中国科学院大学
University of Chinese Academy of Sciences

研究生学位论文中期报告

报告题目 **Android 应用程序函数级能耗估计**

学生姓名 卢琼 学号 **201328015029021**

指导教师 严俊 职称 副研究员

学位类别 工学硕士

学科专业 计算机软件与理论

研究方向 网络分布计算理论与技术

培养单位 中国科学院软件研究所

填表日期 **2015 年 12 月 15 日**

中国科学院大学制

填 表 说 明

1. 本表内容须真实、完整、准确。
2. “学位类别”名称填写：哲学博士、教育学博士、理学博士、工学博士、农学博士、医学博士、管理学博士，哲学硕士、经济学硕士、法学硕士、教育学硕士、文学硕士、理学硕士、工学硕士、农学硕士、医学硕士、管理学硕士等。
3. “学科专业”名称填写： “二级学科”全称。

报告提纲

一、学位论文进展情况，存在的问题，已取得阶段性成果

二、下一步工作计划和内容，预计答辩时间

三、已取得科研成果列表（已发表、待发表学术论文、专利等）

目录

1 研究背景

近些年来，Android 应用市场迅速扩大，应用程序功能激增，越来越多更有趣更多样化的功能被用户所喜爱。然而，其性能问题也日益凸显，Android 手机电池续航时间短，应用程序耗能太快，成为消费者对 Android 手机不满的主要原因。如何平衡 Android 应用程序的多样化与手机耗能的增加，成为一项亟待解决的问题。

研究人员已经探索了多种技术来降低 Android 应用程序的能耗，如提升硬件质量、改善 Android 操作系统的设计等，然而，平台层或硬件级别的改进往往是不够的，一个编码很差的应用程序将会无情的耗尽智能手机电池的寿命，如使用大量的重复计算、调用硬件资源却不释放等。因此，如何评估 Android 应用程序自身的能耗是非常关键的。

应用程序级节能技术也越来越受到研究人员的重视，但是传统的能耗相关研究更多关注系统组件的使用或二进制指令执行时的能耗，这对于开发者来说过于宏观，没有直观的帮助他们理解应用程序的能量消耗是如何在代码上体现的。大多数开发者都甚至不知道能量是如何消耗的，更不用说如何优化应用程序。本课题从开发者的视角，针对 Android 应用程序自身的行为，估计应用程序的函数级能量分布，给开发者更直观的认识，从而引导他们改进效率低下的代码，提升用户体验。

Android 操作系统的资源总量是非常有限的，而人们对于 Android 应用程序的性能要求却在不断提高，如何提升 Android 应用程序的性能已经成为一个亟待解决的研究课题，本文将基于自动化黑盒测试的角度着重研究 Android 应用程序的电量消耗问题。

2 学位论文进展情况，存在的问题，已取得阶段性成果

总体上，学位论文进展状况良好，在理论和实践两个方面，学位论文的进展状况都按照开题报告中的计划顺利进行。理论方面，已设计了 Android 应用程序能耗分析工具的整体框架；系统实现方面，已初步完成了该框架的原型工具的实现，能够对一些简单的 Android 应用程序进行测试与分析。下面是详细的进展情况，包括 Android 应用程序能耗分析现状研究、本文系统架构、存在的问题和已取得的阶段性成果。

2.1 Android 应用程序能耗分析现状研究

国内外针对于提升 Android 应用程序能耗分析方面的研究已有很多。Pathak 等[3]提出了 Eprof，一个细粒度的 Android 应用程序能耗分析器。它首先对待测 Android 应用程序进行插桩，然后利用 Android SDK 中自带的性能分析工具 TraceView。跟踪 Android 应用程序每个组件实时的能量消耗，通过将其映射到函数，计算每个组件分别消耗多少能量，还能计算应

用程序执行时某个线程消耗多少能量。其研究结果表明，造成手机用电巨快的罪魁祸首除了常驻后台的程序外，还有那些在免费程序中的广告。这些免费程序 65%至 75%的电能是由其内置的第三方广告模块所导致，这些广告模块在不启动时依旧将会存在于后台之中，消耗手机宝贵的电能。

Liu 等[5]提出了 GreenDroid，它基于 Java Path Finder，采用符号执行的方法，诊断能量利用率低的问题。GreenDroid 首先找到某些组件（如 GPS、CPU、Memory 等）的敏感数据作为污点，在 Java 虚拟机中运行 Android 应用程序，探索其应用程序状态。在应用程序运行期间，GreenDroid 监控污点的传播路径，将可疑的数据反馈给应用程序。这样，通过追踪应用程序运行时的污点数据传播，就能够分析出污点数据是如何在应用程序的不同状态下被使用的，从而找到 Android 应用程序的能量消耗在何处。

Li 等[6]提出了一种能够计算 Android 应用程序代码行级的能耗信息的方法。该方法分为运行时测量阶段（Runtime Measurement Phase）和离线分析阶段（Offline Analysis Phase）。它首先根据待测 Android 应用程序的控制流图（Control-flow Graph, CFG），对待测应用程序插桩，包括时间戳、线程 id、路径 id 等信息，然后将插桩后的应用程序在基于硬件平台的能耗检测工具 LEAP 上运行，同时收集运行时应用程序各组件（如 Wifi、GPS、CPU 等）的能耗信息；在离线分析阶段阶段，采用静态分析的方法，根据路径元组的相关记录，调整测量出的能耗值，排除系统 API 调用、线程切换等能耗值，然后通过线性回归求解出源代码行级的能耗值，最后通过可视化技术映射到客户端，使得开发者方便查看每行代码的能耗值，从而帮助改进应用程序的性能。Li 等的工作不同于以往的基于组件的或基于函数调用的技术，它结合了基于硬件的能耗测试方法和程序分析与数据建模技术，经过了细致的测量和分析，对 Android 应用程序能耗进行了细粒度的估计，能够计算出每行源代码的能耗信息。然而，当 Android 应用程序的规模较大，细粒度的估计应用程序能耗带来的误差代价将非常大，其逻辑控制复杂、对硬件精确度要求非常高，通常运行效率低下，所以细粒度模型的应用范围不广。

相关类似的研究还有很多，如 Shye 等[9]提出了基于组件利用率的功耗模型，根据组件各自的功耗特性，为每类组件的功耗单独进行建模。该方法虽提高了在线功耗模型估算的准确性，却增加了模型的复杂度和系统的估算成本。2010 年，Cuervo 等[10]也提出了一种基于组件的细粒度能耗测量方法，它基于硬件平台 Power Monitor 对各组件的耗能情况进行测量；2013 年，中山大学的岳川[13]测量 Android 应用程序中各功能函数的使用次数和所消耗的点亮，将获得的数据形成测试矩阵，并以此矩阵作为训练集。利用 SPSS 统计软件和 Excel 电子表格系统做统计分析，对测试矩阵做变量选择、回归分析和拟合建立多元线性回归模型，从而给出了耗电量最大的功能函数。他的研究结果为 Android 程序员优化代码提供了依据，也为最终达到减少耗电量的目标提供了新的思路。2014 年，四川大学的段林涛等[14]提出了一种基于应用程序运行时间的时间能耗模型，与高精度和复杂的应用程序组件能耗模型相比，该模型使用时间变量刻画多种因素，能够快速估算应用程序运行时移动终端产生的能耗。

在自动化测试工具方面，Android 平台黑盒测试比较常用的工具有 Monkey、Robotium[14]。Monkey 是 Android SDK 中自带的一个命令行工具，可以向系统发送伪随机的用户事件流（模拟用户的按键输入、手势输入、触摸屏输入等），通过设置测试对象的测试次数、频率、测试类型，实现对应用程序进行测试。Robotium 是一个基于 Android 应用程序的开源的自动化黑盒测试工具，其底层采用 Android 的 Instrumentation，它能够编写出功能强大、健壮性很强的黑盒测试用例。

2.2 系统架构

通过深入理解与本课题研究内容相关的基础知识，研究已有工作的研究成果及工具，打算采用动态测试与回归分析相结合的方式来实现工具，其系统框架流程如图 1 所示。从图中可以看出，工具的输入为 Android 应用程序 apk 文件，该工具主要包括以下 5 个模块：程序插桩、动态测试、数据预处理、回归分析、模型检验与能耗预测。每个模块的功能概述如下：

(1) 通过对 Android 应用程序插桩，嵌入与函数路径相关的信息；

(2) 通过 Monkey 无线控制 Android 应用程序自动化执行，同时使用 Trepn Profiler 监控能耗的使用情况，采集 Android 应用程序运行时信息；

(3) 通过对采集到的原始数据进行处理，提取特征信息，方便进行下一步回归分析；

(4) 通过采用线性回归，拟合出能耗模型；

(5) 采用交叉检验的方式，对拟合出的模型进行验证，预测给定路径的能耗。

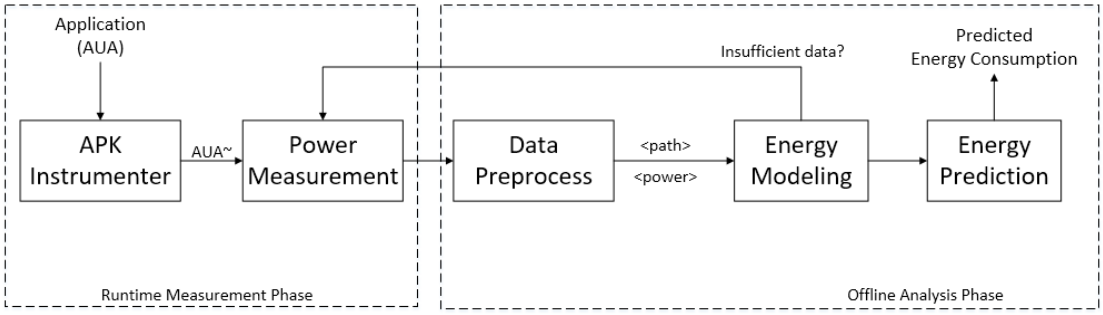

图 1 Android 应用程序能耗分析技术流程

下面将逐一对每个模块进行概述。

2.2.1 程序插桩

程序插桩（Instrumentation）是指在保证被测程序原有逻辑完整性的基础上在程序中插入一些探针，通过探针的执行，抛出程序运行时的特征数据，并通过分析这些数据，获得程序的运行时信息，来实现测试的目的。

Android 应用程序的插桩技术可分为源代码级插桩、中间代码级插桩:

(1) 源代码级插桩：针对 Android 应用程序源代码进行插桩，直接对源代码进行修改。

源代码级插桩具有较高的准确度，但是在确定插入桩代码的位置时需要精细的分析。Android 应用程序开发者通常不会公开源码，故而大多数 Android 应用程序的源码很难获取到，想要直接对源码进行插桩较为困难。并且，Android 应用程序多种多样，其代码结构、运行环境等都各不相同，如若对源码进行分析，那么配置程序将成为一项必要的步骤，这将使研究变的繁琐、耗时耗力。

(2) 中间代码级插桩：中间代码即 Dalvik 字节码，它是一种将 Android 可执行应用程序——apk 文件（Android Package）反编译后获得的中间代码，即 smali 文件。此种插桩技术针对 Dalvik 字节码进行插桩，通过分析 smali 文件的结构，探究 Android 应用程序的执行逻辑，定位到桩代码的位置，从而插入桩代码。与 JVM 是基于栈不同，Dalvik VM 是基于寄存器的，所以基于 Dalvik 的插桩必须考虑如何控制寄存器的使用。

考虑到中间代码的通用性，本课题采用基于 Dalvik 字节码的插桩技术，如何快速的定位到桩代码位置并有效的插入，使在插入有效信息后 Android 应用程序仍然能够正常运行是本课题一大难点。

Android 应用程序插桩技术主要包括三个步骤，分别为反编译、植入探针信息、重打包重签名。

1. 反编译

将 apk 文件反编译为 Dalvik 字节码，即 smali 文件。反编译以后，会在工程目录下生成一个 smali 文件夹，里面存放着所有反编译之后生成的 smali 文件，这些文件会根据源程序包的层次结构生成相应目录，包括资源文件、源码文件、配置文件等。源程序中所有类（包括普通类、抽象类、接口类、内部类）都会在相应目录下生成以该类名命名的独立 smali 文件，它们都将以单独的 smali 文件来存放。在程序插桩时，我们只要关注 smali 文件即可。

2. 植入探针信息

本课题想要在运行时能够获得与函数路径相关的信息，并且使运行时开销影响尽可能小。所以我选取在每个类的每个函数入口处插入函数类名和函数名，这样既能够保证所有的函数都被记录，并且每个函数都只插入一行代码，使其较小的影响运行时开销。通过 Dalvik 的字节码我们不能直接看到原来的逻辑代码，这时可以借助如 Apktool 或 dex2jar+jd-gui 工具来帮助分析查看。首先，在 smali 文件夹下找到所有需要插入函数路径信息的类，在每一个需要插桩的类中找到需要植入探针的所有函数，最后，在这些功能函数中植入相关的函数路径信息。

3. 重编译重打包

重编译重打包技术与反编译技术是相互对立的，反编译是将 apk 解析成可修改的 smali 文件，在修改完 smali 文件之后，将其重新编译、再经过重新签名、重打包，重新生成可执行的 Android 应用程序。虽然形式跟原来相同，都是一个 apk 安装包，但是此时的应用程序已经嵌入了我们想要的函数路径信息，并且不会影响程序自身的功能。

在完成 Android 应用程序插桩之后，将对插桩后的应用程序进行动态测试。

2.2.2 动态测试

软件测试可分为静态分析和动态测试。静态分析是指在不运行代码的方式下，通过词法分析、语法分析、控制流分析、数据流分析等技术对程序代码进行扫描，由于其不执行实际程序，往往很难发现代码运行时的很多问题。与静态分析不同，动态测试是通过在真实环境或模拟环境中直接执行程序进行分析，对于研究程序运行时的行为很有帮助。

本课题的目的是为了研究 Android 应用程序运行时的能耗情况，显然单纯的静态分析是远远不够的。为了获取 Android 应用程序的真实能耗，需要研究程序在实际运行时的行为，这就需要大量的运行时数据作为支撑，所以本课题要采用动态测试的方法。

目前国内外文献中大多使用 Robotium 测试工具来控制 Android 应用程序的执行过程，然而 Robotium 需要测试者自行生成测试工程并构建测试用例来引导测试的执行，这对于少数实验确实是有效的。然而，动态测试往往具有大量的测试需求，这就需要测试者耗时耗力的编写测试用例并执行。并且，测试用例的构建需要测试者对应用程序的功能具有一定的了解，才能编写出有效的测试用例。所以此处我们使用 Monkey 来控制 Android 应用程序进行随机测试，它简单易用，可通过指令控制事件的执行。

在能耗测量工具上，本课题选取 Trepn Profiler 监测应用程序的能耗。它可监测设备本身，也可监测单独的应用程序。可显示应用程序实时的能耗情况，也可以监测组件（如 CPU，WIFI，蓝牙等）的能耗情况。本课题使用 Trepn Profiler 来监测待测 Android 应用程序的能耗，相对于直接监测设备本身而言，可省去很多不必要的误差，如多进程等。

动态测试模块的架构图如下图所示：

图 2 动态测试模块架构图

目前国内外文献中多采用虚拟机来进行实验，虽然实验结果较为理想，但其实很多自然因素导致的误差是真实存在的，他们并没有考虑在内。本课题采取真机进行动态测试，由于智能手机芯片的原因，Trepn Profiler 能够支持的设备有限，并且能耗的测量需要在智能手机断电的情况下进行，所以我们使用无线调试状态，通过 Monkey 来控制 Android 应用程序执行，与此同时，使用 Trepn Profiler 监控能耗情况。

插桩后的应用程序在运行时会生成路径日志，通过提取日志的信息，我们可以分析出待测应用程序的行为特征。同时，每一组行为特征对应着一组能耗测量，多组测试后，可以获得大量相应的实验。我们可以将待测应用程序的路径信息转换为数值，同时，选取有代表意

义的能耗值作为一次测量的耗能，如此，我们可提取出待测应用程序的运行时特征。

在完成动态测试之后，将对动态测试产生的特征数据进行数据预处理。

2.2.3 数据预处理

在动态测试模块中，获取的数据量非常大，我们需要在众多数据中筛选提取出对后续分析有用的数据。动态测试模块得到的数据主要有两类，路径运行时信息和能耗信息。

1. 路径运行时信息提取

在动态测试时，Monkey 在控制 Android 应用程序执行的同时，在后台生成了 log 文件，log 文件中包含系统基本信息、时间信息、虚拟机信息等，我们插桩运行后生成的运行时信息也在此文件中。随着应用程序的执行，该文件会依次记录程序执行的函数信息。我们可以将每一次执行时，各个函数的执行次数统计出来。

2. 能耗信息提取

Trepn 监控 Android 应用程序运行结束后，会生成一个数据库文件。每一次动态测试都会对应生成一个数据库文件，通过扫描数据库中的表和字段，可以找到我们需要的能耗信息。

统计出上述两种特征信息后，我们就可以选取有效的模型进行数据分析了。

2.2.4 回归分析

通过大量阅读文献，并针对本课题的实际情况，考虑到线性回归模型的通用性，本课题首选采用线性回归模型。其中，函数路径特征对应线性回归方程中的自变量，能耗测量值对应线性回归方程中的因变量，拟合出的模型参数即为每个功能函数的能耗，由此可以得到能耗模型，如下所示：

$$x_{i1}w_1 + x_{i2}w_2 + \cdots + x_{in}w_n = e_i$$

其中，i 代表第 i 次测试，w 代表函数的能耗，x 代表其相对应函数的函数执行次数，e_i为第 i 次测试的能耗测量值。

本课题基于梯度下降算法提出了一种改进的算法，引入修正值 u 作为模型的调节系数，通过迭代的方法逼近最小偏差模型，算法如下：

Algorithm 2 Improved Gradient Descent Algorithm

Input: X, E, α, η

Output: $\boldsymbol{w}$

$\boldsymbol{w} \leftarrow 0, \boldsymbol{u} \leftarrow 0$;
repeat
    Give the prediction $\hat{e_t} = x_t \cdot w_t^T + u_t \times e_t$
    Update $\boldsymbol{u}$ according to rule $u_{t+1} = u_t - \eta \nabla_u$
    if any value in $w_t > 0$ **then**
        Update $\boldsymbol{w}$ according to rule $w_{t+1} = w_t - \alpha \nabla_w$
    end if
until $J(w, u)$ convergence

2.3 实验验证

本课题通过以下四个方面来验证方法的正确性，分别为插桩开销分析、模型评估、交叉检验、高能耗函数分析。

2.3.1 插桩开销分析

为了估计程序插桩对应用程序能耗的影响，我们做了一些实验来评估程序插桩的开销。如图 3 所示，共有 10 组实验数据，每组数据包含插桩前能耗、插桩后能耗，每组数据采用同样的随机数种子，生成相同的程序执行路径，从而保证相同的两组数据是在相同的条件下产生。

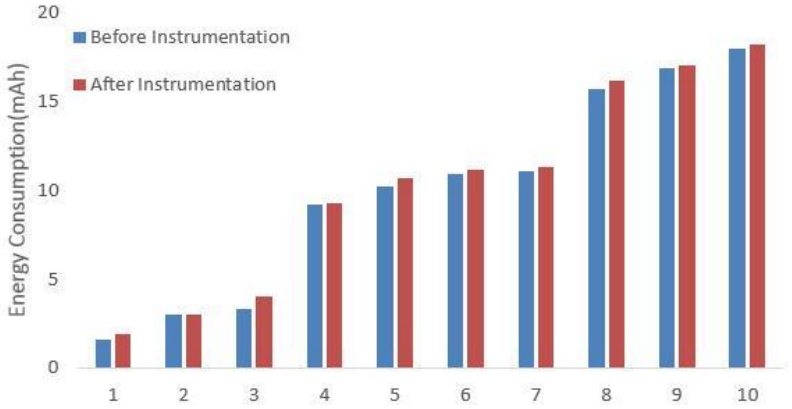

图 3 插桩开销分析

由图 3 可以看出，同种条件下，插桩前后能耗相差不大，可见程序插桩对应用程序的能耗影响较小。

2.3.2 模型评估

通过阅读文献，为了方便与文献中数据进行比对，本课题采用文献[6]中的方法对模型进行评估，分别为累积估计误差（Accumulated Estimating Error, AEE）和复相关系数（multiple correlation coefficient）。

（1） 累积估计误差（Accumulated Estimating Error, AEE）

当利用递推公式对各部分计算结果进行累加时，其误差也会随之累加，最后所得到的误差总和称为累计误差。它通常在[0,1]之间取值，其值越小代表模型的累积误差越小。

（2）复相关系数（multiple correlation coefficient）

复相关系数是测量一个变量与其他多个变量之间线性相关程度的指标，复相关系数越大，表明要素或变量之间的线性相关程度越密切。

本课题选取 4 个应用程序对模型进行评估，以及文献[6]中 4 个应用程序进行数据比对，

如表 1 所示，前 4 个 App 是本课题选取对实验程序，后 4 个 App 是文献[6]中的实验程序。#Class 代表应用程序所含有的类的个数，#Method 代表应用程序所含有的方法的个数，#Inst 代表应用程序中插入桩的个数。Analysis Time 是指实验所耗费的时间，以秒(s)为单位，T_I 代表插桩花销的时间，T_A 代表线性回归花销的时间。Analysis Accuracy 采用如上所述的两个参数进行评估。

由表 1 可以看出，在程序规模与文献[6]相差不多的情况下，本课题更加高效，并且精度与其相差不多。当实验规模较大时，本课题的方法依然适用，文献[6]对此情况没有分析。

表 1 模型评估

App	#Class	#Method	#Inst.	Analysis Time		Analysis Accuracy		Description
				$T_I(s)$	$T_A(s)$	R^2	AEE(%)	
Saolei	132	724	538	1	46	0.88	5.3	Game
CRadio	668	4252	2677	4	106	0.86	8.2	Radio
SogouBrowser	2970	18542	13430	31	186	0.89	7.9	Web browser
SogouRead	3331	22519	16183	44	235	0.78	9.9	Reader
Classic Alchemy	751	4434	–	873	128	0.93	3.4	Game
Bubble Blaster	932	6060	–	460	145	0.90	8.6	Game
Skyfire	684	3976	–	277	97	0.99	4.8	Web Browser
BBC Reader	590	4923	–	353	158	0.94	6.5	RSS reader

2.3.3 交叉检验

除用 2.3.2 中的参数评估模型的效率与精度外，本课题还采用交叉检验的方法对实验进行验证。本课题再通过 2.2.2 中所述动态测试的方法，分别对 4 个应用程序进行 50 次额外的实验，以获取测试集。以下是交叉检验的评估结果，蓝线代表实验的能耗测量值，红线代表通过模型预测的能耗预测值。

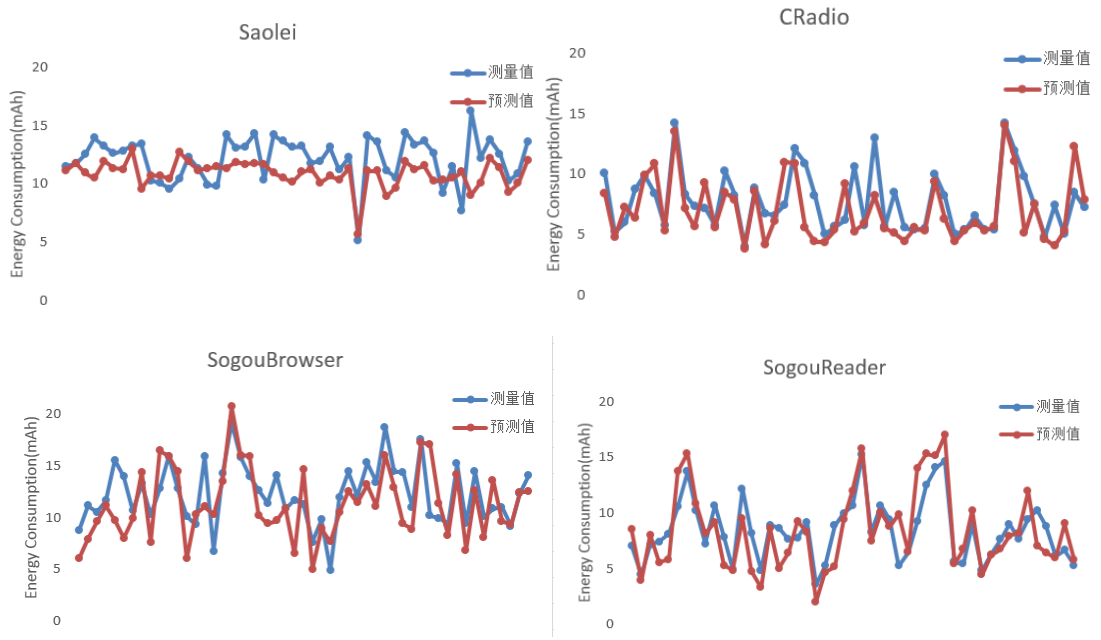

图 4 交叉检验

由图 4 可以看出，通过模型得到的能耗预测值大体趋势与测量值相吻合。表 2 列出了每组实验预测值与测量值的平均误差(Average Error)和标准差(Standard Deviation)。

App	Average Error(%)	Standard Deviation(%)
Saolei	14.3	1.69
CRadio	19.3	2.56
SogouBrowser	19.5	3.20
SogouReader	18.3	3.14

由表中数据可知，模型的平均误差大约在 10%-20%之间，数据偏离不大。

2.3.4 高能耗函数分析

为了进一步分析实验结果，我们对模型中计算出的高能耗函数的中间代码进行检查，图 5 是对选取的 100 个高能耗函数中所进行的操作分类结果。

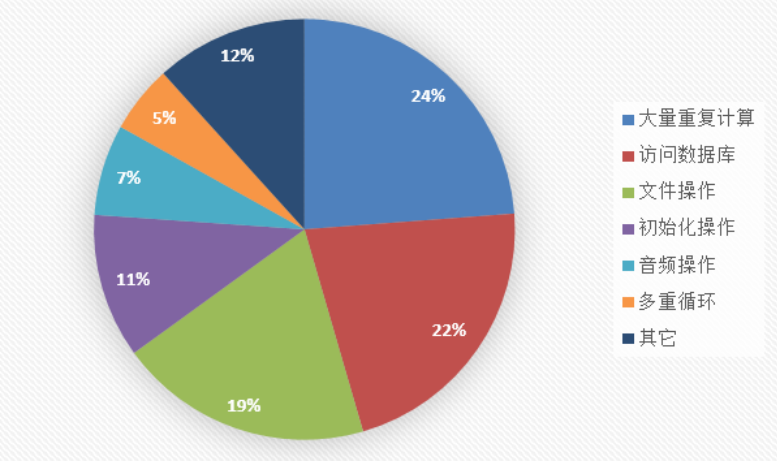

图 5 高能耗函数分析

由图 5 可以看出，从函数级别看，具有大量重复计算、访问数据库、文件等操作的代码更加耗能。

2.4 存在的问题

目前工具的设计和实现中存在的问题有如下几点：

（1）程序插桩部分：在插入桩代码时，对寄存器对控制不是很好，导致某些应用程序插桩后不能执行或程序有所变动。

（2）动态测试部分：测试的实验数量有所不足，对每组实验进行更多次测试，并通过比对不同次数的测试数据，给出合适的测试次数，可使方法在软件测试领域更受欢迎，并且可引导得到的模型更加精准和可信。

（3）线性回归部分：梯度下降算法的步长目前采用定值，虽然通过反复调参对精度影

响不大，但是影响实验效率，且消耗人力、时间。若能改成动态调整步长的方法，实验效率将会大大提升。

3 下一步工作计划和内容，预计答辩时间

3.1 下一步工作计划和内容

（1）完善 Android 应用程序能耗分析的整个框架，包括能耗模型生成及能耗路径生成。框架需要有良好的模块划分，能够方便地在其上实现各种不同的算法改进。

（2）完善实验的数据分析，从多种角度验证能耗模型。

（3）学位论文撰写

3.2 预计答辩时间

2016 年 5 月

4 参考文献

[1] “手机中国” URL: http://www.cnmo.com/news/423626.html

[2] A. Bertolino. Software Testing Research: Achievements, Challenges, Dreams. In Proceedings of Future of Software Engineering, 2007, pp. 85-87.

[3] P. Peterson, D. Singh, W. Kaiser, and P. Reiher. Investigating energy and security trade-offs in the classroom with the atom leap testbed. In 4th Workshop on Cyber Security Experimentation and Test (CSET), pages 11–11. USENIX Association, 2011.

[4] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof. In Proc. of EuroSys, 2012.

[5] Liu, Y., Xu, C., and Cheung, S. C. 2013. Where has my battery gone? Finding sensor related energy black holes in smartphone applications. In Proc. IEEE Int’l Conf. Pervasive Computing and Communications. PERCOM '13. 2-10.

[6] Ding Li, S. Hao, William G. J. Halfond, R. Govindan. Calculating Source Line Level Energy Information for Android Applications. International Symposium on Software Testing and Analysis, ACM, 2013, 78-89.

[7] Y. Liu, C. Xu, and S. Cheung. Characterizing and detecting performance bugs for smartphone applications. In Proc. of the International Conference on Software Engineering, ICSE ’14, pages 1013–1024, 2014.

[8] N. Balasubramanian, A. Balasubramanian, A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In Proc. of the 9th ACM SIGCOMM conference on Internet measurement conference, ACM, 2009, 280-293.

[9] A. Shye, B. Scholbrock, G. Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, ACM, 2009, 168-178.

[10] E. Cuervo, B. Aruna, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “Maui: Making smartphones last longer with code offload,” in MobiSys, 2010.

[11] R. Palit, R. Arya, K. Naik, and A. Singh. Selection and execution of user level test cases for energy cost evaluation of smartphones. In Proceedings of the 6th International Workshop on Automation of Software Test, ACM, 2011, 84–90.

[12] 岳川. 基于程序动态分析的 Android 手机应用耗电量研究. 中山大学, 2013.

[13] 段林涛, 郭兵, 沈艳, 王毅, 张文丽, 熊伟. Android 应用程序能耗分析与建模研究. 电子科技大学学报, 2014.

[14] 张立芬, 周悦, 郭振东, 北京软件产品质量检测检验中心. Android 移动应用测试[J]. 中国新通信, 2013, 3:058.

[15] 朱少民. 软件测试方法和技术. 北京: 清华大学出版社, 2005.

[16] P. Mathur. Foundations of Software Testing. 北京: 机械工业出版社, 2008.

[17] H. Jiawei, P. Jian et al. Data mining: concepts and techniques, 2005.

[18] P. Harrington. Machine Learning in Action, 北京: 人民邮电出版社, 2013.