

信息科学技术专著丛书

基于人工智能的测试用例自动生成与测试用例集优化

邢 颖 宫云战 于秀丽 著

JIYU RENGONG ZHINENG DE CESHI YONGLI
ZIDONG SHENGCHENG YU CESHI YONGLIJI YOUHUA

非外借

北京邮电大学出版社
www.buptpress.com



信息科学技术专著丛书

基于人工智能的测试用例自动生成与测试用例集优化

邢　颖　宫云战　于秀丽　著



北京邮电大学出版社
www. buptpress. com

内 容 简 介

作为软件测试(包括白盒测试和黑盒测试)中的一个基本问题,测试用例自动生成尤为重要,这是因为白盒测试中的许多问题(如控制流测试和数据流测试)以及黑盒测试中的一些问题都可以归结为测试用例生成问题。解决这个问题的本质在于约束系统的建立和求解。约束求解是人工智能的一个传统研究方向。本书将系统地研究如何进行软件系统的约束建模和求解,利用人工智能的各种技术,对一些特殊情况(复杂数据类型、线性约束的区间初始化、库函数等)给出切实可行的解决方案。

提升回归测试效率的一个重要方法是对测试用例集进行优化,目前常见的优化方法有3种,分别是测试用例集约简、选择和优先级排序。这3种方法分别适用于不同的场景,本书主要关注测试用例集约简和测试用例优先级排序,通过对相关问题和已有方法的调研,将现在应用比较广泛的人工智能中的群智能算法和一些进化算法引入测试用例集优化问题中,提出新的测试用例集约简和测试用例优先级排序技术。

本书的主要读者对象为软件工程研究者和从业人员。

图书在版编目(CIP)数据

基于人工智能的测试用例自动生成与测试用例集优化/邢颖,宫云战,于秀丽著. -- 北京 : 北京邮电大学出版社,2022.12

ISBN 978-7-5635-6670-9

Ⅰ. ①基… Ⅱ. ①邢… ②宫… ③于… Ⅲ. ①软件—测试—研究 Ⅳ. ①TP311.55

中国版本图书馆CIP数据核字(2022)第115633号

策划编辑:彭 楠 **责任编辑**:刘春棠 **责任校对**:张会良 **封面设计**:七星博纳

出版发行:北京邮电大学出版社

社 址:北京市海淀区西土城路10号

邮政编码:100876

发 行 部:电话:010-62282185 传真:010-62283578

E-mail:publish@bupt.edu.cn

经 销:各地新华书店

印 刷:唐山玺诚印务有限公司

开 本:787 mm×1 092 mm 1/16

印 张:14.5

字 数:333千字

版 次:2022年12月第1版

印 次:2022年12月第1次印刷

ISBN 978-7-5635-6670-9 定 价:68.00元

·如有印装质量问题,请与北京邮电大学出版社发行部联系·

前　言

随着计算机技术的飞速发展，计算机已经应用到国民经济和社会生活的方方面面，随之而来的是计算机系统的规模和复杂性急剧增加。软件作为计算机系统的灵魂，其规模和复杂性也以惊人的速度发生着变化。软件质量问题已成为制约计算机发展的主要因素之一。许多计算机科学家在展望计算机科学发展方向和策略时，都把提高软件质量放在优先于提高软件功能和性能的地位。

一些常见操作系统的代码达到百万行乃至上千万行，而大规模软件中的缺陷更是难以避免。据统计，在编译和传统的软件测试之后，平均每千行软件代码中包含 10～20 个缺陷。由软件缺陷引起的失效和故障轻则给用户带来不便，重则造成巨大的经济损失，甚至生命财产损失。

虽然已经出现了实用的形式化方法和程序正确性证明技术，但软件测试在今后较长时间内仍将是保障软件质量的重要手段。软件测试在整个软件开发周期中所占比重很大，据统计，软件测试的开销占所有开发成本的 50%，甚至更多。单元测试作为软件测试中的重要环节，是指对软件中的基本组成单位进行测试，其目的就是完整检测代码单元的功能逻辑，找出代码单元本身的所有功能逻辑错误。单纯的人工单元测试通常会花费大量的人力和时间，且不能保证准确性。对于上述提到的大规模软件来说，人工测试几乎是不可能的。而测试用例自动生成技术可以节省测试时间，拓展测试人员的能力，降低开发成本，在软件测试领域发挥着不可替代的作用。

在软件测试开始时，测试用例生成器可能会自动生成大量测试用例。同时，在开发的过程中，软件的代码会不断改进和增加，因此需要执行其测试用例来检查新代码中是否产生了错误。软件开发人员可以通过删除冗余测试用例来减小测试用例集大小，或者通过排序的方法将检测出故障可能性大的测试用例率先运行，同时仍然确保满足所有测试要求，这样测试效率可能会更高，并且测试的有效性也不会受到损害。

本书作者及其团队一直致力于软件测试方面的研究，在本书所涉及的软件测试和人工智能相关方面做了大量研究，为本书研究工作的开展奠定了扎实的学术研究基础。作者之前曾经出版过关注程序分析的专著《源代码分析》(科学出版社，2018 年出版)。比较而言，《源代码分析》更加侧重程序分析和软件工程本身，而本书更加偏重各种人工智能技术在其中的应用，即关注学科交叉的内容。而在此学科交叉领域，本书是国内外第一本如此细致地介绍测试用例相关内容的书籍，因而具有创新性和启发性。

感谢王兴德和王启明在本书撰写过程中给予的大力支持，感谢张旭舟、张博、白玉、徐健豪、李晋忠等对本书内容所作的贡献。

限于作者水平，书中难免存在不足之处，敬请读者批评指正。

作　者

2022 年 2 月

目　　录

第 1 章　软件测试 …… 1

1.1　软件系统开发的可靠性问题 …… 1
1.2　回归测试的必要性 …… 2
1.3　单元测试概述 …… 4
1.4　静态测试与动态测试 …… 4
1.5　黑盒测试和白盒测试 …… 5
1.6　基于路径和覆盖率的测试 …… 7
1.7　约束求解问题 …… 8
1.8　代码测试系统 …… 8
本章参考文献 …… 10

第 2 章　测试用例自动生成技术 …… 12

2.1　静态测试用例自动生成方法 …… 12
2.1.1　常用的静态分析技术 …… 12
2.1.2　典型的测试用例生成技术 …… 16
2.2　动态测试用例自动生成方法 …… 18
2.2.1　直线式程序法 …… 18
2.2.2　分支函数极小化法 …… 19
2.2.3　ADTEST …… 20
2.2.4　迭代松弛法 …… 20
2.2.5　MHS 方法 …… 20
2.3　动静结合的测试用例自动生成方法 …… 26
本章参考文献 …… 27

第 3 章　基于分支限界的测试用例生成 …… 36

3.1　分支限界算法 …… 36
3.1.1　问题定义 …… 36
3.1.2　解决方案 …… 38
3.1.3　路径约束提取 …… 40

3.1.4 求相关变量集和相关变量闭包 …… 42
3.1.5 变量级别确定算法 …… 44
3.1.6 不相关变量移除 …… 45
3.2 搜索加速算法 …… 48
3.2.1 基于抽象解释的区间迭代优化策略 …… 48
3.2.2 变量动态排序决策机制 …… 52
3.2.3 基于爬山法的求解 …… 55
3.2.4 实例分析 …… 61
3.3 实验 …… 65
3.3.1 迭代的区间运算对比实验 …… 66
3.3.2 变量的动态排序对比实验 …… 68
3.3.3 爬山法实验 …… 69

第 4 章 人工智能在测试用例自动生成中的应用 …… 72

4.1 基于矛盾定位的混合回溯技术 …… 72
4.1.1 背景介绍 …… 72
4.1.2 问题的提出 …… 73
4.1.3 3C 算法 …… 75
4.1.4 算法介绍 …… 79
4.1.5 实验分析 …… 81
4.2 基于优化区间运算的一致性判断算法 …… 87
4.2.1 背景介绍 …… 87
4.2.2 问题的提出 …… 90
4.2.3 理论分析 …… 90
4.2.4 算法描述和实现 …… 91
4.2.5 实验分析 …… 95
4.3 基于神经网络技术的测试用例生成时间预测 …… 103
4.3.1 背景介绍 …… 103
4.3.2 问题的描述 …… 106
4.3.3 数据准备 …… 108
4.3.4 模型的确定与实现 …… 111
4.3.5 实例分析 …… 114
4.4 基于强化学习技术的测试用例自动生成技术 …… 117
4.4.1 背景介绍 …… 118
4.4.2 问题的描述 …… 121
4.4.3 强化学习模型的构建及算法描述 …… 121
4.4.4 实例分析 …… 126
本章参考文献 …… 128

第 5 章　特殊程序结构的测试用例生成 …… 131

5.1　动静结合的循环处理模型 …… 131
5.1.1　背景介绍 …… 131
5.1.2　选择性符号执行 …… 132
5.1.3　优化问题和目标函数 …… 132
5.1.4　启发式引导的 $k+1$ 循环处理模型 …… 133
5.1.5　启发式引导的 $k+1$ 循环处理模型算法概述 …… 138
5.1.6　实例分析 …… 139
5.2　基于代码片段和反向符号执行的不可达路径判定技术 …… 140
5.2.1　背景介绍 …… 140
5.2.2　不可达路径判定问题的重定义 …… 141
5.2.3　不可达路径的特征调研 …… 142
5.2.4　依赖循环的等值代码片段 …… 143
5.2.5　对于代码片段性值的有效性验证 …… 145
5.2.6　基于反向符号执行的可达路径生成技术 …… 146
5.2.7　实例分析 …… 147
5.3　基于原子函数的字符串类型测试用例自动生成技术 …… 148
5.3.1　背景介绍 …… 148
5.3.2　字符串约束描述语言 …… 150
5.3.3　原子函数的等价性 …… 152
5.3.4　$\mathscr{L}_{\text{Atom}}$ 的约束求解过程 …… 153
5.3.5　实例分析 …… 155
5.4　库函数的约束求解策略 …… 156
5.4.1　问题的描述 …… 156
5.4.2　算法描述和实现 …… 158
5.4.3　实例分析 …… 161
本章参考文献 …… 163

第 6 章　测试用例集约简 …… 166

6.1　测试用例集约简概述 …… 166
6.1.1　相关技术研究 …… 166
6.1.2　测试用例集约简的基本概念 …… 171
6.1.3　测试用例集约简的数学建模 …… 171
6.2　基于佳点集萤火虫算法的测试用例集约简 …… 172
6.2.1　萤火虫算法 …… 172

6.2.2 佳点集萤火虫算法 …… 173
6.2.3 测试用例集约简建模 …… 179
6.2.4 实验分析 …… 180
6.3 基于蚁狮优化算法的测试用例集约简 …… 183
6.3.1 蚁狮优化算法 …… 183
6.3.2 实验分析 …… 185
本章参考文献 …… 189

第 7 章 测试用例优先级排序 …… 191

7.1 测试用例优先级排序 …… 191
7.1.1 测试用例优先级排序的定义 …… 191
7.1.2 多目标优化问题 …… 191
7.1.3 多目标测试用例优先级排序 …… 195
7.1.4 测试用例集排序的评测指标 …… 195
7.2 基于人工免疫算法的测试用例优先级排序 …… 196
7.2.1 优化目标选取 …… 196
7.2.2 人工免疫算法 …… 196
7.2.3 实验分析 …… 199
7.3 基于多种群粒子群算法的多目标测试用例优先级排序 …… 202
7.3.1 优化目标选取 …… 202
7.3.2 多种群协同粒子群算法 …… 202
7.3.3 基于多种群粒子群算法的多目标测试用例优先级排序 …… 203
7.3.4 实验分析 …… 206
7.4 基于蝙蝠免疫算法的测试用例集排序 …… 210
7.4.1 优化目标选取 …… 210
7.4.2 蝙蝠算法 …… 210
7.4.3 人工免疫系统 …… 212
7.4.4 多目标蝙蝠免疫算法 …… 214
7.4.5 实验分析 …… 218
本章参考文献 …… 221

第1章

软件测试

1.1 软件系统开发的可靠性问题

在20世纪70年代以前，软件对于大多数人来说，还是个新鲜名词。人们只能在政府机构、科研院所、军事机构以及大型企业里才能看到体积庞大、操作复杂的大型计算机以及运行在它们之上的软件系统。从70年代末开始，以微软和IBM为代表的软件、硬件厂商将计算机带进了人们的日常生活。从那以后，软件的发展可谓日新月异，迅速渗透到各个行业。时至今日，软件在我们的日常生活中已经司空见惯，PC、笔记本计算机、平板电脑、智能手机等，无一不是以软件为基础的。小到肉眼难以看清的微型芯片，大到关系国家安全的国防军事工业，软件都按照人们既定的设计，支撑着这个世界的运转。

今天在世界各地，软件都太重要了。一旦软件出现故障，就会造成损失，甚至是灾难。2007年11月11日，美国洛杉矶国际机场海关的计算机发生故障，导致60多个航班受到影响，2万多名旅客滞留在飞机上和候机楼，无法入关，14个小时后计算机故障才得到解决。EDS是美国著名的电子数据系统公司，英国国内收入局曾选择该公司为其设计内部计算机系统。2004年，由于该公司设计的计算机系统不完善，英国税收抵免陷入混乱，造成10亿英镑的损失。在中国，也不乏软件缺陷造成重大损失的例子。2011年，号称“全球最大的中文IT社区”的网站csdn.net被黑客入侵，众多该网站用户的登录信息被黑客打包上传到互联网上。由于现在很多互联网用户在多个网站使用同样的用户名和密码，他们不得不去更改其他网站的密码。而csdn.net将用户密码用明文的形式存储于数据库，也被业内人士嘲笑为“极其不专业和不负责任”。

软件如此重要，软件缺陷造成的影响如此重大，因此保证软件质量就显得尤为关键。软件测试，作为保证软件质量最常用的手段[1]，被很多公司和机构采用。微软公司的测试人员和开发人员的比例一般为1∶1[2]，而在Windows 2000开发团队中，有1 800名测试人员，900名开发人员，测试人员和开发人员的比例达到了2∶1。在谷歌公司，测试人员和开发人员的比例一般为1∶10。微软和谷歌两家公司测试人员和开发人员的比例差

距如此之大，原因是：在微软，工作量巨大的单元测试工作是由测试人员完成的；而在谷歌，单元测试则是开发人员的职责。在国内的龙头企业阿里巴巴，测试人员和开发人员的比例大致为1∶4，而在其子公司蚂蚁金服，由于涉及对安全性要求更高的互联网金融行业，其测试人员和开发人员的比例达到了1∶3，且还有逐年递增的趋势。

就像制造行业每一道工序后都有质检过程一样，软件开发的每一个阶段都有对应的测试环节。以传统软件开发常用的模型为例，软件的开发过程被分为需求分析、概要设计、详细设计、编码阶段。与之对应，软件测试被分为单元测试、集成测试、系统测试、验收测试[3]。而在一些要求更高的系统中，还会涉及性能测试、压力测试、容量测试、兼容性测试、安全性测试等。在这一系列的测试中，单元测试是最低级别的测试活动，它的目的是检测程序模块有无故障存在。也就是说，在基础的测试阶段，不是把程序作为一个整体来测试，而是集中精力测试程序中较小的结构块，以便发现并纠正模块内部的故障[4]。通过单元测试的程序，能够确保每一个模块正确运行，后续的测试如集成测试、系统测试等都是在每个模块能正确运行的基础之上进行的。因此，单元测试是耗时最多、发现问题最早也最多、修复问题成本最低的测试。

软件测试是伴随着软件的产生而产生的，在早期的软件开发过程中，由于软件规模小、复杂度低，也没有形成通用的开发模型，人们通常把软件测试等同于调试，这部分工作常常由开发人员自己完成。

到了20世纪80年代初期，软件进入了快速发展阶段，软件向大型化、复杂化方向发展[5]。这时，一些软件测试的基础理论和实用技术开始形成，软件测试进入了正规化、结构化的阶段。这个阶段的测试是以人工测试为主，人工编写测试用例，人工执行测试用例，人工统计测试结果。人工测试有一个优势——人的思维。对于一些逻辑性很强的软件，人可以在测试的过程中从不同的角度来思考问题，使得设计的测试用例能更好地覆盖软件的功能。

然而，人工测试的代价也是高昂的。测试会遇到许多重复性的工作，重复完成同一项工作，被很多人视为没有价值或者没有挑战的工作。现代软件的规模决定了测试工作量巨大，测试人员在长时间的工作后，会出现疲劳、注意力下降等情况，这不仅导致工作效率下降，也容易导致错误产生[6]。

而自动化测试可以弥补上述缺陷，程序按照人们的设定有条不紊地运行，完成测试工作，它不会觉得重复的工作毫无挑战，也不会因为连续运行数小时而降低精度、产生错误。因此，自动化测试是软件测试发展到一定阶段的必然产物，是软件测试发展的方向。和人工测试相比，自动化测试的优势显而易见，例如，它能够提高测试效率和降低测试成本，将重复性测试独立出来。自动实现快速的回归测试可以避免人工测试容易犯的错测、漏测、多测等错误，模拟人工测试几乎无法办到的多用户并发场景[7]。

1.2 回归测试的必要性

软件测试的关键是测试用例的设计，好的用例设计策略能用较少的用例覆盖更多的

功能点，发现更多的缺陷。因此，自动化测试的关键是要自动地生成好的测试用例，至于测试的执行，只是带着测试用例运行程序而已。

测试用例的设计从来都不是一件简单的事，它需要达到一定的目标。例如，在阿里巴巴公司 Java 工程的测试中，衡量测试用例好坏的标准就有类覆盖率、方法覆盖率、行覆盖率、分支覆盖率等。在不同类型的工程中，各种覆盖率的要求也不一样。这就要求在设计测试用例时要有一定的策略。类覆盖和方法覆盖比较简单，只需要调用该类里面的方法，就算覆盖到了该类及其中的方法。但是，语句覆盖、分支覆盖等要达到一个比较高的覆盖率，就需要在设计测试用例时，能以程序内部结构，特别是分支语句为依据。当然，怎样根据程序的内部结构来设计测试用例，正是测试用例自动生成领域主要的研究内容。近年来，在这个领域已经有一些较为成熟的算法，它们都有各自适用的场景。比较常见的算法有以下几种。

1. 基于随机算法的测试用例自动生成

该算法由 Hamlet 在 1994 年提出[8]，它的基本思想是使用程序输入域上随机产生的数据生成测试用例，将生成的测试用例代入待测程序，检测测试目标是否被覆盖。该算法没有任何的反馈机制，它的优点就是简单，生成测试用例的效率高，由于数据是随机生成的，能较好地避免测试人员的主观因素。当然，由于完全随机，它生成的测试用例覆盖率低，还有较多的无效用例。

2. 基于遗传算法的测试用例自动生成

该算法是一种启发性或者说有反馈机制的算法，由 Jones 在 1995 年首次提出[9]。它实际上是在随机生成用例的基础上，去除不满足约束的测试用例，保留满足约束的用例（次优解），再以次优解为基础，重复这个过程，直到找到最优解。最后得到的测试用例是适合测试环境的最优测试用例。这种算法和随机算法相比，复杂度有所增加，但是生成的测试用例质量（包括覆盖率和执行时间两个方面）明显提高。该算法在开始阶段由于数据量大，能产生大量次优解，然而随着算法的进行，符合要求的次优解越来越少，算法会遇到瓶颈，所以该算法“重大体而不重精确解”。

3. 基于蚁群算法的测试用例自动生成

该算法是 Mcminn 等人将 Dorigo 等人提出的仿生寻优启发式算法引入测试用例生成领域的[10]。该算法先把需要求解的约束抽象为一张有向图 G，把寻找最优解的过程看作一只“蚂蚁”在 G 中移动，移动的过程中不断获取和释放信息（获取的信息如一个变量的取值范围等，释放的信息如是否已经覆盖到目标元素等），最终找到满足约束的解。该算法和遗传算法相比，是一个增强的学习系统，能够更好地利用程序本来的信息，但是由于需要“不断学习”，所以在初期，信息相对匮乏，算法效率偏低，而一旦形成了一定量的信息积累，算法效率将会显著提高。

当然，除了上面所列的 3 种算法，还有其他的算法也在近年被提出，用于测试用例自

动生成。这些算法各有优劣，共同服务于测试用例自动生成领域。

1.3 单元测试概述

单元测试通常是由开发者编写一小段代码，用这段代码检测被测代码的一个很小、很明确的功能是否正常[11]。通常情况下一个单元测试是用于判断某个特定条件下特定函数的行为的，通过执行单元测试，证明某段代码的行为是否和开发者期望的一致。

单元测试是软件测试的早期阶段，它的主要测试对象是函数，主要方法是对被测代码中的控制结构和处理过程进行分析，检查程序的功能是否正常，最主要的检查对象包括语句结构、分支和循环等[12]。由于需要对被测程序的代码进行分析，因此单元测试的代价往往比较高昂。在实际有需求进行单元测试的项目中，人工测试仍然是最主要的单元测试方法，如执行手动设计的单元测试用例、找同事进行代码复审等都是常用的方法。然而，这部分工作相当烦琐，特别是在大型项目中，由于代码量巨大，单元测试的工作量相当大。像人工测试或者代码复审这些传统的方法，对设计测试用例或者审查代码的人要求都比较高，不仅需要他们具有很高的代码水平和测试经验，而且要求他们有耐心。所以，随着软件规模的不断发展，自动化是进行单元测试的迫切需求[13]。

自动化测试虽然能克服人工测试低效和易错的缺点，但是难以很好地实现，原因是被测代码复杂多样，几乎不存在一个通用的解决方法。

1.4 静态测试与动态测试

对程序进行单元测试时，根据是否需要执行被测程序分为动态测试和静态测试。动态测试需要实际执行程序用以收集信息进而生成测试用例，然而动态测试随机选取初值以及执行程序时的不确定性使得程序产生大量迭代，甚至产生溢出。静态测试不实际运行程序，而是通过静态分析和模拟的手段来生成测试用例，开销更小，是测试领域的一个重要分支。

动态测试是由 Webb Miller 和 David L. Spooner 在 1976 年针对浮点型的测试用例生成问题提出的方法。该方法需要反复执行被测程序，并收集程序信息，判断当前特定的测试需求，依据检查运行结果来指导之后的程序分析和用例生成工作。动态测试相比于静态测试具有一定优势，这是因为程序运行时各个输入变量的值就已经确定。然而由于赋值的随机性和试探性的搜索导致的不确定性，动态测试不能保证一定能为存在解的约束集合找到解。

静态测试不实际运行程序，而是通过静态分析和模拟的手段来生成测试用例。这种方法对被测程序语法、结构、接口等结构进行解析后，用符号变量代替实际变量去执行一条程序路径并提取出这条路径上所有需要满足的约束条件，最后通过求解约束满足问题

即约束求解器来生成满足这些约束的测试用例。一般来讲，静态分析技术采用“保守的”分析，并且得到的结果也是可靠的。常用的静态分析技术有以下几种。

1. 符号分析技术

符号分析技术是指用符号变量而不是实际的程序变量去模拟程序的执行过程，即生成用例的过程中并不去实际执行程序。符号分析技术在20世纪70年代被引入程序分析领域[14]，其认为赋值语句是符号分析的参数条件，条件语句则是符号值的约束系统。符号执行包括前向替换和后向替换。前向替换更符合程序真实的自顶向下的执行过程，通过判断路径中的谓词从而得到路径上的约束。后向替换则是从程序出口依次向前替换。

符号分析理论主要面向变量间的现行关联问题求解，而不讨论变量间的逻辑关联。符号分析能够精确地分析程序的行为，并在遍历数据流的过程中对所有变量在任何可达路径下的可能取值进行计算，提供保守的分析结果。但是当程序中包含结构体、数组、字符串、库函数等较为复杂的数据结构或约束形式时，为其生成相应的符号是一项比较困难的工作。本书第5章介绍的库函数约束求解模型即可较好地处理包含库函数的约束，为其输入参数生成用例。

2. 区间抽象和区间运算

抽象解释是一种程序行为的逼近理论，用于不同抽象级来逼近程序的形式语义[15]。它的主要思想就是把程序的语义计算替换为抽象域的计算，抽象执行的结果能够反映程序的实际运行情况。在程序分析过程中，程序的状态集合是由抽象域中的域元素来近似的，而程序的动作语义(如判断、赋值、循环等)则由抽象域中的域操作进行建模。区间抽象域(interval domain)是对程序中变量的取值范围进行的区间抽象，它忽略了变量之间的关系，认为变量之间是相互独立的，即区间抽象是一种非关联的抽象。区间运算就是用区间集合代替具体数值的数学方法，它可以更加保守地描述变量的可能性取值。区间运算由Burkill和Young提出，Moore将区间运算运用到数值分析领域求解舍入误差。在后来的发展中，区间运算被越来越多地运用到科学计算、工程计算领域。为了提高软件测试精度并保证测试结果是可靠的，区间运算被应用到软件测试领域。

1.5 黑盒测试和白盒测试

软件测试从不同的角度有不同的分类，不同的分类有不同的测试方法。一般来讲，根据是否分析程序内部结构或逻辑，软件测试可以分为黑盒测试和白盒测试。

1. 黑盒测试

黑盒测试也被称为功能测试或基于规格说明的测试，它是一种从软件外部对软件实

施的测试，目的是尽可能发现软件的外部行为错误[4]。对于采用黑盒测试相关方法的测试人员来讲，被测程序被看作一个打不开的黑盒，内部的逻辑完全不可见，它只是从输入定义域到输出值域的映射。

在使用黑盒测试方法时，测试人员根据软件规格说明对软件功能、性能、安全性、兼容性等指标进行测试，目的是检测出软件已有的或者潜在的错误，从而保证软件的质量。如果希望采用黑盒测试检测出软件程序中的所有故障，就要对程序进行穷举测试。然而在实际工程中，穷举测试常常是不切实际的。常用的黑盒测试方法有以下几种。

(1) 划分等价类

等价类划分法将程序所有可能的输入数据(有效的和无效的)划分成若干个等价类，从每个部分中选取具有代表性的数据当作测试用例进行合理的分类，测试用例由有效等价类和无效等价类的代表组成，从而保证测试用例具有完整性和代表性。等价类是输入域中互不相交的子集，所有等价类的并集便是整个输入域[4]。划分等价类的测试方法具有测试完备性，可以避免生成冗余的测试用例。

(2) 边界值分析法

大量的软件测试实践表明，故障往往出现在定义域或值域的边界上，而不是在其内部[3]。程序的边界是一种特殊的情况，应用边界值分析法设计测试用例时需要先确定边界情况，再找出合适的边界，针对问题的输入域、边界值条件等细致地进行考虑。

(3) 因果图法

因果图法运用简单的直线、节点和逻辑符号表示原因、结果及其联系。因果图法帮助测试人员按照一定步骤，检测被测程序输入条件的各种组合情况，从而高效而全面地开发测试用例。由于它可以将自然语言转化为严格的形式化规格语言，因此可以指出规格说明中存在的不完整性和二义性[4]。

(4) 决策表法

决策表将作为条件的所有输入及其各种组合值以及对应输出值均罗列出来，形式简明并且可以避免遗漏。由于决策表可以将复杂的问题按照各种情况列举出来，因此可以设计出完整的测试用例集合[3]，但其工作量在几种方法中是最大的。

2. 白盒测试

白盒测试一般用于分析程序的内部结构，因此又称白盒测试为结构测试或基于程序的测试。白盒测试不再将被测程序当作一个不透明的盒子，而要分析其内部逻辑，并达到一定程度的覆盖以判断软件测试的充分性。

白盒测试分为面向程序结构的控制流测试和面向变量的数据流测试。一般情况下，用控制流图中的路径描述程序运行过程。常用的控制流覆盖准则及覆盖测试的标准有语句覆盖、分支覆盖、谓词覆盖以及路径覆盖准则，通过对测试用例在以上测试准则的覆盖率统计来检测测试的完备性。数据流测试则用于测试用例定义和使用的正确性，其常用的覆盖准则有定义覆盖测试准则、引用覆盖测试准则、定义-引用覆盖准则等[4]。

除了以上的覆盖准则外，程序插装也是白盒测试的一个基本测试手段，它通过在被

测程序中插入操作来实现测试目的，如统计覆盖信息、断言证实运行特性等。

1.6 基于路径和覆盖率的测试

软件测试中的控制流和数据流测试等问题都可以归结为面向路径的测试用例生成问题。该问题可以描述为：给定一个程序 P 和 P 中一条路径 w，设 P 的输入空间为 D，求 x，使得 P 以 x 为输入运行，所经过的路径为 w。求解面向路径的测试用例生成问题归根到底是一个约束满足问题(Constraint Satisfaction Problem，CSP)，即怎样使得生成的用例满足当前路径 w 上的所有约束。

约束满足问题的求解往往结合使用推理和搜索方法[16]。推理技术本质上是一个问题等价转换技术，即将问题转化为一个更易于求解且与原问题等价的问题；搜索技术则是指在所有变量的当前值域中寻找问题的解。一般来讲，传统搜索算法的搜索空间非常庞大，因而求解算法效率较低。一致性技术(包括弧一致性、路径一致性等)的出现使得求解约束满足问题的效率得到很大提升。在使用搜索算法求解一个约束满足问题时，通常需要使用启发式算法帮助做一些决定，例如，选取哪一个变量进行赋值或者该给某个变量赋何值等[16]。

测试是否完整需要一个判定标准，覆盖率就是判定测试完备性的一个重要指标。判定一个面向路径生成的测试用例是否满足路径上的约束，就要检测该用例是否覆盖了其目标元素；判定一组面向路径生成的测试用例是不是此组约束集合的解，就要检测该组用例是否覆盖这条路径上的所有目标元素。假设测试完成后，当前用例已经覆盖的目标元素个数为 t，路径上所有目标元素的个数为 T，那么当前用例的覆盖率为

$$p=\frac{t}{T}\times 100\% \tag{1-1}$$

由此可见，覆盖率与目标覆盖元素个数相关，而目标覆盖元素的确定则与覆盖准则相关。一组优秀的测试用例就是用最少的用例去覆盖最多的目标元素。100%覆盖率是软件测试追求的目标，然而这个目标在实际工程中却很难达到，主要有以下几个原因。

1. 程序本身存在不可达路径

不可达路径就是指无论为变量在其输入域中取何值均不能满足此条路径中某个或某些约束。为不可达路径生成用例或者生成失败，即选值失败，或者随机生成不满足约束要求的任意用例，使得程序中的某些代码或目标元素无法被覆盖。

2. 程序存在死码

程序死码表示在系统实现阶段，由于设计问题，程序存在一些永远不会被执行的代码，因此在生成用例阶段有一些目标元素可能永远不会被覆盖，最终影响覆盖率。

3. 测试用例有遗漏

在用例设计阶段，由于测试人员疏漏或者约束求解系统存在的缺陷，生成的测试用例不够完备，不能覆盖所有目标元素，覆盖率不能达到100%。

虽然达到100%的覆盖率是所有软件测试活动追求的目标，但是我们还必须考虑到现实情况中各种条件的限制，用最小的代价完成最完备的测试。因此在软件测试阶段，各个公司、测试项目等需要根据自己的需求和实际情况制定最合适的覆盖准则和测试标准。

1.7 约束求解问题

为指定的程序路径自动生成测试用例是软件单元测试的一个基本问题。解决该问题的一个难点是为路径中的约束建立一个系统并且求解[17]，这就是人工智能中的一个传统问题——约束求解问题。在数学上，约束求解问题被定义为一个对象的集合的状态必须满足一些约束或者限制。而在面向路径的单元测试中，约束求解的含义是把该路径中所有的约束提取出来，组成一个约束系统，通过一定的算法，找到一组特定的值，使得这一组值能够满足约束系统中的每一个约束。

随机法是最直观、最简单的算法，不管是什么约束，它都随机地为约束中的变量选值。也不管选出的值是否符合约束，甚至都不管选出来的值对约束来说是不是一个合法的取值，例如，对于条件语句“if(log(x)>0)”，随机法可能给 x 的值是一个负数。

而其他的算法和随机算法相比，都加入了一些反馈的机制和利用被测代码本来信息的机制。例如，对于条件语句“if(x>5)”，这些算法在经历了初始几次的取值都小于5的失败后，可能就会感知到也许 x 的取值应该大于5。

总之，这些算法的作用就是找到一个合适的机制，来求解路径中的约束，达到生成的测试用例能满足约束的目的。

1.8 代码测试系统

代码测试系统(Code Testing System，CTS)是一款面向C语言的自动化单元测试工具。CTS支持多种覆盖准则，通过静态分析技术对被测试单元自动生成测试用例，同时动态执行测试用例并统计代码覆盖率；支持代码度量、故障定位以及回归测试。CTS的自动生成测试用例、回归测试以及故障定位等功能可以大大提高测试效率，提高测试的完整性。CTS的总体框架如图1-1所示。

1. 程序预处理

这一阶段主要是扫描被测程序，对被测文件进行模块划分，划分出测试单元，生成驱