插装语句
  if(y>z)                                  %e
    themax=y;
  else                                     %f
    concatPath('f');                       %插装语句
    themax=z;
  end;
end;
```

图 4-3 插桩后程序实例

Fig.4-3 Program instance after instert

4.3 参数编码生成

对于面向路径测试，影响被测单元期望路径的输入数据的个数和类型是不确定的，因为遗传算法作用在参数编码上与实际问题本身无关，所以，把遗传算法应用到测试数据生成的关键任务是参数的选择，并且生成参数编码的方法。

4.3.1 参数的选择

在模块的路径测试中与路径相关且有联系的参数有入口参数、全局变量、局部变量三类。选择和被测模块里给定被测路径条件表达式联系密切的变量编码，其余的变量都不给予编码。

针对入口参数的原则是：只要给定路径的条件表达式中包括该参数即编码，其它变量一概不编码；针对全局变量与局部变量的编码原则是：只要给定路径的条件表达式中包含了该变量，就要给予编码；针对测试模块的出口参数，由于它不会影响所被测的路径，所以不对它给予编码。

4.3.2 编码方式

遗传算法要把问题的解空间用有限字母表上的定量长度的串表示。所以当遗传算法生成测试数据时，要找到恰当的映射方式 f，让测试程序的入口参数 $\{y_1, y_2, ...y_n\}$ 可以表示成一种合适的编码形式：

$$f:\{c_1', c_2', ...c_n'\} \rightarrow \{y_1', y_2', ...y_n'\}$$

遗传算法在进行遗传操作时，算子 g 对参数编码进行操作，变换编码的结构，形成新的编码形式：

$$g:\{c_1, c_2, ...c_n\} \rightarrow \{c_1', c_2', ...c_n'\}$$

f 逆映射 f'解码得输入参数的当前值 $\{y_1', y_2', ...y_n'\}$：

$$f:\{c_1', c_2', ...c_n'\} \rightarrow \{y_1', y_2', ...y_n'\}$$

评价解码后参数的当前值，假设所得的参数编码形式{y1',y2',...yn'}没有达到终止条件，遗传算子 g 就循环以上过程，直至发现目标参数值。

入口参数很多的情况下，针对二进制编码能够使用“多参数级联编码”的方案。换句话就是将各个变量的编码按次序级联，构成一条二进制串，详细过程如下：

Y_1 Y_2 Y_n

级联之前：$a_{11}\ a_{12}\ldots a_{1m}$　　$a_{21}\ a_{22}\ldots a_{2m}$　…　$a_{n1}\ a_{n2}\ldots a_{nm}$

级联之后：$a_{11}\ a_{12}\ldots a_{1m}a_{21}\ a_{22}\ldots a_{2m}\ldots a_{n1}\ a_{n2}\ldots a_{nm}$

4.4 改进的适应值选择策略

为了更好的评估适应度函数的性能，本文改进了适应值的选择策略。首先根据个体间适应值的近似性，把群体分成不定的几个组，在各个组中执行轮盘赌。然后用选中组内的个体相互作用得到新的染色体。因此，新染色体与被选中的组有一定的关联，但和组内的个体又存在区别，有效提高了适应值选择的精度。改进后的选择策略如下：

(1) 首先依据染色体适应值的高低对种群中的个体排序；

(2) 把这些个体平均分到不同的组，每组有 d 个个体。这样，排序完的第 i 个个体被分到第 j 组，$j=1+(\text{int})i/d$。所以，第 1 组个体的序号为$(1,2,3,...d)$，第 2 组个体的序号为$(d+1,d+2,...,d+d)$，依次类推；

(3) 分完组后，每个组都参与轮盘赌，第 j 组的概率为$c(j)=1/j$，使用轮盘赌方法每次选出一个组，轮盘赌执行的次数为种群大小；

(4) 选中一个组以后，必须由该组产生一个新的个体。

当组内相互作用获得新染色体的时候，要从两种情况详细考虑：

(1) 选中的分组排名靠前，此时分组中的每个染色体一般都比较好，但不够精确，要从小范围开始寻找。所以，针对排列在 c 之前的分组 j，采用公式(4-1)得到它每一维的中心，当中 x[i][k]表示群体里的第 i 个染色体的第 k 个基因；接着用公式(4-2)得到它每一维的标准差，公式(4-2)里加上 0.01 目的是若标准差是 0 时，仍旧具有搜索能力；末尾以每一维的中心 mean[j][k]为中心，用 std[j][k]作标准差，得到一个与高斯分布一致的数。结果获取的数与这组的个体有关，如果组里个体相互区别小，那么标准差小，产生的数出现于该组的中心附近机率大；假如个体相互区别较大，产生的数就会和这组的中心区别相对大，满足一种自适应的搜索。有时高斯分布产生的数也许不确信比原先的个体好，那么，在获得一个新个体之后，会评价新个体的适应度，假设较该组最好的个体更好，那么新产生的个体就会遗传到下一代群体当中，否则选取该组最好的个体保留到下一代群体；

$$mean[j][k]=\frac{\sum_{i=j*d+1}^{(j+1)*d}x[i][k]}{d} \tag{4-1}$$

$$std[j][k]=\sqrt{\frac{\sum_{i=j*d+1}^{(j+1)*d}(x[i][k]-mean[level_k][j])^2}{d}}+0.01 \quad (4\text{-}2)$$

(2) 针对位置在 c 后面的分组，组里的个体通常相对差，且个体间区别也相对大，所以，在此种条件下就在该组里任意选取一个个体保留到下一代种群。

4.5 GenerateData 算法设计实现

4.5.1 基本定义

程序结构的控制流图及控制依赖图的定义如下：

(1) 程序结构可用控制流图 CFG=(N,E,entry,exit)表示，其中 N 是结点集，表示语句，E 是边集，表示语句间可能的控制流向，entry 是唯一的源结点，对应程序的开始语句，exit 是唯一的汇结点，对应程序的终止语句。

(2) 在 CFG 中，$\forall$ N_i, N_j $\in$N，若存在一条路径 P= N_i,e_1,e_2,...,e_m, N_j，则称 e_1,e_2,...e_m 为从 N_i 到 N_j 的直接路径。

(3) 在 CFG 中，$\forall$ N_i, N_j $\in$N，若从 N_i 到结点 exit 的每条直接路径(不含 N_i 和出口)均包含 N_j，则称 N_i 被 N_j 后支配。

(4) 在 CFG 中，$\forall$ N_i, $N_j$$\in$N，若存在一条从 N_i 到 N_j 的直接路径，且路径中所有结点 W(除 N_i 和 N_j 外)都后支配于 N_j 且 N_i 不后支配于 W，则称 N_j 控制依赖于 N_i。

(5) 程序结构的控制依赖图CDG用有向图G=(N,P,E,entry, exit)表示，其中N表示语句结点，用椭圆表示，P表示谓词结点，用方形表示，E表示结点间的控制依赖关系，entry是唯一的虚拟源谓词结点，exit是唯一的虚拟汇结点。

(6) 按深度优先搜索遍历 CDG，得到覆盖谓词结点集 W={W_i,...,W_j}，且 $\forall$ Wi$\neq$Wj，称为控制依赖图谓词路径 CDGPath[43]。

如图 4-4 实例程序，其控制流图(CFG)，控制依赖图(CDG)如图 4-5 所示。从中可看出结点 6 后支配除自身和 exit 外的所有结点，结点 3，4 和 5 不后支配任何结点，结点 2 后支配 1 和 entry，结点 1 后支配 entry。结点 4 控制依赖于 3T，结点 5 控制依赖于 3F。结点 1，2，6 控制依赖于 entry。

```
Program Example
      integer i,j,k
      read i,j,k
      if(i<j)
        if(j<k)
          i=k;
        else
          k=i;
        endif
      endif
      print i,j,k
end Example
```

图 4-4 程序实例

Fig.4-4 program instance

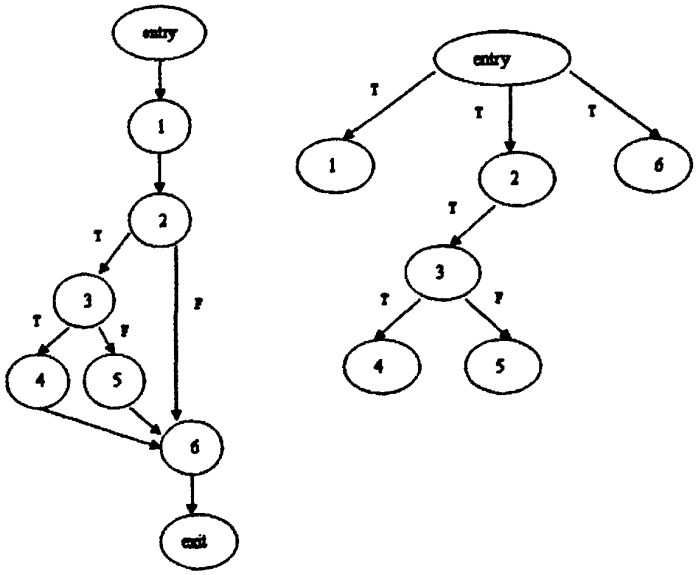


图 4-5 左面 CFG 图，右面 CDG 图

Fig.4-5 CFG on the left, CDG on the right

4.5.2 算法描述

GenerateData 使用遗传算法搜索满足测试需求的测试用例。遗传算法仅仅依靠适应值的高低而选择下一代个体而导致局部搜索能力较弱的特点，在遗传算法中加入了4.4描述的适应值选择策略。它可以有效克服这个问题，提高GA的精度。通过这种方式而选择出的新的一代，更能体现适应度选择的精度。

一个解决方法（染色体）是一组测试数据，比如说一组输入值。算法把测试数据做为输入值执行程序来凭估数据的好坏。记录执行这些测试数据时程序里的谓词标号。这组谓词与结点控制依赖谓词路径中的谓词相比较，结点表示当前测试需求也就是搜索目标。一组测试用例的适应度依赖于谓词数量。它与控制依赖谓词目标路径有关联：适应度函数通过衡量程序的控制依赖图中的两条路径之间的相似程度来计算适应度值。GenerateData使用适应值选择下一代测试用例。使用交叉，变异去产生新一代测试种群。

算法 GenerateData
输入 Program:要测试的程序
CDG:程序的控制依赖图
Initial:测试用例初始种群
TestReq:一组（测试需求，标记）
输出 Final:程序P的测试用例集
TestReq:一组（测试需求，标记）
声明 CDGPaths:CDG中的一组谓词路径
Scoreboard:满足测试需求的记录
CurPopulation,NewPopulation:一组测试用例
Target:产生的测试用例符合测试需求
MaxAttempts():当试图为一个Target超过了最大数目，返回true的函数
OutOfTime():当超过时间限制时，返回true的函数。否则返回false
Begin
Step1:初始化设置
[1] 建立 CDGPaths
[2] 建立初始化 Scoreboard
[3] 建立 CurPopulation
Step2:产生测试用例
[4] while((some(r,unmarked)∈TestReq) and not OutOfTime()) do

[5] 从 TestReq 里选择没有标记的 Target
[6] While(Target not marked and not MaxAttempts() do
[7] 使用 CDGPaths，针对指定适应度函数计算当前种群的适应值
[8] 根据适应值排序 CurPopulation
[9] 依照改进的选择策略，取 d=2，c=4 作为测试参数，选择 NewPopulation 的父辈
[10] 从当前测试用例 CurPopulation 产生 NewPopulation
[11] 用 NewPopulation 执行程序
[12] 根据是否满足测试需求更新 Scoreboard and mark TestReq
[13] endwhile
Step 3:清空并返回
[15] Final=满足 TestReq 的测试用例
[16] return (Final,TestReq)
End

GenerateData 使用的遗传算法是典型的，因为算法不是搜索一个单一的优化方案，而是有多个搜索目标-每一个测试需求都是一个搜索目标。因此，贯通整个搜索过程，GenerateData的目标依据测试需求是否满足而变化。尽管选定的适应度函数保持不变，每一个测试用例必须依照新的被覆盖的测试需求被重新评估。

GenerateData选取下列做为输入：

Program，辅助测试程序；CDG，程序控制依赖图；Initial，一组初始化的测试数据，有可能是空的；TestReq，一组需要被满足的测试需求。使用局部变量，CDGPath，Scoreboard，CurPopulation，NewPopulation 和Target函数，MaxAttempts()和OutOfTime()，GenerateData输出：Final，一组测试数据和TestReq，满足测试需求（被标记）的列表。

4.5.2.1 GenerateData的第一步 GenerateData的第一步是初始化和预处理步，包括计算CDGPaths，Scoreboard的初始化。首先，对于程序中的每一条语句GenerateData产生CDGPaths，即包括谓词结点的非循环的路径，该谓词结点位于CDG语句结点和entry结点之间。对于语句和分支覆盖，Scoreboard可以是一个bit vector，任何时候当相应的语句或分支被一组测试用例满足的时候，一个位bit（初始值为false）被设定。如果要知道语句或分支的执行频率，Scoreboard可以是一个integer的vector，其中每一个元素初始化为零，当相应的语句或分支被一组测试数据覆盖时即增加。最后，GenerateData通过把Initial加

到CurPopulation来产生CurPopulation。如果测试用例需要较高的多样性，算法随机产生由用户提交的约束的测试用例，把它们加到CurPopulation。种群的多样性给Generatedata一个广泛的各种不同的可选择的测试用例，使得算法具有较高的性能。

4.5.2.2 GenerateData第二步 GenerateData的第二步是一个外层的While循环，用于产生测试用例。对于路径覆盖，所有测试需求被标记(mark)，或者OutOfTime()返回真值，否则while loop一直迭代下去。尽管搜索测试用例的时间限制并不能证明没有被标记的测试需求是不可行的，但是的确暗示了这种可能性。在GenerateData执行后，所有没有标记的测试需求必须被手工检查。假如一个产生的测试用例覆盖了一个测试需求或者测试需求被认为是不可行的，则该测试需求被标记。内层的while loop作用于一个指定的没有标记的Target，它是在TestReq中没有被标记的测试需求中选择的。当Target被一组测试用例满足，或者MaxAttempts()返回true，while loop迭代结束。GenerateData使用适应度函数在Curpopulation中评价每个与Target相关的测试用例，计算测试用例的适应值。

在计算CurPopulation中每个测试用例的适应值后，GenerateData依照4.4介绍的选择策略从CurPopulation中选择测试用例做为NewPopulation的父代(line 9)；经过交叉变异，使种群多样性得到提高。因为该操作是快速、简单的，他们可以被执行多次，增加Target实际被覆盖的机会。

在内部while loop的最后一步，GenerateData用Newpopulation(line 11)中的每个成员执行Pragram，并相应更新Scoreboard。假如CurPopulation中至少有一个测试用例满足Target，算法标记TestReq中的Target，内部while loop终止，算法尝试找到一个新的Target。否则，NewPopulation被赋予到CurPopulation，算法再一次试图满足Target。假如GenerateData在MaxAttempts()之前没有找到满足Target的测试用例，算法暂时放弃目标，选择下一个测试需求做为新的Target。整个搜索继续直到所有的测试需求被满足或者超过时间限制，这时外层while loop终止。

4.5.2.3 GenerateData的第三步 在这最后一步，算法分派所有满足TestReq的测试用例集合到Final(line 15)，并且返回Final和TestReq(line 16)。

算法的时间复杂度主要由initialization，set up(lines 1-3)和产生测试的loop循环(lines 4-14)决定。Initialization 和 set up产生CDGPaths，需要程序的控制依赖图($O(n^2)$)，Scoreboard(O(n))，初始种群Curpopulation的产生(O(p))。因此，第一步的时间复杂度$O(\max(n^2, P))$。生成测试评估在Curpopulation中的适

应度值的循环，产生NewPopulation，更新Scoreboard，这些时间复杂度都是O(P)，loop对CurPopulation进行排序(O(PlogP))并且对每组测试执行程序(O(P×e))。因此，每个内部循环的叠代的时间复杂度是O(max(P×e，PlogP))。内部循环次数的范围从1到最大m。因此内部循环总的时间复杂度是O(m×max(P×e，PlogP))。

外部循环(lines 4-14)执行直到所有的测试需求被标记或者超过时间限制。也就是说外部循环迭代0次或多次，随运行的不同而变化。时间限制由用户在步骤2的执行时间中设置上限。

GenerateData的空间复杂度取决于它使用的主要结构。CDGPath是O(n×k)，因为k是个常数，通常不大与7，所以空间复杂度为O(n)。CurPopulation，NewPopulation 是O(P×L)；ScoreBoard 是O(r)。对于路径覆盖，r是要覆盖的路径数，因此，对于路径覆盖，Scoreboard的增长由产生的路径数决定。

4.6 本章小结

本章以控制流图为基础分析测试用例与执行路径的关系，改进适应值的选择策略。新的策略把群体分成不定的几个组，在各个组中执行轮盘赌。在选中的组中，由该组内的个体综合作用产生新的个体。使用改进遗传算法设计测试用例自动生成系统，利用改进选择策略提高适应度值选择的精度，有效评估适应度函数的性能。

第5章 试验和分析

为了验证3.3节中适应度函数的性能，设计基于Java语言的测试用例产生系统GenerateDate，该系统以遗传算法为核心算法，并针对不同的路径覆盖标准进行实验。

该系统的工作流程如下：首先，用户从界面选择要测试的程序路径及适应度函数，然后通过命令开始生成测试数据。由随机输入生成器根据提取出的路径上的变量及其定义域随机生成一组输入，驱动被测程序运行。被测程序运行时计算出的评价函数值返回给遗传算法包，算法包据此来评价种群中每个个体位串的优劣，并通过遗传算子（选择、杂交、突变）的操作改变个体位串的结构，形成新一代更优种群，如此往复，直至超过了Gas的限制。

5.1 程序插装

实验使用了一个简单的有限搜索空间的程序来评估适应度函数。三角形程序(TRITYP) 是测试中的典型示例，问题描述为：接收3个数A、B 和C 输入为三角形的边，根据边的关系输出三角形类型。因为该程序的路径容易统计，共14条路径并且一些路径比其它的路径更容易被覆盖，在众多的软件测试研究中被作为一个基准程序来使用。例如，三条输入边的取值范围是[0...1023]，只有1023个可能的组合满足等边三角形的需求。

根据指定的逻辑路径对被测试程序进行插桩，在程序单元内部指定的逻辑路径所经过的每个分支点前插入一组谓词标号指示被执行的路径，当一组测试用例驱动被测单元执行时，能够为适应度函数产生一组谓词串。本实验是在每个条件分支谓词后放置一个代表所执行路径的字符串，若该判定条件为真，则路径标号获得一个代表真值的字母T，否则该标号为F。通过一个布尔型数组path来记录执行过程是否执行过此分支。如果程序执行过此分支，则数组path中相应位置的值为T，否则为F。这样对不同的路径，数组path有不同的值。

5.2 试验结果分析

研究中，对GAs实行两种限制，第一，近化代数达到200000代，GAs停止产生测试用例。第二，在程序执行性能超过100000仍没有找到指定的路径，

GAs终止。表5-1是种群数量为80，染色体长度为10，交叉概率为0.80，变异概率为0.03，分别使用六种不同适应度函数的实验结果。

表5-1 TRITYP路径的执行性能

Table 5-1 Performance executions by TRITYP path

	TT1	TT2	TT3	TTAll
BP1	751	42436	42967	70060
BP2	301	9315	40168	13210
NEHD	3300	3346	31144	118575
IPP	3231	5369	21586	32099
DO	2142	22603	46790	145558
DO-NEHD	3280	44916	35397	141739

表中TT1，TT2，TT3，TTALL分别表示非三角形，等边三角形，直角三角形以及所有可能路径。表的值是计算的均值。例如，左上角的单元说明使用适应函数BP1产生覆盖路径TT1的测试用例所需要的执行性能是751。表中实验结果表明各种方法的性能不同，在特定路径上分支谓词法，IPP这两种适应度函数的性能优于其它的。

5.2.1 TriTyp 路径 1T

在一组输入(x,y,z)集合中，单一的分支路径 TT1（1T）搜索成功率为0.979%。因为它是一个非常简单和容易找到的路径，所有的方法都连续完成每一次运行。适应度函数性能结果见表 5-2。

表 5-2 路径 TT1 结果

Table 5-2 Path TT1 results

	BP1	BP2	NEHD	IPP	DO	DO-NEHD
Mean	857	335	3667	3590	2381	3645
Median	630	275	1723	2325	1366	1177
Standard deviation	810	234	4456	4347	2968	5288
Completed runs	100	100	100	100	100	100

表 5-3 means 两两比较-TT1

Table 5-3 Pairwise comparison of means-TT1

	BP2	NEHD	IPP	DO	OD-NEHD
BP1	521	2810*	2733*	1524*	2879*
BP2		3332*	3255*	2046*	3310*
NEHD			77	1286*	22
IPP				1209*	55
DO					1264*

表 5-3 的值显示测试产生方法在 mean 性能执行中成对比较的不同，星号代表差异较大，因为我们感兴趣的是最小的执行性能，成功产生该路径的测试用例的适应度函数是：适应度函数 BP1，BP2。因为 TT1 是单一的分支测试，这两种函数都鼓励输入参数的距离。

5.2.2 TriTyp 路径 2F3F4F5T6T

路径 TT2 是判别任意三条边是否能在指定范围内组成等边三角形。只占用了非常小的搜索空间(0.000085743%)。测试结果见表 5-4，'Completed runs'指的是 GA 按照路径要求成功搜索的运行次数。从可靠性角度来说，分支谓词和 IPP 性能好，每次都找出满足所需路径覆盖的测试用例。DO 也是非常可靠的，只有两次没有成功。从效率的角度来看，分支谓词和 IPP 性能也是居前的，它们仅需要最少的性能去找到路径。尽管 DO 只有两个未完成的运行，但是它占用了很大的性能去寻找路径。NEHD 没有完成的 Completed runs 有一半，DO-NEHD 超过了代数的限制，在 100 次当中只完成了 30 次。

表 5-4 路径 TT2 结果

Table 5-4 Path TT2 results

	BP1	BP2	NEHD	IPP	DO	DO-NEHD
Mean	42436	3346	10350	5369	22603	44916
Median	42136	2905	7053	3826	15429	46865
Standard deviation	26400	2022	8699	4639	18703	24626
Completed runs	98	100	56	100	98	30

表 5-5 means 的两两比较-TT2

Table 5-5 Pairwise comparison of means-TT2

	BP2	NEHD	IPP	DO	OD-NEHD
BP1	36802*	43806*	41186*	22037*	2755
BP2		7004	4383	14765*	39557*
NEHD			2617	21769*	46560*
IPP				19149*	43941*
DO					24791*

这种情况可能的解释是，GA 产生测试用例的种群被一些测试用例所主导，这些测试用例是搜索近似指定路径的，覆盖了大量的搜索空间，但不是预期指定的路径。这种结果说明了 BP1 的问题，类似这样的测试用例，恰巧是两个路径之间一个结点差异，占主导的路径和与指定路径有更多共同结点的路径一样都被鼓励。发生这种情况是因为 BP1 适应度函数的惩罚功能使用了一个等级机制去决定惩罚的大小。NEHD 和 IPP 允许更多情况下分化，这样往往更早的得到解。

5.2.3 TriTyp 路径 2F3F4F5F7F8F9T

路径 TT3 是判断是否为直角三角形。覆盖这条路径仅占用搜索空间的 0.000169%。整个测试性能结果见表 5-6。

IPP 是在实验中是唯一可信赖的适应度函数。覆盖路径达 96%，平均产生少于 22000 个测试用例。而其它的适应度函数大部分都没有成功。IPP 在这条路径有较好的性能是由于它不鼓励搜索已发现的路径。BP1 和 BP2 却都受到了这样的困扰。因为输入的值对距离测量产生了太多的影响。比如说，输入 x，y，z 分别为 2，3，4，距离度量是 3（|22+33-44|=3），3，4，6 的距离度量是 11（|33+44-66|=11），一个较低的距离度量是比较好的，但是第二个输入测试用例却与成功的测试用例（3，4，5）只有一个数字的距离。

表 5-6 路径 TT3 结果

Table 5-6 PathTT3 results

	BP1	BP2	NEHD	IPP	DO	DO-NEHD
Mean	38670	36070	31144	21586	46790	35397
Median	37719	33419	29733	15530	44001	27854
Standard deviation	21988	25187	25467	18432	27067	28523
Completed runs	42	35	12	96	11	15

5.2.4 路径 ALLTT

最后一组 TRITYP 运行被设计去找到程序中所有可能的路径。为了做到这一点，搜索从列表中的第一条路径开始；然而，正如产生的测试用例覆盖的其它路径，那些路径被记录下来。这一过程允许偶然的路径覆盖。当每一条都被覆盖，函数继续发现尚未被覆盖的路径，直到所有 14 条路径被覆盖，达到了最大的 200000 性能执行。

IPP 又一次是最可靠的测试，完成了 100 次运行中的 95 次，平均发现了 13.91 个路径。BP2 虽然只完成了 60%，但是 BP2 仅用了比 IPP 还少的测试用例成功找到了 14 条路径，见表 5-7。

图 5-1 为 GenerateData 设置适应度函数为 NEHD 在路径 TT2 上的运行结果。

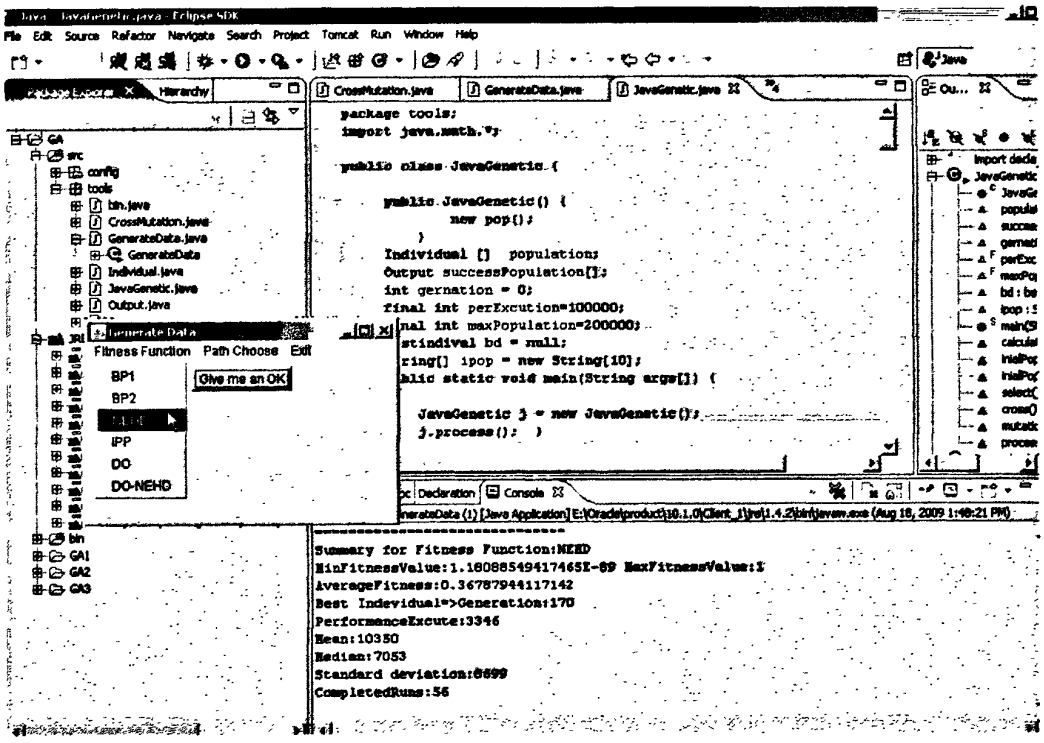


图 5-1 系统输出

Fig.5-1 System out

表 5-7 路径 AllTT 结果

Table 5-7 ALLTT results

	BP1	BP2	NEHD	IPP	DO	DO-NEHD
Mean	77845	15046	113861	35666	161732	157468
Median	45426	4789	115150	20748	161732	157468
Standard deviation	58085	23598	49851	36962	0	31752
Completed runs	11	60	13	95	1	2

5.3 本章小结

本章考察了几种适应度函数，理论分析与各项测试都表明，使用适应度函数分支谓词和 IPP 的性能比其它的几种适应度函数均有提高，在基于路径测试的遗传算法中，使用这两种适应度函数计算方法是最有效的。下一步的工作是如何结合这两种函数，构造新的适应度函数。

结论

软件测试是保证软件质量的重要手段，路径测试数据生成问题是软件测试中的一个基本问题。主要对路径测试用例的自动生成方法进行了研究。

本文主要针对遗传算法的不同适应度函数计算方法，提出了对面向路径的适应度函数进行评估，采用改进的选择策略以提高 GA 的精度，给出了算法设计和实验分析，总结如下：

(1) 分析了面向路径的适应度函数，构造两种改进的适应度函数，提出对适应度函数进行综合评价。

(2) 对遗传算法的适应度的选择做了改进工作，使得其可以根据组合个体的情况生成与之接近的个体作为交叉、变异操作的下一代群体，这样既保证了个体的多样性，同时也增强解的空间搜索能力。

(3) 以控制流图为基础分析测试用例与执行路径的关系，一组测试用例的适应度依赖于谓词数量。它与控制依赖谓词目标路径有关联。设计测试用例自动生成系统，给出算法。

(4) 以软件测试研究中基准程序 TRITYPE 作为被测程序，来实现测试数据的生成过程。设计了一组实验评估面向路径的测试用例自动生成中适应度函数的性能，6 种适应度函数相互比较。结果表明个体的评价即适应度的计算在遗传算法中起着重要的作用，给出了适应度函数改进的方向。

本文在路径测试数据生成算法研究方面还有一些工作需要做：

(1) 把 IPP 与分支谓词这两种适应度函数结合起来使用，是下一步适应度函数改进所需要做的工作。

(2) 对于复杂的数据类型的参数编码需要进一步研究其分析和处理方法。

参考文献

[1] ELFRIEDE DUSTIN，JEFF RASHKA，JOHN PUAL. Automated software testing-introduction、management and performance[M]. 北京：清华大学出版社，2003：20-25.

[2] W T TSAI，D VOLOVIK，T F TKEEFE. Automated test case generation for program specified by relational algebra queries[J]. IEEE trans on Software Engineering，1990，16(3)：316-324.

[3] E WEYUKER，T GORADIA，A SINGH. Automatically generating test case data from a boolean specification[J]. IEEE Trans on Software Engineering，1994，20(5)：353-363.

[4] BIRD D，MUNOZ C U. Automatic generation of random selfchecking test cases[J]. IBM Systems Journal，1983，22(3)：229-245.

[5] RAMANMOORTHY C V，HO S B F，Chen W T. On the automated generation of program test data[J]. IEEE Transactions on Software Engineering，1976，SE-2(4)：293-300.

[6] DEMILLO R A，OFFUTT A J. Consrtaint-based automatic test data generation[J]. IEEE Transactions on Software Engineering，1991，17(9)：900-910.

[7] L CLARKE. A system to generate test data and symbolically execute programs[J]. IEEE Transactions on Software Engineering，1976，SE-2：215-222.

[8] BOGDAN KOREL. Automated software test data generation[J]. IEEE Transactions on Software Engineering，1990，16(8)：870-879.

[9] 奚红宇，徐红，高仲仪. Ada 软件测试用例生成工具[J]. 软件学报，1997，8(4)：297-302.

[10] GALLAGHER M，NARASIMHAN V L. ADTEST：A test data generation suite for Ada software system[J]. IEEE Transactions on Software Engineering，1997，23(8)：473-484.

[11] GUPTA，MATHUR A P，SOFFA M L. Generation test data for branch coverage[C]. Proceedings of the 15th IEEE International conference on Automated Software Engineering，Grenoble，2000：219-227.

[12] 单锦辉，王戟，马晓冬等. 面向路径的测试数据自动生成工具及其图形界

面 Tcl/TK 设计[J]. 计算机工程与应用，2002，1：74-77.

[13] BORGELT K . Software test data generation from a genetic algorithm[M]. Industrial Applications Of Genetic Algorithms. CRC Press 1999：49-68.

[14] LIN J，YEH P. Automatic test data generation for Path Testing using Gas[J]. Information Sciences，2001，131：47-64.

[15] JONES B F，EYRES D E，STHAMER H H. A strategy for using genetic algorithms to automatic branch and fault-based testing[J]. The Computer Journal，1998，41(2)：98-107.

[16] WEGENER J，BARESEL A，STHAMER H. Evolutionary test enviroment for automatic structural testing[J]. Information and Software Technology，2001，43(14)：841-854.

[17] 汪浩，谢军凯，高仲仪. 遗传算法及其在软件测试数据生成中的应用研究[J]，计算机工程与应用，2001，37(12)：64-68.

[18] 荚伟，谢军凯，奚红宇等. 遗传算法在软件测试数据生成中的应用[J]. 北京航空航天大学学报，1998，24(4)：434-437.

[19] 景志远. 遗传 K-均值算法在软件测试算例自动生成中的应用研究[J]. 油气田地面工程，2003，22(4)：15-16.

[20] 傅博. 基于模拟退火遗传算法的软件测试数据自动生成[J]. 计算机工程与应用，2005，12：82-84.

[21] PRAVEEN RANJAN SRIVASTAVA，PRIYANKA UPTA，YOITA ARRAWATIA et. al. Use of Genetic Algorithm in Generation of Feasible Test Data[J]. SIGSOFT Software Enineering Notes，2009 (34)：1-4.

[22] JOSE CARLOS，MARIO ALBERTO，RANCISCO. A strategy for evaluating feasible and unfeasible test cases for the evolutionary testing of object-oriented software[C]. Proceedings of the 3rd international workshop on automation of software test，Leipzig，Germany，2008：85-92.

[23] 袁玉宇. 软件测试与质量保证[M]. 北京：北京邮电大学出版社，2008:88-98.

[24] J EDVARDSSON. A survey on automatic test data generation[C]. In Proceedings of the Second Conference on Computer Science and Engineering in Linkoping，1999，21-28.

[25] PAUL C，JORGENSEN. 软件测试[M]. 韩柯译. 北京：机械工业出版

社，2005：57-58.

[26] 张海藩．软件工程导论[M]．北京：清华大学出版社，2000：139-143.

[27] 乐鑫喜．基于退火遗传算法的测试用例自动生成[D]．武汉：武汉理工大学，2005：21-32.

[28] HOLLAND J H. Adaptation in Nature and Artificial Systems[M]. MIT Press，1992：2-18.

[29] GOLDERG D E. Genetic Algorithms in Search，Optimization and Machine Learning[M] . Addison-Wesley，1989：20-94.

[30] DE JONG K A. Aanalysis of the behavior of a class of genetic adaptive Systems[D]. Michigan：University of Michigan，No. 76-9381，1975：20-64.

[31] 王鹏．基于改进遗传算法的面向路径测试用例自动生成方法研究[D]．大连：大连交通大学，2006：20-29.

[32] 陈有青，徐蔡星，钟文亮，等．一种改进选择算子的遗传算法[J]．计算机工程与应用：2008，44(2)：44-49.

[33] ZHUO XIAO LAN，LIN YING，ZHAN JUN. Comparison of selection strategy in genetic algorithm[C]. Proceedings of the 12th Chinese Automation & Computing Society Conference in the UK，Loughborough，England，2006：245-250.

[34] 姚尧．一种基于遗传算法的软件测试用例生成新方法[J]．计算机与数字工程，2009，37(1)：18-21.

[35] 金虎．基于面向路径的遗传算法的测试用例自动生成[J]．计算机工程，2007，33(3)：22-23.

[36] 孙亚娟．基于遗传算法的路径测试数据自动生成方法研究[D]．河北：河北工业大学，2006：30-40.

[37] PAULO MARCOS SIQUEIRA BUENO，MARIO JINO. Automatic test data generation for program paths using genetic algorithms[J]. International Journal of Software Engineering and Knowledge Engineering ，2002，12(6)：691-709.

[38] MICHAEL CC，MCGRAW G，SCHATZ MA. Generating software test data by evolution[J]. IEEE Transactions On Software Engineering 2001，27(12)：1085-1110.

[39] MANSOUR N，SALAME M. Data generation for path testing[J]. Software

Quality Journal 2004，12：121-136.

[40] WATKINS A. The automatic generation of software test data using genetic algorithms[C]. Proceedings of the Fourth Software Quality Conference，Scotland，1995：300-309.

[41] D BERNDT， L JOHNSON，J PINGLIKAR. Breeding Software Test Cases with Genetic Algorithms[C]. Proceedings of the 36th Hawaii International Conference on System Sciences，2002：338-348.

[42] R P PARGAS，M J HARROLD，R R PECK. Test-Data Generation Using Genetic Algorithms[J]. The Journal of Software Testing，Verification and Reliability ，1999：263-282.

[43] 伦立军，丁雪梅，李英梅. 基于遗传算法的测试数据生成研究[J]. 计算机工程，2005，31(21)：82-84.

攻读学位期间发表的学术论文

[1] 刘双悦，王培东．一种基于改进遗传算法的面向路径测试用例自动生成方法[J]．自动化技术与应用，2010，3（已录用）

致谢

首先真诚感谢我的导师王培东教授，他不但在选题和课题研究方面，资料查询方面给予了我莫大的帮助和指导，而且在生活方面给予了我很大的关心和支持。在攻读硕士期间，王老师更以严谨的治学态度和对学术的孜孜以求精神始终感染和激励着我，使我能够以饱满的热情和端正的态度去研究课题相关的理论。同时支持我走出实验室在北京捷图公司加以锻炼，在担任软件性能测试的工作中，使我的理论知识得以在实际工作中能够充实完善；在论文的完成阶段，王老师更是细致地审阅，使我最终能够顺利地完成毕业论文的工作。我相信，这两年多的研究生学习生活，将是我一生中最难以忘怀的记忆。在此，请允许我向尊敬的导师王培东教授致以最崇高的敬意和衷心的感谢！

感谢哈尔滨理工大学对我的培养，为我提供了参加专业知识论坛和讲座的机会，是她使我在良好的文化氛围内得到了锻炼；感谢计算机科学与技术学院的各位老师对我在学习和生活上的支持；同时也感谢图书馆的老师们为我提供了很好的阅读空间和时间，让我能及时的找到所需的资料，以及实验室各位同学们，与他们的探讨，使我开阔了思路，获得了灵感。

感谢我的家人和众多关心我成长的亲友，是他们无微不至的关心、爱护和支持，一直伴随着我，使得我的学习与工作能够顺利进行。