北 京 科 技 大 学

硕士学位研究生
选题报告及文献综述

蜕变测试与适应性分区测试的集成方法与工具研究

指导教师：孙昌爱

单　　位：计算机与通信工程学院

学　　号：G20168664

作　　者：代贺鹏

专业名称：软件工程

入学时间：2016 年 9 月 10 日

2017 年　9 月　1 日

目　录

1 选题的背景和意义

1.1 选题的背景

软件测试是保证软件质量的重要方法，贯穿于软件开发的整个周期。测试人员根据待测软件的规格说明以及程序的内部结构精心设计一批测试用例，并利用这些测试用例执行程序，然后根据测试预期判断输出结果是否正确。

检测软件故障的测试技术有很多，例如：随机测试(random testing)[1]、覆盖测试(coverage testing)[2]、组合测试(combinatorial)[3]、分区测试(partition testing)[4]。然而在绝大多数情况下，运用这些技术需要一个能判断输出是否正确的机制——测试预期(test oracle)。在实际运用中，测试预期可能不存在或者需要很大的代价去运用测试预期。由此，当缺少预期时验证软件的输出是否正确是一件很困难的事情[5]。蜕变测试技术(MT)是能够缓解测试预期问题的技术之一[6]。MT 技术运用待测程序的一些属性定义一些蜕变关系(MRs)。原始的测试用例和识别的蜕变关系可以用来生成衍生测试用例。然后原始测试用例和衍生测试用例分别在待测程序中执行，最后判断原始测试用例的输出以及对应的衍生测试用例的输出是否违反了蜕变关系。

很明显，识别的蜕变关系的质量影响 MT 技术的故障检测效率。已经有很多的研究调查怎么样选择或者生成好的蜕变关系[7-12]。除了蜕变关系原始测试用例的生成挑选以及执行过程也影响 MT 技术的故障检测效率。然而在以往的研究中，随机测试是选择原始测试用例的主要的方法。随机测试技术由于本身的随机特性对于一些非平凡的故障可能需要较大的代价才能选中某一测试用例让然后揭示该故障。分区测试技术[13]最初是为了“系统”的产生测试用例，以便提高随机测试技术的测试效率。分区测试首先将输入域分为不相交的若干个分区，在理想状态

下每一个分区具有同构的性质，即一个输入引起了软件故障，那么与该输入同分区的其它输入也将能够引起软件故障。由此只需要在每一个分区之中选择一个或几个测试用例进行测试。从而提高随机测试解释故障的效率。但是在一些情况下原始的分区测试技术并不能提高随机测试技术的测试效率。其原因是，在实际情况下分区的同构的性质并不能得到保证，此时分区测试技术的效率可能不高。适应性分区测试在分区测试的基础上利用测试过程的信息改进传统分区测试的测试效率。因此本研究利用 APT 作为控制测试过程的策略，MT 作为一种判断结果是否正确的机制，提出一种新的缺少测试预期时的测试策略——基于适应性分区的蜕变测试(AP-MT)。

1.2 蜕变测试

1.2.1 蜕变关系的概念

传统的软件测试技术需要判断测试用例的实际输出与预期输出是否一致，这种判断机制叫做测试预期[14]。但是在以下两种情况下会遇到测试预期问题：

1. 不存在测试预期
2. 存在测试预期但是运用这个测试预期需要很大的代价

为了缓解测试预期问题 Chen 提出了蜕变测试[6]，并且该技术已经成功地运用到各种领域[15-19]。蜕变测试技术的关键是根据待测软件的属性识别蜕变关系。原始测试用例经过蜕变关系的作用可以生成衍生测试用例，生成的衍生测试用例与对应的原始测试用例形成一个测试用例组。蜕变关系以及蜕变测试的正式的定义如下[20]：

定义 1：蜕变关系

设程序P是函数f的实现，$x_1, x_2, \ldots, x_n (n \geq 1)$是函数的不同变量，如果输入之间存在关系R使得$\mathrm{f}(x_1), \mathrm{f}(x_2), \ldots, \mathrm{f}(x_n)$满足关系$R_f$，即：

$$\mathrm{R}(x_1, x_2, \ldots, x_n) => R_f\big(\mathrm{f}(x_1), \mathrm{f}(x_2), \ldots, \mathrm{f}(x_n)\big) \quad (1)$$

则$\mathrm{MR}(\mathrm{R}, R_f)$为程序P的蜕变关系，显然如果程序P正确，则有：

$$\mathrm{R}(I_1, I_2, \ldots, I_n) => R_f\big(\mathrm{P}(I_1), \mathrm{P}(I_2), \ldots, \mathrm{P}(I_n)\big) \quad (2)$$

成立，其中$I_1, I_2, \ldots, I_n$是对应$x_1, x_2, \ldots, x_n (\mathrm{n} \geq 2)$的测试用例。

1.2.2 蜕变测试的过程

假设程序P是函数f的实现，MT 技术的测试过程如下[21]：

1. 识别函数f的一个蜕变关系MR
2. 用适当的测试用例选择策略生成原始测试用例I
3. 基于蜕变关系MR由原始测试用例生成衍生测试用例I'
4. 原始测试用例I与衍生测试用例I'分别在程序P上执行，得到相应的输出O, O'
5. 验证I，I'，O以及O'是否违反了MR：如果违反了MR，那么程序P就认为存在缺陷

对于待测软件的每一个蜕变关系MR_i重复以上的步骤。

1.3 适应性分区测试(APT)

由于随机测试与分区测试是基于不同的直觉，因此在揭示软件中不同种类的故障时，两种测试策略可能时互补的。基于这一点 Cai 在[22-23]中提出了随机分区测试。在随机分区测试中，软件的输入域被分为m个分区$c_1, c_2, \ldots, c_m$，每一个分区被选择的概率为p_i，并且所有分区都选择的概率满足$\sum_{i=1}^{m} p_i = 1$。随机分区测试的测试过程分为两个步骤：首先，根据测试剖面$\{p_1, p_2, \ldots, p_m\}$选择一个分区$c_i$;

然后在分区c_i中随机选取一个测试用例进行测试。当测试剖面与各个分区的失效率相差很大时随机分区测试的测试效率可能不高。Cai 在[24]中提出了动态随机测试策略该策略根据测试的历史信息动态改变测试剖面。但是动态随机测试在调整分区对应的概率时整个测试过程参数的大小保持不变。这一点在一些情况下可能不是好的策略。我们课题组基于“引起故障的输入趋向于集簇在连续的区域”这一观察[25-26]，提出了在测试过程中适应性的调整分区对应的被选择的概率。并且通过 5 个现实中的真实存在的程序验证适应性分区测试的测试效率比传统随机分区测试的测试效率高。

1.4 研究的意义

1.4.1 缓解了测试预期问题

测试预期在实际的测试过程是不可避免的，并且很难测试。本文研究的 AP-MT 测试策略可以在缺少测试预期时利用原始测试用例以及衍生测试用例及对应的输出是否满足蜕变关系，若不满足，则说明程序中存在缺陷。因此 AP-MT 策略缓解了测试预期问题。

1.4.2 解决了测试预期不存在时 APT 的应用问题

传统的适应性分区测试需要一种判断输出结果是否正确的机制——测试预期。当测试预期不存在时，根据 APT 策略挑选测试用例并执行之后，不能确定输出的结果是否正确。因此，在这种情况下 APT 测试策略的应用性受到了挑战。本研究提出的 AP-MT 测试策略，引入了蜕变测试的蜕变关系概念，可以根据原始测试用例、对应的衍生测试用例及一致的输出是否违反相应的蜕变关系来判断

待测软件中是否存在缺陷。由此解决了传统的 APT 测试策略在缺少测试预期时不能运用的问题。

1.4.3 从“本质”上提高了蜕变测试的测试效率

大多数蜕变测试的控制测试过程的策略是随机测试，但是随机测试没有利用待测程序的信息也没有利用测试的过程信息，因此采用随机测试作为控制测试过程的策略时，在揭示一定数目的故障方面，往往需要更多的测试用例。APT 策略在分区测试的基础上适应性的控制分区被选择的过程，使得失效率高的分区更有可能被选到，由此在揭示一定数目的故障方面，很有可能比传统的随机测试效率更高。所以将 APT 代替 RT 作为蜕变测试的过程控制策略，将提高蜕变测试的测试效率——用更少的测试用例揭示一定数目的故障。

2 研究现状

在该章节主要介绍适应性分区测试的研究进展以及提高蜕变测试的测试效率的研究进展。

2.1 蜕变测试研究方向介绍

Sergio 在[27]中收集了 1998 年到 2016 年期间的 119 篇关于蜕变测试的英语语言攥写的文献，并且对这 119 篇文献的研究主题进行归纳总结，结果如图 2.1 所示。从图中可以看出 49%的文章是将蜕变测试与其它领域结合，剩下的文章大部分研究构造蜕变关系(19%)，一部分研究内容是与其它测试技术结合(10%)，还有一些是研究执行原始测试用例(5%)与生成原始测试用例(4%)。由此可以看出关注蜕变测试本身，旨在提高蜕变测试本身测试效率的研究大多集中在构造蜕变关系，

此外测试过程也影响蜕变测试的测试效率，但是在这一方面研究的很少。

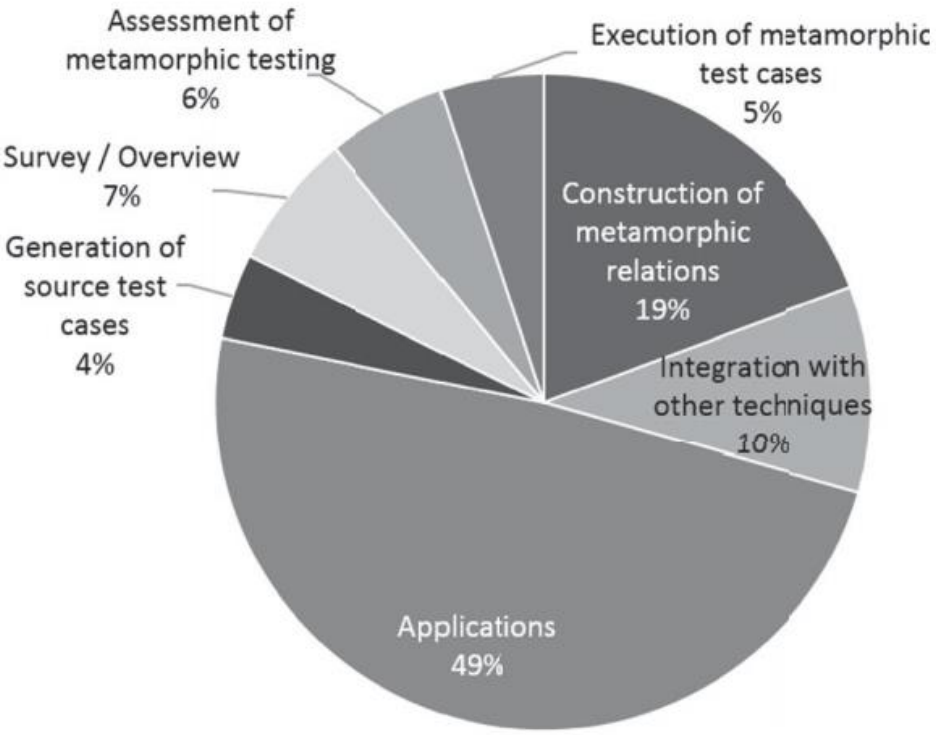

图 2.1 研究主题

2.2 METRIC 识别蜕变关系

构造蜕变关系需要从待测软件本身出发人工识别。很多的研究提出了不同的识别蜕变关系的方法，要么跟现有的蜕变关系结合要么自动化产生蜕变关系。Liu[28]中提出了 CMR 方法，该方法通过结合已有的蜕变关系产生新的蜕变关系。Zhang[29]提出了一种基于搜索的方法，该方法可以生成多项式形式的蜕变关系。Su[30]在前期程序不变性推理研究的基础上提出了 KABU 方法动态地推理可能地蜕变关系 Chen 在[31]中提出一个基于待测软件规格说明书的方法学——METRIC。METRIC 在 category-choice 框架上识别蜕变关系。在这种框架下，测试人员将待测软件的输入域分成 category 和 choice，将不同的 choices 组合形成完整的测试帧。测试帧是抽象的测试用例，定义了可能的输入组合。每次从测试帧集合中选

取两个测试帧作为识别蜕变关系的候选对，然后由测试人员决定该候选对是否可以用来识别蜕变关系。METRIC 使得测试帧、原始测试用例、蜕变关系与衍生测试用例关联起来。接下来将通过一个停车计费系统举例说明 METRIC 方法识别蜕变关系的过程。

Example1(停车计费系统)：停车计费系统可以接受不同类型的交通工具和不同类型的车。该系统支持工作日以及周末停车，并且车主可以提供折扣券或者预估停车时间。如果实际停车时间在车主预估范围内则可以享受 40%的折扣。具体的细节如表 2.1。顾客可以选择提供折扣券或者预估停车时间。显然贵客可以选择既不提供折扣券也不预估时间，但是贵客不能提供折扣券并且预估时间。另外顾客只能在同一天里停车，也就是假如一辆车在 23.30 停车，该顾客必须在 24.00 之前离开。categories 和 choices 可以从停车计费的规格说明中获得，具体内容如表 2.2。将属于不同 categories 中的 choices 有效的组合可以得到完整的测试帧，从表 2.2 中可以得到 90 个完整的测试帧，例如 $\text{test frame}_1 = \{\text{type of } vehicle_{motorcycle}, \text{day of } week_{weekday}, \text{discount } coupon_{no}, \text{actual } hours_{(0.0,2.0)}\}$。具体的测试用例可以将形如$\text{test frame}_1$的完整测试帧具体化得到。例如将 test frame_1 具体化可以得到测试用例 $\text{test case}_1 = \{\text{type of } vehicle_{motorcycle}, \text{day of Monday}, \text{discount coupon} = \text{no}, \text{actual hours} = 1.4\}$。

表 2.1 停车计费系统收费细节

Actual hours of parking	Weekday			Saturday and Sunday		
	Motorcycle	Car:2-door	Car:others	Motorcycle	Car:2-door	Car:others
(0.0,2.0)	4.00	4.5	5.0	5.0	6.0	7.0
(2.0,4.0)	5.00	5.5	6.0	6.5	7.5	8.5
(4.0,24.0)	6.00	6.5	7.0	8.0	9.0	10.0

表 2.2 停车计费系统范畴和选项列表

categories	choices
Type of vehicle	type of $vehicle_{motorcycle}$, type of $vehicle_{motorcycle}$
Type of car	Type of car_{2-door}, Type of car_{others}
Type of week	Day of $week_{weekday}$, Day of $week_{saturday\ or\ sunday}$
discount coupon	Discount $coupon_{yes}$, Discount $coupon_{no}$
Estimated hours of parking	Estimated $hours_{(0.0,2.0)}$, Estimated $hours_{(2.0,4.0)}$, Estimated $hours_{(4.0,24.0)}$
Actual hours of parking	Actual $hours_{(0.0,2.0)}$, Actual $hours_{(2.0,4.0)}$, Actual $hours_{(4.0,24.0)}$

METRIC 方法识别的蜕变关系涉及到两个有区别的输入，因此测试人员识别一条蜕变关系只需要考虑两个有区别完整的测试帧，并且将这样的两个测试帧称为候选对。METRIC 方法是识别蜕变关系的步骤如下：

（1） 挑选两个有意义并且有区别的完整测试帧作为候选对

（2） 由测试人员决定挑选的候选对是否可以识别蜕变关系，如果可以则将相应蜕变关系记录下来。

（3） 重复(1)直到穷尽所有的候选对或者识别的蜕变关系数目达到了预期。

有区别的完整测试帧($test\ frame_1, test\ frame_2$)一定包含了一些不同的 choices：要么（情形 1）$test\ frame_1, test\ frame_2$有相同的 categories 但是有不同的 choices，要么（情形 2）$test\ frame_1$包含一个或多个$test\ frame_2$中不包含的 categories。不同的 choices（对应情形 1）和额外的 choices（对应情形 2）都可以看作两个测试帧有不同的 choices，不同的 choices 对应的 categories 叫做不同的 categories。由此可以将候选对分为 3 种情况：1，choice-only 候选对（测试帧不同的原因是由于情形 1）；2，category-only 候选对(测试帧不同的原因是由于情形 2)；3，category-choice 候选对(既不是 choice-only 候选对也不是 category-only 候选对)。接下来以停车计费系统为例展示 METRIC 在测试帧的基础上识别蜕变关系的过程。假如有以下完整的测试帧：

$$B_1 = \{type\ \ of\ \ vehicle_{motorcycle}, day\ \ of\ \ week_{weekday}, discount\ coupon_{no}, actual\ \ hours_{(0.0,2.0)}\}$$

$$B_2 = \{type\ \ of\ \ vehicle_{motorcycle}, day\ \ of\ \ week_{weekday}, discount\ coupon_{no}, actual\ \ hours_{(2,0,4.0)}\}$$

$$B_3 = \{type\ \ of\ \ vehicle_{motorcycle}, day\ \ of\ \ week_{weekday}, discount\ coupon_{no}, estimated\ \ hours_{(2,0,4.0)}, actual\ \ hours_{(2,0,4.0)}\}$$

$$B_4 = \{type\ \ of\ \ vehicle_{motorcycle}, day\ \ of\ \ week_{weekday}, discount\ coupon_{no}, actual\ \ hours_{(4,0,24.0)}\}$$

首先考虑B_1和B_2，这两个测试帧的 categories 相同但是 Actual hours of parking 这个 category 中的 choice 不同，因此B_1和B_2属于 choice-only 候选对。在B_1和B_2只有实际的停车时间不同，直观上停车时间越长所需要的费用越多，因此可以识别的蜕变关系如下：

(MR1)假设一辆摩托车在工作日停车，并且车主既没有提供折扣券，也没有预估预期停车时间，如果实际的停车时间从(0.0,2.0)到(2.0,4.0)那么停车费用也增加。

然后考虑B_3和B_4，B_3的费用一定要比B_4的低，因为一方面B_3有正确的预估时间，可以享受 40 的折扣；并一方面B_3的停车时间比B_4的低。因此可以识别的蜕变关系如下：

(MR2)假设一辆摩托车在工作日实际停车时间为(2.0,4.0)，并且车主既没有提供折扣券也没有预估停车时间那么所需要的费用将比实际停车时间为(0.0,2.0)并且提供了正确的预估时间的费用要高。

METRIC 方法可以让测试人员系统的、高效的识别蜕变关系，并且将原始测试用例、测试帧、蜕变关系以及衍生测试用例之间关联起来

2.3 适应性分区测试介绍

分区测试旨在系统地产生测试用例，并且希望分区中存在同构地性质，即如果一个分区中地测试用例揭示了软件中地故障，那么该分区中的其它测试用例也能揭示软件中的故障，同理若不能揭示软件中的故障，那么该分区中的其它测试用例也不能揭示软件中的故障。但是实际中分区的同构性质很难保证，因此在一些情况下，原始的分区测试的测试效率不如随机测试。适应性随机测试根据引起故障的输入趋向于集簇在连续的区域这一直觉，在测试的过程中动态的改变测试剖面。适应性分区测试可以在测试的过程中根据每一个分区的测试用例揭示故障的情况动态的改变分区被选择的概率。在测试过程中如果一个分区中测试用例揭示了软件中的故障，那么就认为该分区有较高的失效率因此增大该分区被选择的概率同时减小其它分区被选择的概率 ;相反如果一个分区中的测试用例没有揭示软件中的故障，那么就认为该分区具有较低的失效率因此减小该分区被选择的概率同时增大其它分区被选择的概率。在测试过程中动态适应性的改变测试剖面，使得具有较高失效率的分区更有可能被选中，具有较低失效率的分区被选中的几率较低。

本课题组刚刚在 Transactions on Reliability 上投的文章中提出了两种算法 :1，基于 Markov 链的适应性分区测试 MAPT；2，基于奖惩机制的适应性分区测试 RAPT。通过 3 个现实中真实存在的规模较大程序(LOC>5000)的 10 个版本验证 MAPT 与 RAPT 相较于传统的随机分区测试的效率有明显的提高。接下来以 MAPT 为例，展示 MAPT 测试算法。 假如将待测软件的输入域划分为m个分区 $\{1,2,\ldots,m\}$。MAPT 算法将测试用例所在的分区当成状态，因此不同状态下的状态转移的概率可以以矩阵的形式表现出来:

$$\mathrm{P}=\begin{pmatrix} p_{1,1} & \cdots & p_{1,m} \\ \vdots & \ddots & \vdots \\ p_{m,1} & \cdots & p_{m,m} \end{pmatrix}$$

并且状态转移矩阵具有性质 :$\forall \mathrm{i}, \sum_{j=1}^{m} p_{i,j}=1$, 也就是矩阵的每一行的和都为 1。如果·分区$c_i$被选择并执行了一个测试用例，那么分区$c_j$被选择并执行一个测试用例的的概率为$p_{i,j}$。MAPT 测试方法将根据一下过程适应性的改变$p_{i,j}$的值。

假如分区c_i被选中并且执行了测试用例t。如果该测试用例揭示了软件中的故障那么对$\forall \mathrm{j}=1,2,\ldots,\mathrm{m}$并且$\mathrm{j}\neq \mathrm{i}$，有

$$p'_{i,j}=\begin{cases} p_{i,j}-\frac{\gamma\times p_{i,i}}{m-1} & if\ \ p_{i,j}>\frac{\gamma\times p_{i,i}}{m-1} \\ p_{i,j} & if\ \ p_{i,j}\leq\frac{\gamma\times p_{i,i}}{m-1} \end{cases} \tag{1}$$

$$p'_{i,i}=1-\sum_{j=1,j\neq i}^{m} p_{i,j} \tag{2}$$

否则，如果这个测试用例没有揭示软件中的缺陷那么对$\forall \mathrm{j}=1,2,\ldots,\mathrm{m}$并且$\mathrm{j}\neq \mathrm{i}$，有

$$p'_{i,j}=\begin{cases} p_{i,j}+\frac{\tau\times p_{i,j}}{m-1} & if\ \ p_{i,i}>\frac{\tau\times(1-p_{i,i})}{m-1} \\ p_{i,j} & if\ \ p_{i,i}\leq\frac{\tau\times(1-p_{i,i})}{m-1} \end{cases} \tag{3}$$

$$p'_{i,i}=\begin{cases} p_{i,i}-\frac{\tau\times(1-p_{i,i})}{m-1} & if\ \ p_{i,i}>\frac{\tau\times(1-p_{i,i})}{m-1} \\ p_{i,i} & if\ \ p_{i,i}\leq\frac{\tau\times(1-p_{i,i})}{m-1} \end{cases} \tag{4}$$

从公式可以看出每一个分区调整的幅度跟当前分区被选择的概率大小有关并且，当前分区被选择的概率较大说明在测试的过程中当前分区有更高的概率揭示软件中的故障或者测试人员根据以往的测试经验认为该分区具有较高的可能存在故障。因此增大这样的分区概率时幅度应该大一点，减小的幅度小一点。并且从状态i转移到状态i的概率只跟i状态对应的分区内的测试用例执行情况决定，如果该分区内的测试用例一直没有揭示软件中的故障，说明该分区的失效率很低$p_{i,i}$的值会一直减小，从而增加其它分区被选择的概率，并且$p_{i,i}$的值不会受其它分区测试结果的影响。MAPT 的具体算法如下所示。

算法：MAPT
输入：γ，τ，$p_1, p_2, \ldots, p_m$
1:对状态转移矩阵初始化，$p_{i,j} = p_j$
2:令执行的测试用例数目 noTC = 0
3:while 终止条件不满足时
4: if noTC = 0
5: 根据测试剖面$\{p_1, p_2, \ldots, p_m\}$选择一个分区$c_i$
从分区c_i中选择一个测试用例t
6: else
7: 根据当前测试用例所在的分区c_i将状态转移矩阵的第i行
作为测试剖面$\{p_{i,1}, p_{i,2}, \ldots, p_{i,m}\}$,选出下一步测试的分区$c_j$
8: 在分区c_j中选择一个测试用例t
9: end_if
10: 将测试用例t在待测软件中执行
11: noTC++
12: 基于测试结果按照相应的计算公式更新状态转移矩阵
13:end_while

3 研究内容

3.1 适应性分区测试与蜕变测试结合可行性分析

3.1.1 适应性分区测试与蜕变测试结合中的问题

在 1.2.2 章节可以看出蜕变测试的第一步时要识别待测程序中的蜕变关系。蜕变关系的“好坏”影响蜕变测试揭示故障的效率。除此之外原始测试用例会影响蜕变测试的测试效率。适应性分区测试根据测试过程中的测试信息，如果一个分区中的测试用例揭示了软件中的故障，那么就认为该分区具有较高的失效率，因此增大该分区被选择的概率减小其它分区被选择的概率。最终使得具有较高失效率的分区具有较高的被选择概率，较低失效率的分区具有较低的被选择概率。因此适应性分区测试比传统的随机测试在理论上能够更快的找到具有故障揭示能力的测试用例。由此从理论上适应性分区测试可以帮助蜕变测试更快地找到具有

潜在故障揭示能力地原始测试用例。但是蜕变测试与以往传统的测试技术例如分区测试、动态随机测试等的不同点在于多出了蜕变关系。即便一个测试用例具有揭示故障的潜力如果经过“不好”的蜕变关系作用之后生成的对应衍生测试用例，在待测程序上执行之后，并不一定能够违反该蜕变关系，即不能揭示软件中的故障。因此在给定的待测程序中某一个测试用例与蜕变关系结合形成衍生测试用例的次序也将影响 MT 技术的测试效率。本文为了研究方便引入 Chen 在[27]中提到的 METRIC 蜕变关系识别技术，该技术蜕变关系是从成对的完整测试帧中识别。因此特定的两个测试帧形成的测试用例只与从这两个测试帧中识别的蜕变关系结合形成衍生测试用例。从而将选取测试用例与选取蜕变关系联系起来。

3.1.2 适应性分区测试怎么控制蜕变测试的执行过程

在蜕变测试的测试过程中，假设将待测软件的输入域划分为若干分区，那么在执行过程中涉及到的对象有 :原始测试用例(Source test cases)、分区(Partitions)、衍生测试用例(Follow-up test cases)、蜕变关系(MRs)以及待测程序 P。并且待测软件的蜕变关系是由 METRIC 技术得到的。那么除了待测程序 P 以外在测试过程中涉及到的其它对象之间的关系如下图 3.1 所示。从图中可以看出蜕变测试执行过程中原始测试用例(Source test cases)、分区(Partitions)、衍生测试用例(Follow-up test cases)、蜕变关系(MRs)均跟测试帧(Test frame)有关。并且测试帧可以等同于分区。在实际的测试过程中不同的待测程序测试帧的检测故障的能力不同，因此如果在测试过程中识别检测能力强的测试帧将提高蜕变测试的测试效率。适应性分区测试可以利用测试过程的执行信息识别具有较高故障检测能力的测试帧。所以适应性分区测试与蜕变测试结合在理论上可以提高蜕变测试的测试

效率。

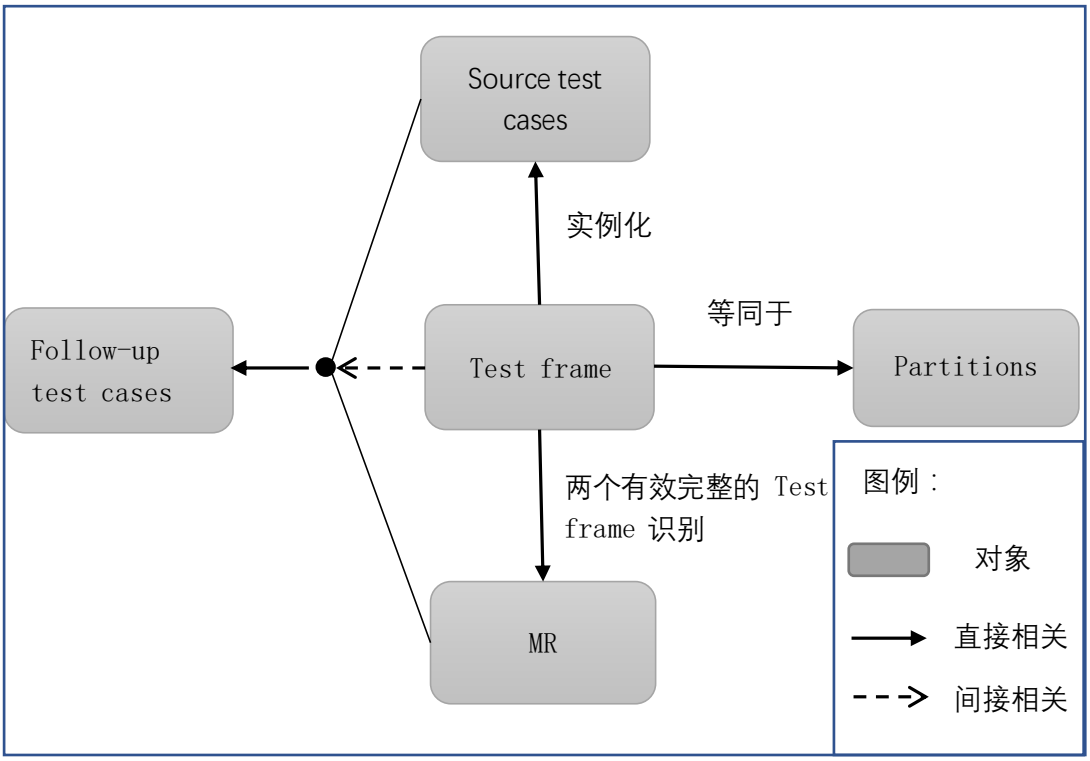

图 3.1 蜕变测试过程涉及到的对象之间的关系图

3.2 适应性分区与蜕变测试集成的具体算法实现

实现的具体控制算法应当考虑以下两个方面：

a) 适应性的控制分区与蜕变关系的选择，使得蜕变测试相对于传统的随机测试作为过程控制策略的测试效率更高，即用更少的测试用例揭示更多的故障。

b) 平衡算法的复杂度。这里主要是指时间复杂度。由于要根据测试的过程适应性的选择分区与蜕变关系，使得额外的计算大量增加，如果时间复杂度非常大，即便是相对于传统的随机测试效率有所提升，但是在实际应用中也是不可取的。所以要平衡这两者的关系。

3.3 基于适应性分区测试技术的蜕变测试支持工具

该工具可以集成 METRIC 技术的支持工具 MR-GEN，在 MR-GEN 的基础上加上测试用例的具体话部分以及适应性分区的控制算法。

第一，该工具首先对包含完整测试帧的文件进行解析，为接下来的蜕变关系识别以及分区做铺垫。

第二，允许用户定制候选对的的一些信息，例如 :识别蜕变关系的最大数目。

第三，展示候选对以便用户判断是否可以识别蜕变关系。

第四，记录用户描述的蜕变关系。

第五，计算每一个测试帧距离其它测试帧的距离矩阵。

第六，插入适应性分区控制算法

第七，允许用户配置执行的信息，例如：停止条件。

第八，执行测试用例并记录测试结果

第九，将识别的蜕变关系以及最后的测试结果生成测试报告。

4 研究工作进度安排

4.1 工作计划

时间段	任务
2017.9-2017.10	收集各种资料并阅读文献
2017.11-2017.12	设计 APT 应用在 MT 中的具体算法
2018.1-2018.4	设计实验进行验证
2018.5-2018.6	开发工具辅助实验
2018.7-2018.8	调试工具
2018.9-2018.10	撰写论文，准备答辩

4.2 最终成果形式

（1） 将 APT 测试技术与 MT 测试技术集成并且开发相应的支持工具；

（2） 在有影响力的期刊或者会议上发表一篇论文。

5 参考文献

[1] Hamlet R. Random Testing[M]// Encyclopedia of Software Engineering. John Wiley & Sons, Inc. 2002:970--978.

[2] Lyu M R, Horgan J R, London S. A coverage analysis tool for the effectiveness of software testing[C]// International Symposium on Software Reliability Engineering, 1993. Proceedings. IEEE, 1993:25-34.

[3] Nie C, Leung H. A survey of combinatorial testing. ACM Comput Surv[J]. Acm Computing Surveys, 2011, 43(2):11.

[4] Weyuker E J, Jeng B. Analyzing Partition Testing Strategies[J]. IEEE Transactions on Software Engineering, 1991, 17(7):703-711.

[5] Barus A C, Chen T Y, Grant D, et al. Testing of heuristic methods: a case study of greedy algorithm[C]// Ifip Tc 2 Central and East European Conference on Software Engineering Techniques. Springer-Verlag, 2008:246-260.

[6] Chen T Y, Cheung S C, Yiu S M. Metamorphic testing: a new approach for generating next test cases[J]. 1998.

[7] Chen T Y, Huang D H, Tse T H, et al. Case Studies on the Selection of Useful Relations in Metamorphic Testing[C]// Ibero-American Symposium on Software Engineering and Knowledge Engineering. 2004.

[8] Cao Y, Zhou Z Q, Chen T Y. On the Correlation between the Effectiveness of Metamorphic Relations and Dissimilarities of Test Case Executions[C]// International Conference on Quality Software. IEEE, 2013:153-162.

[9] Mayer J, Guderlei R. An Empirical Study on the Selection of Good Metamorphic Relations[C]// Computer Software and Applications Conference, 2006. COMPSAC '06. International. IEEE, 2006:475-484.

[10] Asrafi M, Liu H, Kuo F C. On Testing Effectiveness of Metamorphic Relations: A Case Study[M]. IEEE Computer Society, 2011.

[11] 王瑢, 贲可荣. 蜕变关系构造基本准则与策略研究[J]. 计算机科学, 2012, 39(1):115-119.

[12] Liu H, Liu X, Chen T Y. A New Method for Constructing Metamorphic

Relations[C]// International Conference on Quality Software. IEEE Computer Society, 2012:59-68.

[13] Weyuker E J, Jeng B. Analyzing Partition Testing Strategies[J]. IEEE Transactions on Software Engineering, 1991, 17(7):703-711.

[14] Weyuker E J. On Testing Non-Testable Programs[J]. Computer Journal, 1982, 25(4):465-470.

[15] Chan W K, Cheung S C, Leung K R P H, et al. A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications[J]. International Journal of Web Services Research, 2007, 4(2):61-81.

[16] Chen T Y, Huang D H, Tse T H, et al. Case Studies on the Selection of Useful Relations in Metamorphic Testing[C]// Ibero-American Symposium on Software Engineering and Knowledge Engineering. 2004.

[17] Chen T Y, Tse T H, Zhou Z. Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing[C]// ACM Sigsoft International Symposium on Software Testing and Analysis. ACM, 2002:191-195.

[18] Zhou Z Q, Zhang S J, Hagenbuchner M, et al. Automated functional testing of online search services[M]. 2010.

[19] Sun C A, Wang G, Cai K Y, et al. Towards Dynamic Random Testing for Web Services[C]// Computer Software and Applications Conference. IEEE, 2012:164-169.

[20] Wang R. Researches on Basic Criterion and Strategy of Constructing Metamorphic Relations[J]. Computer Science, 2012, 39(1):115-119.

[21] Barus A C, Chen T Y, Kuo F C, et al. The Impact of Source Test Case Selection on the Effectiveness of Metamorphic Testing[C]// Ieee/acm International Workshop on Metamorphic Testing. IEEE, 2016:5-11.

[22] Cai K Y, Jing T, Bai C G. Partition testing with dynamic partitioning[C]// Computer Software and Applications Conference, 2005. COMPSAC 2005. International. IEEE, 2005:113-116 Vol. 1.

[23] Cai K Y, Gu B, Hu H, et al. Adaptive software testing with fixed-memory feedback[J]. Journal of Systems & Software, 2007, 80(8):1328-1348.

[24] Cai K Y, Hu H, and Ye F. random testing with dynamically updated test profile[J]. Software Reliability Engineering, 2009, Fast abstract 198.

[25] Ammann P E, Knight J C. Data Diversity: An Approach to Software Fault Tolerance[J]. IEEE Transactions on Computers, 1988, 37(4):418-425.

[26] Finelli G B. NASA Software failure characterization experiments[J]. Reliability Engineering & System Safety, 1991, 32(1–2):155-169.

[27] Segura S, Fraser G, Sanchez A B, et al. A Survey on Metamorphic Testing[J]. IEEE Transactions on Software Engineering, 2016, 42(9):805-824.

[28] Liu H, Liu X, Chen T Y. A New Method for Constructing Metamorphic Relations[C]// International Conference on Quality Software. IEEE Computer Society, 2012:59-68.

[29] Zhang J, Chen J, Hao D, et al. Search-based inference of polynomial metamorphic relations[C]// ACM/IEEE International Conference on Automated Software Engineering. ACM, 2014:701-712.

[30] Su F H, Bell J, Murphy C, et al. Dynamic inference of likely metamorphic properties to support differential testing[C]// Ieee/acm, International Workshop on Automation of Software Test. IEEE, 2015:55-59.

[31] Chen T Y, Poon P L, Xie X. METRIC: METamorphic Relation Identification based on the Category-choice framework ☆[J]. Journal of Systems & Software, 2016, 116:177-190.