南京大学

本 科 毕 业 论 文

院　　系　计算机科学与技术系

专　　业　计算机科学与技术

题　　目　面向场景的移动应用模型融合技术

年　　级　2017 级　　学 号　171860687

学生姓名　钱正轩

指导教师　张天　　职 称　副教授

提交日期　2021 年 6 月

面向场景的移动应用模型融合技术

Scenario-oriented mobile application model fusion technology

南京大学本科生毕业论文（设计、作品）中文摘要

题目：面向场景的移动应用模型融合技术

院系：计算机科学与技术系

专业：计算机科学与技术

本科生姓名：钱正轩

指导教师（姓名、职称）：张天 副教授

摘要：

当今时代，移动应用已成为软件开发的主流。移动应用的主要特点是事件驱动的设计模式，应用的执行流程取决于用户产生的事件。模型是软件工程开发周期中一个重要的构件，移动应用的事件驱动特点为其建模带来了难度。

在为移动应用进行建模时，人工对应用进行探索虽然符合用户使用逻辑，但是探索得到的模型并不完善，且成本较高；使用自动化工具对应用进行建模会具有较高的探索度，但执行序列较为混乱。需要将两种模型的优点综合起来，得到更加完善的模型。

本文提出了一套面向场景的模型融合的算法和工具，可以将人工探索的模型与工具探索的模型进行合并，同时将人工探索模型的场景标注信息泛化到合并后的模型中。借助此算法，我们可以以较低的成本构建规模较大且信息充足的移动应用模型。

本文的主要工作包括：

1. 提出了一套面向场景的模型融合的算法
2. 使用 Python 实现该算法的原型工具
3. 设计实验评估了工具的融合效果

关键词：移动应用；模型合并；标签泛化

南京大学本科生毕业论文（设计、作品）英文摘要

THESIS: Scenario-oriented mobile application fusion technology

DEPARTMENT: Computer Science and Technology

SPECIALIZATION: Computer Science and Technology

UNDERGRADUATE: Zhengxuan Qian

MENTOR: Associate Professor. Tian Zhang

ABSTRACT:

Today, mobile applications have become an important part of modern software development. The main feature of mobile applications is event-driven programming, which means the flow of the program is mainly determined by events generated by the user. The model is an important artifact in the development cycle of software engineering, and the event-driven characteristics of mobile applications make it difficult to model.

When modeling mobile applications, although manual exploration is in line with the user's logic, the model obtained by the exploration is not perfect and the cost is high. Using automated tools to model the application will have a higher degree of exploration. However, the execution sequence is more disorderly. It is necessary to combine the advantages of the two types of models to obtain a more complete model.

In this paper, we propose an algorithm and a set of tools for scenario-oriented model merging, which can merge the manually explored model with the model explored by automated tools, and generalize scenario annotations of the manually explored model to the merged model. With these tools, we can build a large-scale and well-informed mobile application model at a much lower cost.

The main work of this paper includes:

1. Proposed a set of scenario-oriented model fusion algorithms.

2. Implemented the prototype tool of the algorithm using Python.

3. Design experiments to evaluate the effect of tools.

KEY WORDS: Mobile Applications; Model Merge; Label Generalize

目录

1 引言

1.1 研究背景

在当今时代，移动应用飞速发展，其规模和复杂程度都在不断增长。区别于传统的命令行应用，移动应用的一大特点在于其事件驱动的模式，即用户在使用移动应用时与界面交互，不同的交互动作（如点击、长按、滑动……）会触发不同的事件，移动应用在捕获到事件后会执行相应的处理动作、改变应用的状态、从而改变所展示的界面，用户会再次在新的界面上触发事件进行交互，这样的循环构成了移动应用的基本运行场景。移动应用不同的状态间由事件触发的跳转构成了应用的控制流。

在软件工程的开发生命周期中，模型是一个非常重要的构件，模型的质量对软件开发有至关重要的影响。移动应用事件驱动的特点导致其执行路径相对于一般的程序会更加复杂多变，不同的交互事件触发的应用状态的跳转构成了一个复杂的图结构，相应地为移动应用的建模带来了难度。

在对移动应用进行建模时，常见的方法是人工对其进行探索，触发尽可能多的事件跳转，或是使用自动化测试工具，如 Monkey[1]，UI Automator[2]等自动化框架。不同的测试人员以及不同的测试工具在对同一个移动应用进行建模时会产生不同的遍历路径和图，得到不同的移动应用模型。测试人员人工探索的模型往往不够完善，但是更加符合用户使用应用的逻辑；测试框架生成的模型往往覆盖度较高，但是执行序列较为混乱。

随着移动应用开发逐渐成为现代软件开发的主流，移动应用模型的重要性也逐渐凸显出来，目前已经有一些工作针对移动应用的模型建立模型数据库[3]，但人工建立移动应用的模型数据库成本太大，而机器探索得到的模型信息不足。本文提出了一种面向场景的模型合并方法与工具 SMF(Scenario-oriented Model Fusion)，可以将人工建立的带有场景功能信息的模型与机器探索的模型合并，并且将场景功能信息泛化到合并后的模型上，从而能够通过合并模型的方式，使用少量的带标注的模型与大量无标注的模型泛化得到大量带有标注的模型，以较

低的成本构建拥有较多信息，同时具有一定规模的移动应用模型库，便于后续的研究以及开发工作。

1.2 相关工作

1.2.1 移动应用数据库

本文工作受到了 Biplab Deka 的工作[3]的启发，这篇文献提出了一个移动应用的数据库 Rico，其收集了约 9700 个 Android 应用，从中获取了超过 10000 条用户交互轨迹以及超过 70000 个不同的 UI，涵盖了 27 个不同的种类。Rico 收集的数据包括了应用 UI 的截屏、布局的结构、用户与应用的交互、应用元数据、UI 相似度的标注等，这些数据可用于实现数据驱动的移动应用设计。

Rico 的数据通过测试人员手动探索与自动化工具相结合的方法收集。Deka 基于 ERICA[4]开发了一套群众外包探索系统，可以让测试人员通过网络操作实机进行遍历，收集数据时测试人员用于尽可能解锁更多应用状态（如一些需要登陆/验证等输入才能解锁的状态），而自动化工具会对解锁的状态中的可交互元素进行遍历，如果遇到需要输入的状态则使用之前人工探索时的输入。

Rico 这种人工探索与工具探索结合的方法取得了较好的效果，也给了我们启发：是否可以将只有人工探索能得到的信息泛化至工具探索的结果上，以较低的成本产生大量的数据供下游工作使用。本文提出的面向场景的模型合并工具 SMF 是对这种思路的一种尝试。

1.2.2 协同建模与合并

随着现在软件的复杂度逐渐上升，在为软件建模时单个建模者的能力逐渐达到了极限，多人协同建模成为了为软件建立高质量模型的解决方案。现在也有大量工作对模型融合技术进行研究。

基于共生图的推荐算法 CGRA[5]，提出了一种协同建模的系统，其中每个个体建模者可以建立自己的模型，同时从系统中获取推荐，而个体的模型也可以提

升系统的推荐效果。基于共识主动性（stigmergy）的多人协同建模[6][7]，使用类似蚁群的间接交互方式进行建模者的信息交流，使用合并反馈的机制提升模型质量。基于模型转换的合并技术[8]是进行基于元模型的模型转换，使用转换的结果辅助模型的合并。

这些研究工作也为未来对 SMF 工具的改进与进一步开发提供了思路，工具的合并也可以不仅限于将人工模型的信息泛化到机器模型中，而是实现不同测试人员之间的模型的互相补充与完善，或者是进行交互式的协作建模以及合并。

1.2.3 模型一致性

在软件工程的开发过程中，模型驱动工程[9]（MDE, Model-Driven Engineering）是一个重要的分支。在 MDE 中，模型的重要性得到了提高，成为了软件开发的主体，在开发过程中，往往会基于多种不同的元模型进行建模，并进行模型的生成、映射与转换。模型的转换主要分为两类：模型与模型间的转换；模型到代码的转换，在开发过程中势必会对模型做出改动，而不同的模型以及模型和代码之间如何对这种改动保证一致性也是一个重要的研究课题。

文献[10]提出了一种用于合并不同版本的模型的方法，可以在一致的情况下自动合并模型，并且如果出现了不一致或冲突，会向建模者提供可行的修复方案。由于需要修复的部分来源于对原始模型的修改，故修复方案的规模也被限制在一个可行的范围内。文献[11]则是分析并研究了一系列针对统一建模语言（UML, Unified Modeling Language）的模型的一致性维持方法。文献[12]提出了一种针对 UML 模型的合并时的冲突解决方案。

除此之外，也有工作[13]对模型的演化造成的不一致进行恢复做出了研究，或是研究 MDE 开发过程中的模型组合问题[14]。上述的对于 MDE 中模型一致性问题的研究也对本文的工作提供了启示，本文研究的重点虽然是模型的融合，但其中也涉及了不同来源的多个模型，模型间同样存在不一致的问题。对模型一致性问题的研究为工具未来的改进和进一步开发提供了启发与方向。

1.3 本文工作

在本文的工作中，我们提出了一套面向场景的模型融合方法 SMF，首先将通过不同途径得到的移动应用的执行过程建模为一个抽象的平台无关的应用执行路径（ARP, App Running Path）模型，然后将人工探索得到的 ARP 模型与机器探索得到的 ARP 模型合并。在合并的过程中，人工模型中的场景功能信息标注会被泛化到合并后的模型上，从而得到一个含有更多场景标注信息的 ARP 模型。

本文选择开源的 Android 应用作为研究的目标，SMF 整体的框架可划分为以下几个子模块：

- 预处理模块
- 合并模块
- 状态相似度判断模块
- 跳转相似度判断模块
- 图遍历模块

本文还基于 SMF 方法实现了对应的原型工具，在工具中实现了三种不同的状态相似度模块，并将其与工具解耦，实现相似度算法的可插拔可配置。本文设计了相应的实验，测试比较了这三种不同相似度算法的效果。

工具的处理流程见图 1

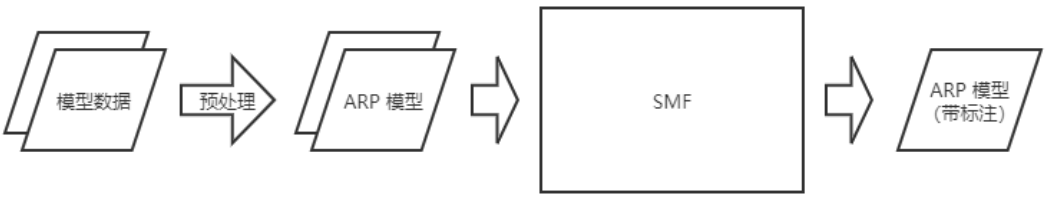

图 1 工具流程示意

2 技术背景

2.1 ARP 模型

应用执行路径（ARP, App Running Path）模型是一个平台无关的、抽象的概念模型，其刻画了移动应用及其运行时的状态。ARP 模型采用以下模块对移动应用进行建模：

- 应用的元信息
- 应用的状态集合
- 应用的执行路径
- 应用的场景信息

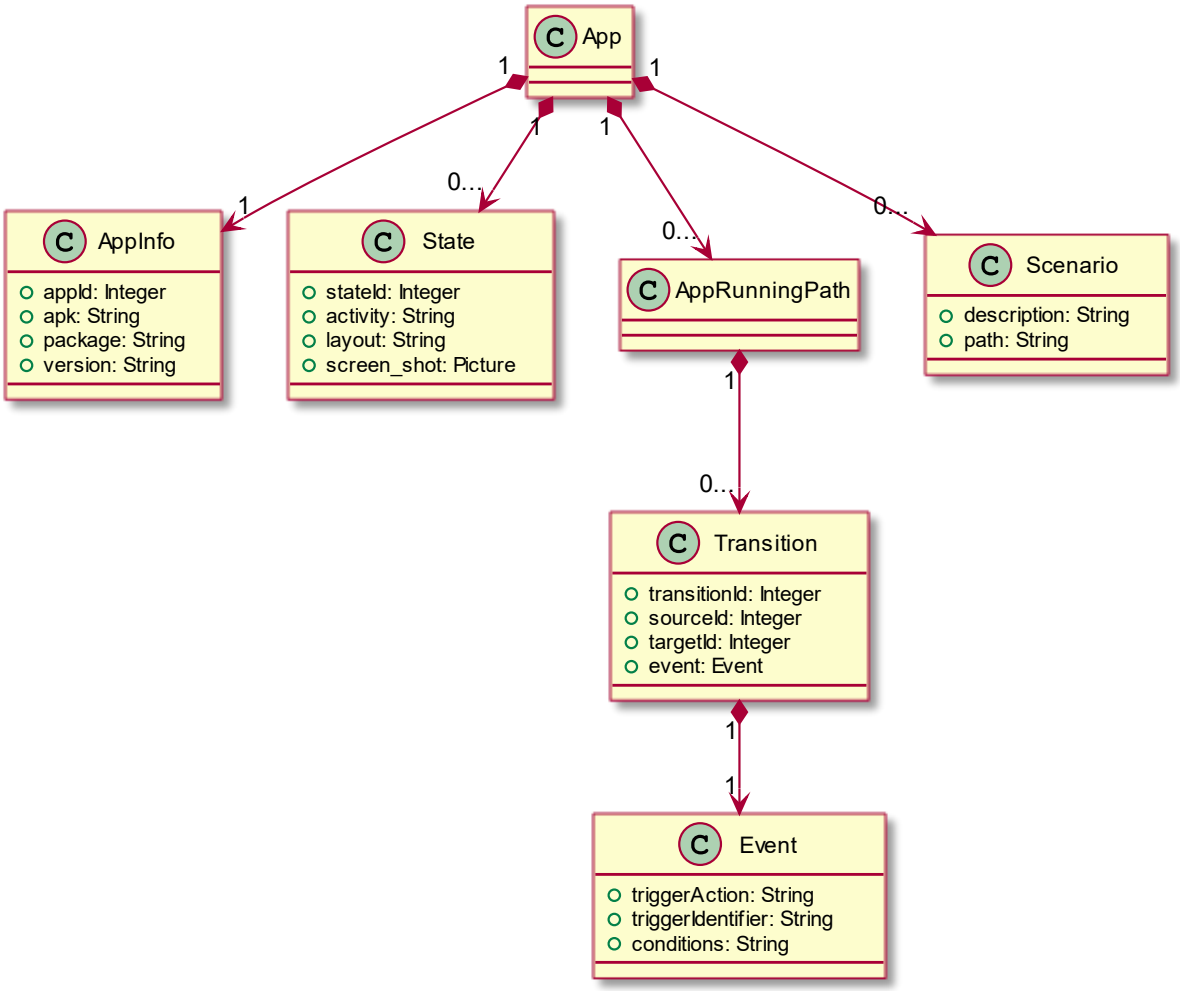

图 2 ARP 概念元模型

应用元数据是移动应用最基本的数据，包含了应用的名称、概要、版本、开发者等一系列可用于不同维度查询筛选使用的信息。

应用的状态集合是当前应用所有状态（state）的集合。由于移动应用事件驱动的特性，以及移动设备以屏幕为主要交互设施的特点，移动应用承载交互与展示功能的界面刻画了应用当前的执行状态。在具体实现 ARP 模型时，用两类信息刻画移动应用某一执行时刻的状态信息：展现出的截屏文件，以及用以描述当前界面布局结构的布局文件。除此之外状态也可以包含更多的信息，如对于 Android 应用，状态中也可以包含当前界面所属的 Activity 信息。

应用的执行路径是一系列跳转（transition）的集合。跳转关系是一个三元组<source, target, event>，其中 source 与 target 分别表示跳转的源状态与目的状态，而 event 表示了触发该次跳转的具体交互动作。event 中的信息包括了交互动作的交互对象、交互类型、交互区域等。

应用的场景信息是一系列带标签的路径的集合，路径是跳转的集合，而标签信息指示了该路径属于应用某个具体的功能场景。这也是本文工作中需要泛化的部分。

2.2 移动应用 UI 布局

不同平台的移动应用虽然有不同的平台差异，但其布局控制遵循相同的理念。移动应用的一个界面通常由若干组件组成，组件包括控件与容器。控件有具体显示内容，如文字、图片、按钮、输入框等；容器本身不可见，但可以将控件或其他容器以某种规则组合起来，如表格、列表、多栏内容等。

组件之间的嵌套关系天然形成了一个树型结构，一个移动应用的界面通常由一个容器作为根，组件的嵌套关系体现为树中的祖先后代关系。这种树结构约束了组件间的包含关系。

大多数移动应用开发使用 XML 文档作为布局的描述格式，组件体现为文档中的元素，组件的嵌套关系体现为元素的嵌套关系，组件的属性可以作为元素的属性，也可以单独作为一个 XML 元素。应用可以使用预先定义的布局文档，也可以在运行时动态地添加组件。现在主流的移动应用自动化测试框架均提供了工具用

以在测试时动态查看当前的布局信息，并且可以将布局信息持久化为外存中的文件（通常同样为 XML 格式）。持久化的动态布局信息可以作为 ARP 模型的状态的布局信息。

2.3 自动化测试

在对移动应用进行测试时，常见的方法是基于某些测试框架，编写框架能够接受的脚本文件或是调用框架提供的接口，驱动在虚拟机或真机上运行的应用进行执行跳转。框架通常会提供许多便于测试的功能，包括对待测试应用的安装与启动，模拟用户的输入（点击、长按、滑动，键入文本……），获取待测应用运行时的信息，获取系统信息，模拟系统事件等。编写脚本调用这些功能，即可实现更为复杂的测试序列。驱动测试框架的可以是人工编写的测试脚本，也可以是计算机程序基于某些策略生成的交互动作序列。测试框架同样可以用于对移动应用进行建模：通过测试框架驱动移动应用进行跳转，探索并记录应用不同的状态。接下来将介绍一些主流的自动化测试框架。

2.3.1 UI Automator

UI Automator[2]是由谷歌开发，包含于 Android SDK 中的一个 UI 测试框架。UI Automator 基于界面，可以跨应用进行界面测试，其测试不依赖于被测试应用的源代码，常用于黑箱测试。

UI Automator 提供了一个图形化脚本工具 UI Automator viewer，该工具可以捕获并保存当前 Android 设备的界面信息，包括截屏文件与布局文件。也可以再次加载之前捕获过的文件。通过该工具，可以较为直观地观察当前界面与布局结构之间的联系，辅助编写测试脚本。

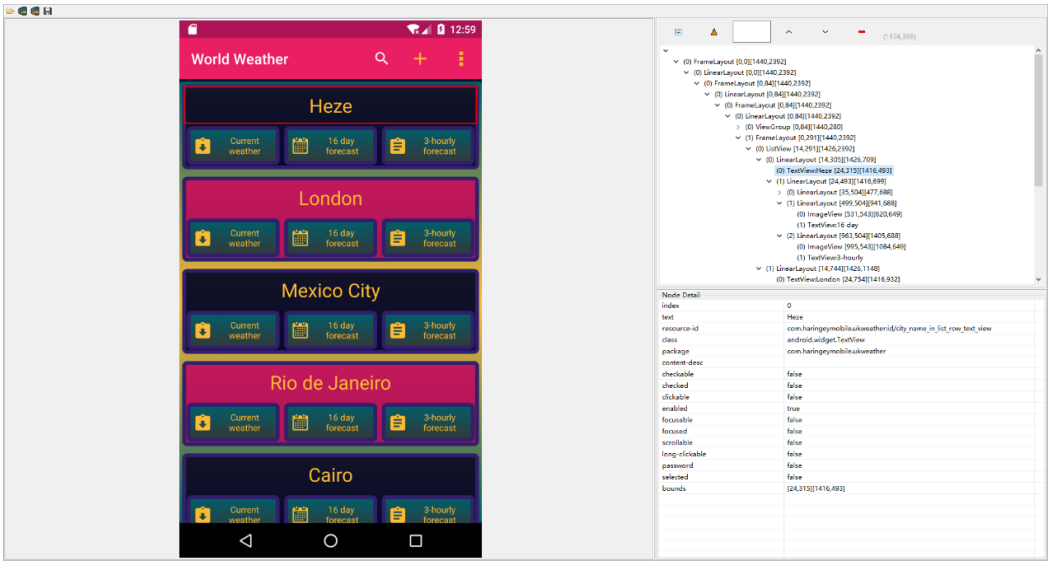

图 3 UI Automator Viewer

UI Automator 的功能由 Java/Kotlin 编写的类库以及相关 API 提供，同样有开源项目[15]用 Python 封装了 UI Automator 的功能，可以用 Python 编写自动化测试脚本。

2.3.2 Appium

Appium[16]是一个开源的自动化测试框架，可以自动化测试 Android，iOS，Windows 平台上的原生或混合应用，以及移动 web 应用。

Appium 是个封装度更高的框架，其本身的设计为跨平台使用，可以用一套 API 测试多个平台的应用，其底层调用的是不同平台原生的自动化测试框架，如对于 iOS 调用苹果公司的 XCUITest，而对于 Android 调用 UI Automator。

Appium 框架本身采用了客户端-服务器架构。Appium 服务器承担主要的驱动工作，调用平台原生框架进行测试和动作的执行。Appium 客户端与服务器通过 http 协议连接，负责控制服务器进行测试。任何能实现 http 客户端的语言都可以实现 Appium 客户端，现在已经有许多使用主流编程语言实现的 Appium 客户端，包括 Java，Python，Ruby，C#等。

Appium 的桌面端程序除了为服务器封装了一个图形化界面外，同样提供了一个 Inspector，可以查看应用程序运行时的布局层级，便于编写测试脚本。

Inspector 同时也能以交互式的方式执行简单的测试动作，如点击和滑动等。

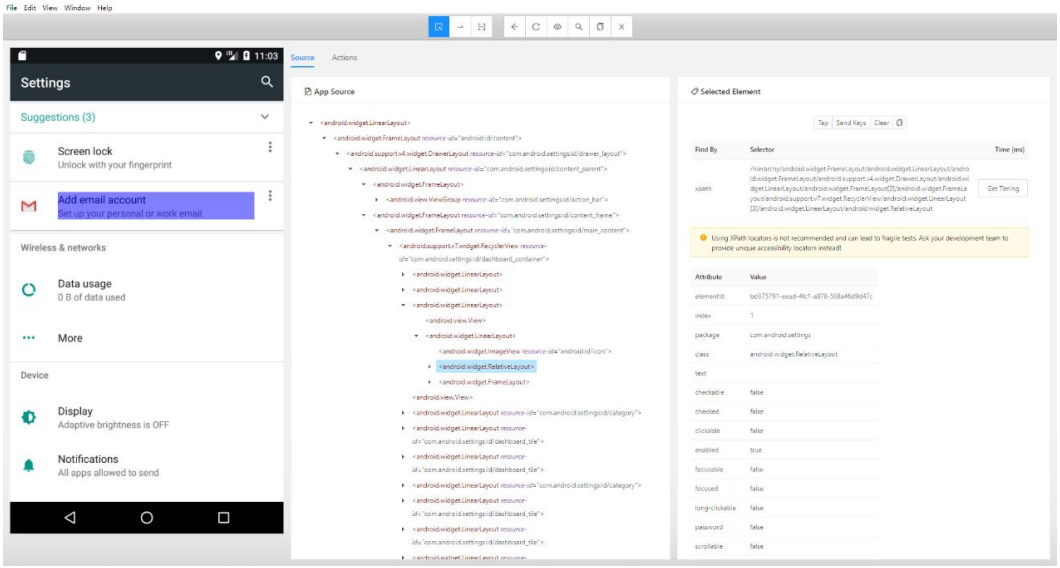

图 4 Appium Inspector

2.3.3 Q-testing

Q-testing[17]是一个基于强化学习策略的 Android 应用自动化探索的框架，其用马尔可夫决策过程描述对 Android 应用的探索过程，使用 Q-learning 的策略指导探索应用的过程，可以在较短的时间内达到较高的覆盖度。Q-testing 工具调用原生的 UI Automator 框架驱动应用和获取应用状态信息。

Q-testing 使用状态的相似度作为强化学习过程中的奖赏，倾向于探索与已探索状态更不相似的状态，工具使用长短期记忆（LSTM, Long Short-Term Memory）模型抽取状态的特征，并且通过特征向量的曼哈顿距离比较状态的相似度。

3 方法介绍

本章介绍 SMF 方法的整体框架与各模块细节，后续章节将介绍原型工具的具体细节。

3.1 方法概览

方法的整体模块组成与数据流动见图 5，我们基于功能将方法划分为不同的子模块，并且定义了全局统一的数据交换格式，使得工具易于扩展，且可以作为模块整合进更大的平台中。

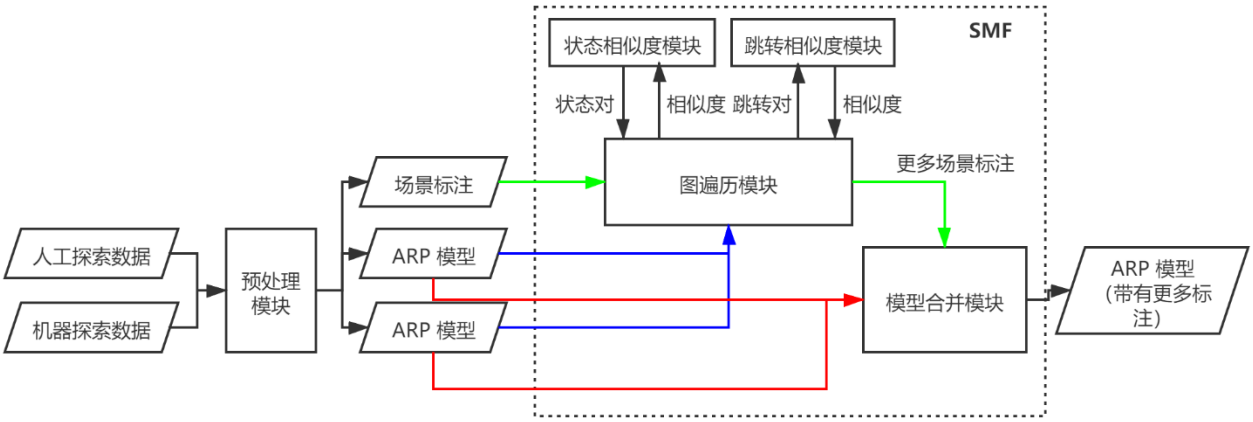

图 5 SMF 框架图

预处理模块将多种不同的数据来源（测试人员探索的模型以及机器探索的模型）处理加工为统一格式的 ARP 模型，包含移动应用的跳转图模型，每个状态对应的截屏文件以及布局文件，便于后续进一步的处理。对于人工探索的模型，预处理模块还会提取出场景路径信息。

模型合并模块接受预处理模块的输出，将两个 ARP 模型合并为一个 ARP 模型，格式不变，对于不同来源的状态会以不同的前缀进行标注区分。模型合并模块还会将图遍历模块得出的场景路径添加到合并后的新模型上。

图遍历模块主要实现标签泛化功能，其依赖两个相似度判断模块，分别是针对状态的状态相似度模块与针对状态间跳转的跳转相似度模块。在计算两种相似

度的前提下，图遍历模块使用一种基于宽度优先搜索（BFS, Breadth-first Search)的方法寻找与给定场景路径所匹配的子图,然后使用深度优先搜索(DFS, Depth-first Search）得到子图中的所有路径，由模型合并模块将场景标注泛化到这些路径上去。

3.2 预处理

为移动应用建模的数据可以有多种来源，为此需要将其进行预处理，变换成统一的格式用于后续的处理。我们将不同来源的模型统一处理变化成上文提到的ARP 模型，具体而言，一个 ARP 模型需要包含以下信息，预处理模块负责从原始的数据来源中提取出这些信息。

- 应用的状态信息
- 应用的跳转关系
- 应用的元数据
- 应用的场景路径信息

应用的状态信息包括了应用运行不同时刻时的截屏信息以及界面的布局信息，预处理模块用唯一的序号标识不同的状态，序号关联了相应的截屏与布局信息。我们研究的对象主要是 Android 应用，在进行状态信息收集时，每个状态还关联了该状态所属 Activity 的信息（Activity 是 Android 的核心组件之一，代表一个有界面的屏幕，承载主要的展示和交互功能)。

跳转关系则是不同状态间的有向联系，预处理模块使用源状态和目的状态的序号对标识一系列跳转关系（两个状态间可能通过不同的方式跳转)，跳转关系与触发跳转的事件相关联。人工探索的模型数据中还会包含场景路径信息，场景路径是带标签的跳转关系集合，预处理模块会将标签与跳转关系关联，同时也会单独将每个场景路径以路径的形式提取出来。

应用的元数据主要是应用的名称，开发者，版本等信息，预处理模块将这些信息也提取出来,便于之后将大量 ARP 模型组织起来时从不同的维度进行搜索查询。

3.3 模型合并

模型的合并主要分为两个步骤，首先是将两个模型的状态集合进行合并；然后在合并后的状态集合的基础之上，将原本模型的跳转模型合并为一个跳转模型图。

上文提到在预处理时，会使用唯一的序号标注状态，在合并状态集合时为了防止序号重复，会在原本的序号前加上不同的前缀来标识不同来源的模型。将状态集合合并后，对于跳转模型，合并的方法是引入一个新的伪状态。伪状态没有对应的截图文件和布局文件，只是作为合并后的 ARP 状态的入口点。合并模型时，在伪状态和原本模型的入口状态间添加跳转关系，然后将原本模型的跳转关系添加到新的模型中去，原本模型的跳转图成为新模型跳转图的子图，这样的好处是后续可以在该模型上以增量的形式合并新的模型并且继续进行标签泛化过程。图 6 是一个合并的例子。

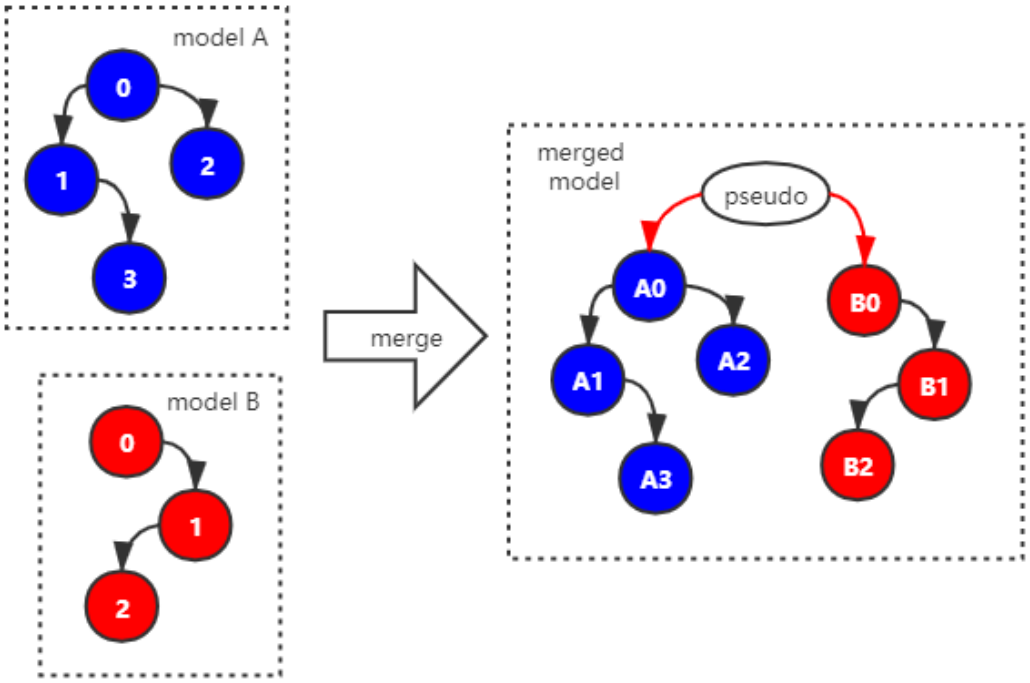

图 6 合并模型示例

3.4 状态相似度判断

在ARP模型中,与应用模型的状态相关的信息有两种:截屏文件与布局文件。在前期调研中，我们探索了基于截屏文件的相似度计算方法，但是移动应用的截图不同于一般的图像数据，其细节较少，内容与颜色无关，且有效信息占整幅图片的比重较小，基于图形的相似度算法对于截屏文件并不能得到很好的效果。因此我们选择基于布局文件计算状态间的相似度。此前已经有不少工作探索了 XML 文档的相似度计算，包括基于语义和结构的 XML 文档相似度计算[18]，基于树编辑距离下界的相似度估计[19]，基于有效路径权重的 XML 匹配算法[20]等。在调研后我们实现了其中的一些算法，并且针对 UI 布局文件的特殊性做出了针对性的调整，使得算法能取得更好的效果。

除此之外，上文中提到的Q-testing工具中通过长短期记忆（LSTM）模型抽取状态的特征，用以计算状态间的相似度，我们也将该部分功能封装为相似度计算模块，与其他相似度计算算法进行比较。

我们统一了相似度模块的接口，使得相似度算法可插拔可配置，不同的方法接受相同形式的输入参数，输出在[0,1]区间内的相似度数值，这样的设计使得不用修改泛化算法的其他部分即可方便地更换相似度算法。同样地，只要封装为统一的接口，也可以轻松地添加新的相似度算法。

3.4.1 基于语义和结构的相似度判断

文献[18]中提出了一种基于语义和结构的 XML 文档相似度判断方法，该算法整体分为三个层级：

- 基于语义相似度与编辑距离计算节点间相似度
- 基于动态规划方法计算路径的相似度
- 将 XML 文档分解成从根节点出发的路径集合，基于路径间的相似度计算路径集合的相似度

在使用该算法计算布局文件的相似度时，我们针对布局文件的特殊性做出了一定修改。首先是节点间相似度，原算法计算的对象是自然语言文本，算法会将

文本分词后计算其语义相似度，同时计算文本的字符编辑距离相似度，取二者的最大值作为节点的相似度。布局文件 XML 树的节点是组件，其本身除了属性并没有其他内容，组件属性中以下几种对相似度判断较为重要：

- text：对于会在界面上显示文本的组件，该字段即为显示的文本内容
- content-desc：描述组件的功能，在使用“辅助使用”功能时，该字段的内容会被屏幕朗读使用
- resource-id：唯一标识组件，主要用于开发工作时对组件的定位
- class：组件的类型，如果是派生自 SDK 提供的组件类型，在此处会显示基类的类型
- package：组件所属的包

计算节点相似度时，首先判断两个节点的 class 属性与 package 属性是否相同，如果不同则认为两个组件相似度为 0，如果相同则将 text，resource-id，content-desc 三个属性的值拼接为一个字符串，使用字符编辑距离相似度作为节点的相似度。

在计算路径相似度时，原算法提出了一种基于最大相似子序列（MSS, Maximal Similar Subsequence）的路径相似度定义。算法提出 XML 文档中距离根节点更加近的节点往往更能反映文档的结构信息，因此在定义路径相似度时加入了基于深度的衰减因子，然而对于布局文件来说，距离根节点更近的组件通常是容器而非具体的控件，对于两个状态布局来说，控件相比容器往往含有更多的信息，因此在实现路径相似度算法时，我们去掉了这个衰减因子，使用最大相似子序列作为路径相似度，具体实现采用动态规划的方法。

对于 XML 文档得到的路径集合的相似度，我们直接使用了原算法的方法，并没有进行额外的修改，最终输出的相似度被归一化至[0,1]区间内。

3.4.2 基于树编辑距离下界的相似度判断

树编辑距离（Tree Edit Distance）是一种衡量树结构之间相似度的参数，与字符编辑距离类似，其定义为将一颗树通过插入/删除/替换转换为另一颗树所需要的最小操作次数。显然树编辑距离越小，两棵树越相似，然而现在并没有高

效地计算树编辑距离的算法。文献[19]提出可以通过多种方法估计树编辑距离的下界，通过取这些方法得出的编辑距离的最大值，即可较好地逼近真实的树编辑距离。

文献中首先提出可以通过将树转换为字符序列，然后以字符序列的编辑距离作为树编辑距离的下界。由于树编辑距离是以节点为单位进行操作，因此在将树转换为字符序列时，每个节点需要对应一个字符。节点中对相似度判断较为重要的属性在上文中已经介绍过，其中 text，resource-id，content-desc 属性并不是所有节点都有非空的值，package 属性取值范围较小，而 class 属性既能较好地反映布局的结构，又具有有限的取值范围，因此我们选择将每个节点的 class 属性映射为一个字符，得到一颗部分程度上反映了布局组件嵌套结构的树，对这棵树分别做前序遍历与后序遍历，得到的字符序列可以用于计算字符编辑距离，字符编辑距离的最大值即为树编辑距离的一个下界。

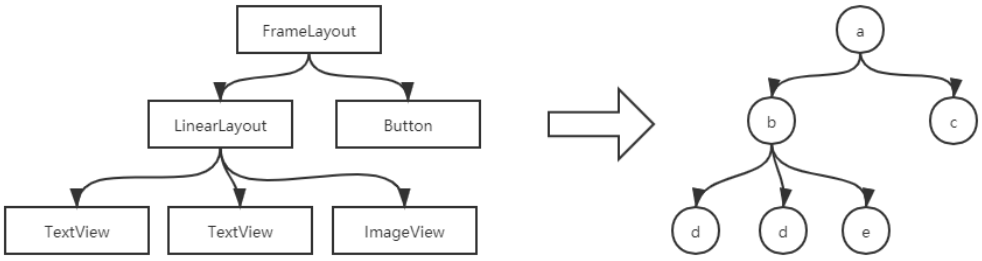

图 7 映射过程示例

文献还提出三种基于直方图（Histogram）的估算下界的方法，其基本思路是将树转换为向量，然后通过向量的 L1 距离估算树的编辑距离，文献中提到了三类直方图：叶距离直方图（Leaf Distance Histogram），度直方图（Degree Histogram）与标签直方图（Label Histogram）。

一个节点的叶距离定义为以该节点为根的子树中，从根到任意叶节点的最大距离，也就是以该节点为根的子树的高度，则如果树高为 h，叶距离直方图就是一个长度为 h+1 的向量，其第 i 个分量的值为树中叶距离等于 i 的节点个数。

度直方图的定义类似，如果树中最大的度为 d，则度直方图是一个长度为 d+1

的向量，其第 i 个分量的值为树中度等于 i 的节点个数。

标签直方图统计的是树中节点标签出现的频次，其向量每个分量的值代表了某个特定标签对应的节点在树中出现的频次。在前期调研中我们通过小规模的实验发现，如果使用 class 属性作为节点的标签，标签直方图得出的向量并不能很好地反映布局的特性，因为标签直方图只考虑了组件类型的频次，不能反映出组件之间的嵌套关系以及整个布局的结构，而相同 app 的不同界面其组件类型的频次分布并没有非常大的差异，故在实现相似度算法时，并没有采用该方法。

叶距离直方图与度直方图均只考虑了树的结构信息，文献还提出一种基于二叉距离(Binary Branch Distance)的估算方法，兼顾了树结构信息与内容信息。该方法需要将普通的树先转换为二叉树，这里我们使用了一种传统的转换算法，即对于任一的一个节点，如果其有子节点，则将第一个子节点作为左子树的根，如果其有兄弟节点，则将第一个兄弟节点作为右子树的根，从根开始递归进行，则可以将树转换为二叉树。转换后的二叉树每个节点及其左右子节点的标签都可以转换为一个长度为 3 的前序遍历序列（如果某个子节点为空，则用特殊的符号表示空即可），统计该序列的频次，同样可以将树转换为一个向量，然后可以用向量的 L1 距离估算树编辑距离。

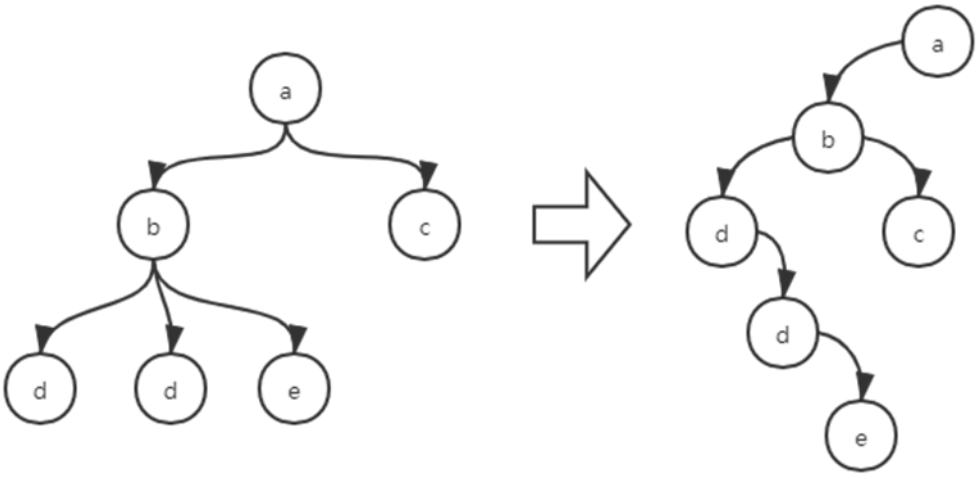

图 8 树转换为二叉树

在实现时，我们选择基于上文提到的节点映射为单字符后的树进行变换，如

图 8 所示，变换后的二叉树每个节点分别可以转换为标签：ab*，bdc，d*d，c**，d*e，e**。

文献中共提到了 5 种估算树编辑距离下界的方法，我们实现了其中的 4 种，并且对于每种方法，针对布局结构 XML 树的特殊性做出了修改。不同方法得出的树编辑距离下界取最大值则为估算的树编辑距离，将其归一化至[0,1]区间，用 1 减去归一化后的距离则得到相似度。

3.4.3 基于 LSTM 的相似度判断

上文中提到了基于强化学习的自动化测试工具 Q-testing，在该工具中使用状态的相似度作为强化学习策略的奖赏，驱动工具尽可能探索更多不同的状态。Q-testing 使用 LSTM 模型抽取状态的特征，对于状态的布局树，其遍历树中的每个节点，将组件的属性编码为长度 214 的向量，整棵树会被编码成 214*100 的矩阵（组件数不够的用 0 做填充）作为 LSTM 的输入，输出一个长为 100 的特征向量，计算向量的 L1 距离可以得到两个状态的相似度。

我们使用同样的规则对状态布局树进行编码，并且使用了已经训练好的 LSTM 模型，然后将该部分模块封装为统一的接口，将向量的 L1 距离归一化至[0,1]区间，用 1 减去归一化后的距离得到相似度。

3.5 跳转相似度判断

在 ARP 模型中，除了状态本身附带的信息，触发状态间跳转动作的信息也是十分重要的。在进行标签泛化的过程时，仅仅考虑状态间的相似程度是不够完善的，一条场景功能路径同样需要考虑到状态间跳转动作的匹配程度。本文提出了一种较为简单直观的跳转相似度判断算法，通过对比跳转动作的交互区域进行相似度判断。

在进行移动应用测试时，交互动作可以分为两类，一类是具有具体交互区域的，如点击，长按，编辑文字等，一类是没有具体交互区域的，如滑动，返回等。如下是一些跳转信息的示例。

表 1 跳转信息示例

```
"action": [
  "8@menu",
  "7@click(content-desc='More options',bounds=\"[932,66][1080,192]\")"
]
```

对于有交互区域的动作，模型会记录下两个坐标，分别代表交互区域的左上端点和右下端点。在比较两个跳转动作时，算法会首先尝试通过正则表达式匹配的方式提取交互区域坐标，如果两个动作都是有交互区域的，则比较其交互区域的重叠程度，算法会首先根据截屏文件对应的分辨率将两个坐标归一化到[0,1]区间，然后计算两个区域的 Jaccard Index，即两个区域相交的面积除以两个区域相并的面积，这是一个常用的集合相似度判断方法。Jaccard Index 会输出一个[0,1]区间的结果，越接近 1 说明两个区域重合程度越高。

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

在实际实验时，我们发现在实验数据中会有两个交互区域包含的情况，如果按照公式计算并不会得到很高的相似度分数，但是在观察原始数据后，我们发现这往往是同一个交互动作，交互区域产生区别的原因是不同来源的建模数据会将同样的操作定位到不同控件，一个常见的例子是点击一个列表项，有些记录会定位到外层的 LinearLayout，而有些记录会定位到内层的 TextView。

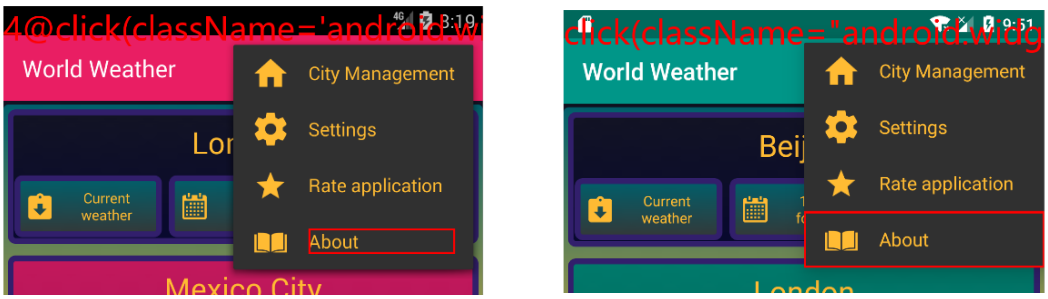

图 9 交互区域示例

为了处理这种情况，我们额外添加了规则，如果两个区域成包含关系，则认为其相似度为1.0。这也比较符合设计移动应用的原则与使用时的经验，如果两个交互区域包含，其通常可以触发相同的交互动作，一个比较典型的例子是大部分App的设置界面有可开关的选项，点击Switch控件/文字标签或是点击该选项的容器部分都能切换功能的开关。

对于没有交互区域的动作，这些动作除了滑动外基本都是系统事件，如Android的三个虚拟按键：menu，back，recent apps，对于这些动作，我们选择采用正则表达式将动作类型提取出来，然后比较其是否相同，如果相同则认为动作相似度为1.0，否则为0.0。

算法 1 跳转动作相似度算法

```
Algorithm 1: 跳转动作相似度算法
Input: 两个待比较的动作
Output: 动作的相似度
1  if 两个动作都有交互区域 then
2    计算两个动作交互区域的 Jaccard Index
3    if 两个交互区域成包含关系 then
4      return 1.0
5    else
6      return Jaccard Index
7    end
8  else if 两个动作均有交互事件 then
9    if 交互事件相同 then
10     return 1.0
11   else
12     return 0.0
13   end
14 else
15   return 0.0
16 end
```

由于跳转模型中一对状态间可能有多个跳转动作，在实际计算时，会对两个跳转动作列表中的每对跳转动作分别计算相似度，结果取所有相似度的最大值。

3.6 图遍历

本文实现了一种基于宽度优先搜索（BFS, Breadth-First Search）的图遍历算法，用于将场景功能标注泛化到 ARP 模型的跳转模型中。由于人工探索的模型已经有了场景标注，且一般不会出现重复的场景功能路径，故本算法仅在无场景标注的机器模型上运行，在没有额外说明的情况下，下文中提到的跳转模型均指无标注的机器探索的跳转模型。

算法分为两个部分，遍历算法主体以及一个 Wrapper。遍历算法接受一个图中的结点作为输入，输出一个以该节点为根的有向无环图，该图中的路径即是算法判断与场景路径相匹配的路径，算法的伪代码如下：

算法 2 标签泛化算法

Algorithm 2: 标签泛化算法
Input: 起始节点，跳转模型，场景路径
Output: 匹配的子图

```
初始化队列 q，将起始节点放入 q;
初始化结果图 G;
初始化已访问节点集合 visited;
for i ∈ [1, len(path)] do
  初始化队列 temp;
  visited ← visited ∪ q;
  foreach node ∈ q do
    将 node 加入 G;
    获取场景路径从第 i-1 个状态到第 i 个状态的跳转动作;
    根据跳转动作相似度与状态相似度，获取下一跳的节点列表 next_nodes;
    next_nodes ← next_nodes \ visited;
    foreach next_node ∈ next_nodes do
      在 G 中加入边 (node, next_node);
      将 next_node 放入 temp;
    end
  end
q ← temp;
end
return G;
```

算法使用 BFS 作为框架，每次外层循环会基于前一次遍历的结果向外推进一

跳，最多推进 n 步（n 为给定场景路径的长度），算法使用一个队列存储当前待处理的节点列表，并且在获取下一跳的节点列表时会筛去已经访问过的节点，以确保结果图是一个有向无环图。算法较为核心的部分是第 10 行，即如何获取下一跳节点列表的部分，该部分使用了上文提到的状态相似度模块与跳转相似度模块，其伪代码如下：

算法 3 获取下一跳节点列表

Algorithm 3: 获取下一跳节点列表
Input: 当前节点，跳转模型，源跳转动作，源节点，状态相似度算法，相似度阈值
Output: 下一跳节点列表

```
1  初始化列表 next_nodes;
2  根据跳转模型，获取当前节点的所有后继结点 succs;
3  foreach succ ∈ succs do
4    根据跳转模型，获取从当前节点到 succ 的跳转动作;
5    计算源跳转动作与目的跳转动作的相似度 action_sim; /* 算法 1 */
6    计算源节点与 succ 的状态相似度 state_sim;
7    similarity ← action_sim + state_sim;
8    if similarity ⩾ 相似度阈值 then
9      将 succ 加入 next_nodes;
10   end
11 end
12 return next_nodes;
```

在该算法中将状态相似度算法作为参数在调用时传入，针对不同的相似度算法，我们给出了不同的相似度阈值，这一部分将在后续的实验部分详细说明。

遍历得到结果图后，会从根节点出发递归进行深度优先搜索（DFS, Depth-First Search），由于结果图是一个有向无环图，这个递归过程一定能够结束。算法使用一个栈驱动 DFS，这个栈同样保存了从根节点出发到当前节点的路径，当遍历到没有后继的节点时（即标签泛化算法推进到最后一步时加入图的节点），便认为得出了一条完整的路径，会将此时的栈内容保存下来，最终输出一个路径的集合。

除了遍历结束后的 DFS，遍历前也需要确定起始节点，Wrapper 的功能就是在跳转图中选择起始节点，并且对每个起始节点都运行一次泛化算法，然后对得

到的结果图进行遍历得出路径集合，最后将所有路径的集合合并后输出结果。

算法 4 标签泛化 wrapper

Algorithm 4: 标签泛化 Wrapper
Input: 跳转模型，场景路径
Output: 路径集合

```
1  初始化路径集合 paths;
2  在跳转模型中计算每个节点与场景路径第一个节点的相似度，取相似度从高到低前
3  10 个节点作为 candidates;
4  foreach candidate ∈ candidates do
5     以 candidate 为起始节点，运行标签泛化算法;
6     对得到的结果图，进行 DFS 得到路径集合 temp;
7     筛去 temp 中长度小于 1 的路径（即只有一个节点的路径）;
8     paths ← paths ∪ temp;
9  end
10 return paths;
```

由于场景路径标注是基于跳转路径的，因此会筛去结果中没有形成路径的输出。在得到路径集合后，会将场景标注根据输出结果添加到合并后的跳转模型对应的路径中。

4 工具实现与实例研究

本章主要介绍基于上文提出 SMF 方法的原型工具的具体实现，下文将详细介绍各个模块的实现细节。我们选择基于 Python 语言实现该工具，具体实现形式为可执行的脚本文件。工具的模块划分与数据流动与上述方法模块图类似，见图 5。

工具实现了上述的三个不同的状态相似度算法，并且将该部分设计为可配置可插拔的形式，可以通过修改配置文件或者在调用脚本文件时输入不同命令行参数的方式更换不同的状态相似度模块。

4.1 预处理模块

为移动应用建模的数据可以有多种来源，为此需要将其进行预处理，变换成统一的 ARP 模型用于后续的处理。上文提到的 ARP 模型是一个概念模型，在具体实现时，各个状态附带的信息（截屏文件，布局文件）可以以文件的形式通过文件系统组织，使用时只需以读取文件的形式将其内容读入内存即可。对于模型的跳转关系，其内存形式我们选择使用 Python 的第三方库 NetworkX[21]实现，而持久化存储则选择 JSON[22]格式的文件。

NetworkX 是一个用于创建，操作，存储图结构和网络结构的库，实现了多种有向图/无向图类型以及常用的图算法。NetworkX 使用一种“字典的字典”的形式组织图数据，整个图结构是一个字典，其键是图中的节点，而其值是另一个字典，该字典以{目的节点：边数据}的形式存储所有从键代表的节点出发的边，以下 Python 代码描述了一个有两个节点的有向图，图中有一条从状态 1 指向状态 2 的边，其边有两个属性：weight 和 time_sequence。

表 2 NetworkX 代码示例

```
import networkx as nx

graph = {
    "1": {
        "2": {"weight" = 1, "time_sequence" = [1,2,3]}
    },
    "2":{}
}

G= nx.DiGraph(graph)
```

JSON（JavaScrpit Object Notation）是一个轻量级的数据交换格式，有较好的可读写性，且由于定义简单，同样易于机器解析与生成。JSON 是一种语言无关的文本格式，其由两个基本结构组成：对象（Object）是一系列键值对的组合，数组（Array）是一系列值的有序组合。而 JSON 的值（Value）可以是字符串，数字，对象，数组，true，false，null。Python 自带读写 JSON 格式字符串的库，可以在 JSON 对象与对应的 Python 数据类型之间进行转换。使用 JSON 可以轻松地表示上述的图结构。

表 3 JSON 代码示例

```
{
  "1": {
    "2": {
      "weight" = 1,
      "time_sequence" = [1,2,3]
    }
  },
  "2":{}
}
```

在预处理时，我们将应用的状态使用唯一的序号进行标注，使用序号命名应用对应的截图文件与布局文件，然后将跳转信息转换成上述的 JSON 格式，状态的序号作为图的节点，跳转关系作为图的边，而跳转动作作为边的附加信息，如果一对状态间有多个跳转，则不同的跳转动作组织成一个列表。除此之外，我们

还在预处理时筛去了跳转模型中的自环。如下是一段预处理后的跳转模型的例子。对于应用元数据和场景路径等信息，也同样处理为 JSON 格式的文件进行保存。

表 4 预处理后模型示例

```
"2": {
  "0": {
    "action": [
      "5@back",
      "3@click(className='android.widget.TextView',instance='3',bounds=\"[
101,1258][290,1384]\")"
    ]
  },
},
```

工具运行时会将 JSON 格式的跳转模型读入内存，转换成 Python 中的字典格式，然后创建为 NetworkX 中的有向图对象。

我们将预处理模块实现为一个单独的脚本文件，调用后可以将原始数据转换并输出为预处理后的 ARP 模型，后续的工具主体将基于该输出结果进行处理，这样设计的灵感来自于编译器前后端分离的思路，预处理模块要处理多种不同来源的模型数据，需要针对不同的情况进行修改或扩展，这种特性决定了预处理模块和后续的合并主体不能有过高的耦合度。解耦后可以将多种不同的预处理模块与合并工具进行组合，使用时只需链式调用即可。

4.2 模型合并模块

在实现模型合并时，我们同样使用上述 NetworkX 库作为操作模型的工具。需要合并的内容是应用的状态和跳转关系，工具会初始化一个新的有向图对象，在加入初始的伪起始状态后，将两个模型的跳转模型的边分别加入新的模型。为了避免状态标识符重复，我们采用前缀+原始状态序号作为新状态的序号，然后在伪起始状态到每个原始模型的起始状态间添加空的跳转关系。

合并模块在完成合并后会调用图遍历模块，将模型信息和场景路径信息作为输入，模块会输出泛化后的场景路径，合并模块随后会将场景路径信息添加到合

并后的模型上去。如下是一个合并后的 ARP 模型的跳转模型片段示例：

表 5 跳转模型示例

```
"MS#1": {
  "MS#2": {
  "action": [
    "click(className=\"android.widget.TextView\",instance=\"-
1\",bounds=\"[597,129][1028,186]\")"
  ],
  "tag": [
    "设置: 设置锁屏"
  ]
},
```

合并模块的脚本文件是工具的主体，其接受同一应用的两个不同版本的模型作为输入，在合并的过程中会调用图遍历模块提供的接口获取泛化的场景路径信息。这里也选择将合并框架与图遍历模块解耦，便于未来对工具进行扩展，如在合并过程中添加泛化场景标签以外的功能等。

4.3 图遍历模块

4.3.1 相似度模块

工具将相似度模块实现为不同的函数，供图遍历模块调用。上文提到了多种不同的状态相似度判断算法，我们将这些算法使用 Python 实现。在具体实现时，不同的相似度算法有着不同的输入形式与输入数据，为了能够实现多种相似度算法的可插拔与可配置，我们将相似度算法封装为统一的接口，其接受同样的输入：应用名称以及待比较的两个状态的版本及其唯一标识符。不同的相似度算法输出的相似度形式不同，对于本身输出[0,1]区间内的相似度算法，我们直接采用其输出，而如果输出不在该区间内，如基于树编辑距离的相似度算法，其输出结果是两个状态 XML 树的编辑距离下界，我们在实验数据上进行了测试，根据其大致分布将其归一化到[0,1]区间后返回。

Python 有着函数式编程的特性，函数可以像一般的对象一样作为程序中的参数与返回值进行传递。我们在实现图遍历模块时，将状态相似度算法函数作为参数传入，由于之前统一了相似度模块的函数签名，故可以替换不同的相似度算法而不用修改依赖其的图遍历模块。我们将不同函数对象组织为一个相似度函数表，在调用脚本工具时可以根据命令行参数的不同选择不同的状态相似度算法进行计算。如下是命令行工具的帮助界面，可以指定采用的相似度算法与对应的相似度阈值。

```
usage: merge.py [-h] {0,1,2,3,4} {0,1,2} output {lstm,ted,doc} threshold

positional arguments:
  {0,1,2,3,4}     app index
  {0,1,2}         model version index
  output          output dir
  {lstm,ted,doc}  xml similarity
  threshold       similarity threshold

optional arguments:
  -h, --help      show this help message and exit
```

图 10 命令行工具调用示例

在实现相似度计算模块时，有一些算法运行时间较长，为了提高实验时的效率，我们在模块中实现了一个快查表，可以根据状态对查询其相似度，对于不在表中的状态对则将其加入表中，加快了实验进程。后续实验中只有在测试不同算法运行时间时关闭了该快查表。

4.3.2 图遍历模块

图遍历模块主要进行场景功能标签的泛化过程，我们实现了前文所述的遍历算法，这部分模块同样是接受 ARP 模型作为输入，同时也需要输入对应的场景路径信息。在这里我们同样使用了 JSON 文件来组织场景路径数据。图遍历模块会将泛化后的路径集合输出。

我们同样将图遍历模块实现为一个单独的脚本文件，这样可以有两种使用方式：在测试泛化算法效果或调试代码时可以不用完全将模型合并，只是调用脚本文件观察输出的路径信息；合并工具使用该模块时可以直接调用其提供的 API 获

取内存中的路径集合信息。这样做的好处是可以将图遍历模块独立出来，一方面使得工具划分更为清晰，一方面也可以在未来将该模块用于其他工具。

4.4 实例研究

接下来，我们将通过展示一个具体的模型合并的例子，来演示 SMF 工具的运行原理和过程。我们选择开源 Android 应用 World Weather 作为研究对象，为该应用建模的测试人员是一个有着一定移动应用开发经验和使用经验的硕士研究生。工具首先将该模型数据预处理为前文提到的 ARP 模型。

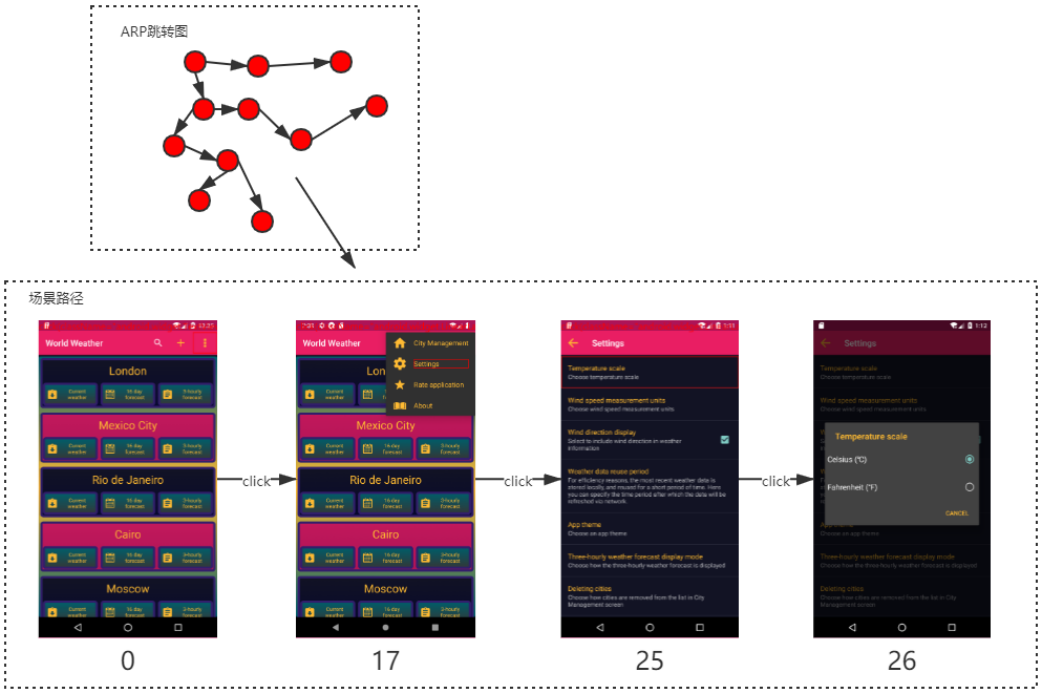

图 11 ARP 模型中的场景路径

在该模型中，我们选取了测试人员标注的一条场景路径：0-17-25-26，这条路径代表的功能场景是调整应用显示温度的单位（摄氏度/华氏度）。接下来，工具会同样地将机器遍历得到的模型预处理为 ARP 模型，然后根据算法 4 得出遍历的起始节点。

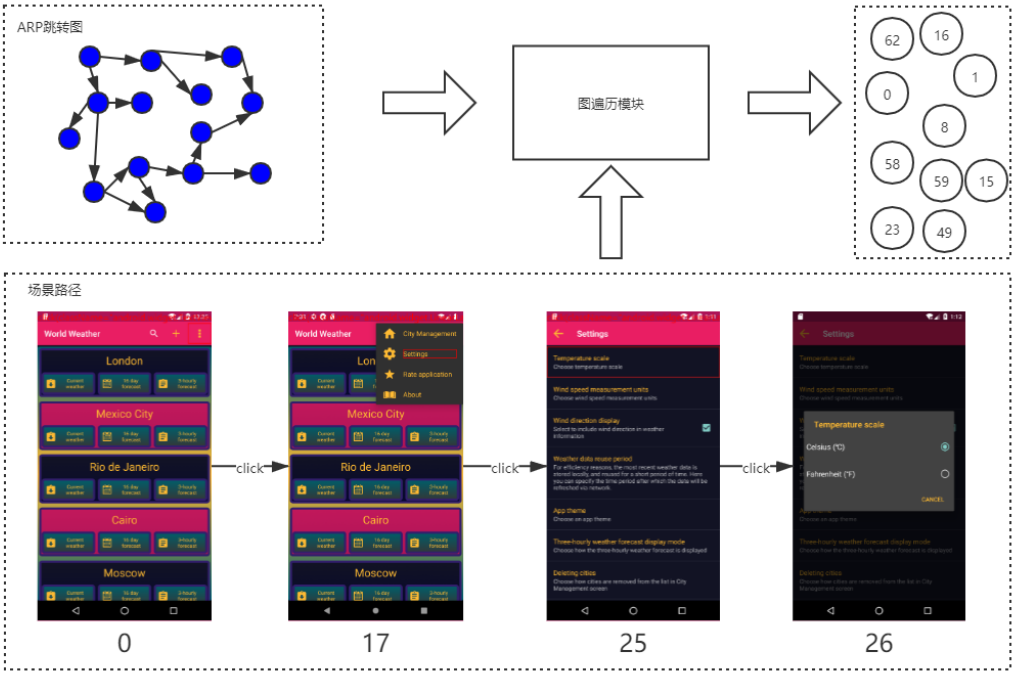

图 12 获取遍历起始节点

在获取了起始节点列表[62, 16, 0, 1, 8, 58, 59, 15, 23, 49]后，工具会对列表中每个节点依次执行算法 2，这里我们以节点 8 作为例子，继续演示工具的运行流程。以节点 8 作为起始节点，工具会以宽度优先搜索的形式开始遍历，由于给定场景路径的长度为 4，故该遍历会向外推进 3 次。遍历的过程中对于下一跳节点的选择使用了相似度计算模块的功能，具体细节可以参考算法 1 与算法 3。为了避免遍历结果图产生环，我们在遍历过程中记录了已访问的节点集合。

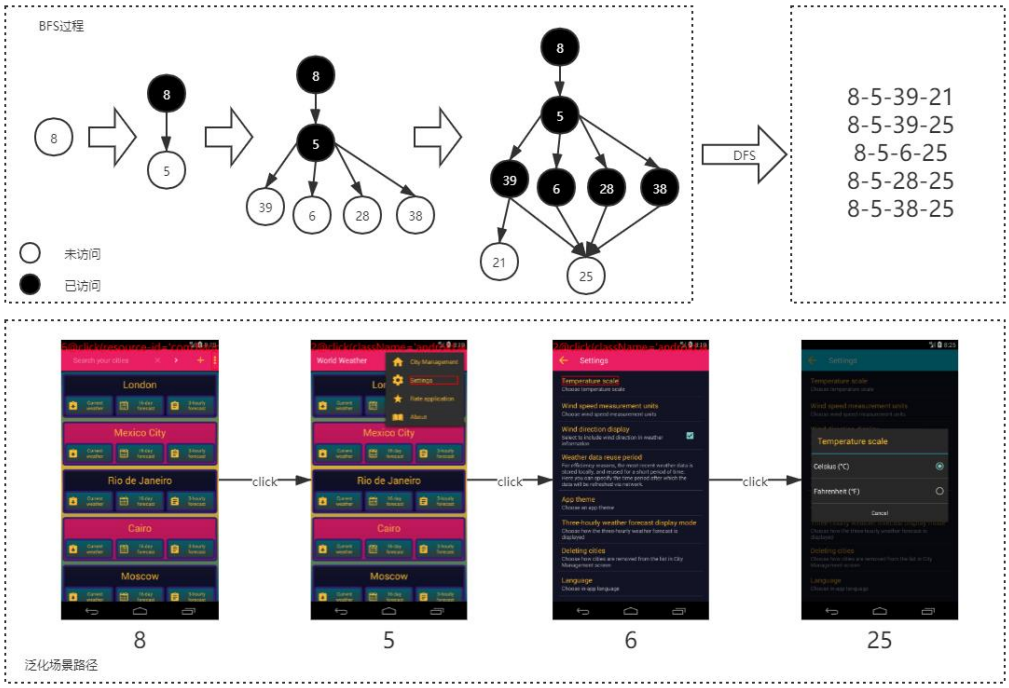

图 13 图遍历过程

如图 13 所示，工具对每个起始节点都会进行遍历过程，得到机器跳转模型的一个子图，然后在子图上进行深度优先搜索得到路径集合。这里我们也挑选出一条路径：8-5-6-25，展示其界面截屏与跳转动作。每个节点在遍历后都会得到一个对应的路径集合，工具会筛去其中仅有一个状态的路径，然后将路径集合合并至一个集合，输出结果会传递给模型合并模块，以在合并过程中进行标签的泛化。

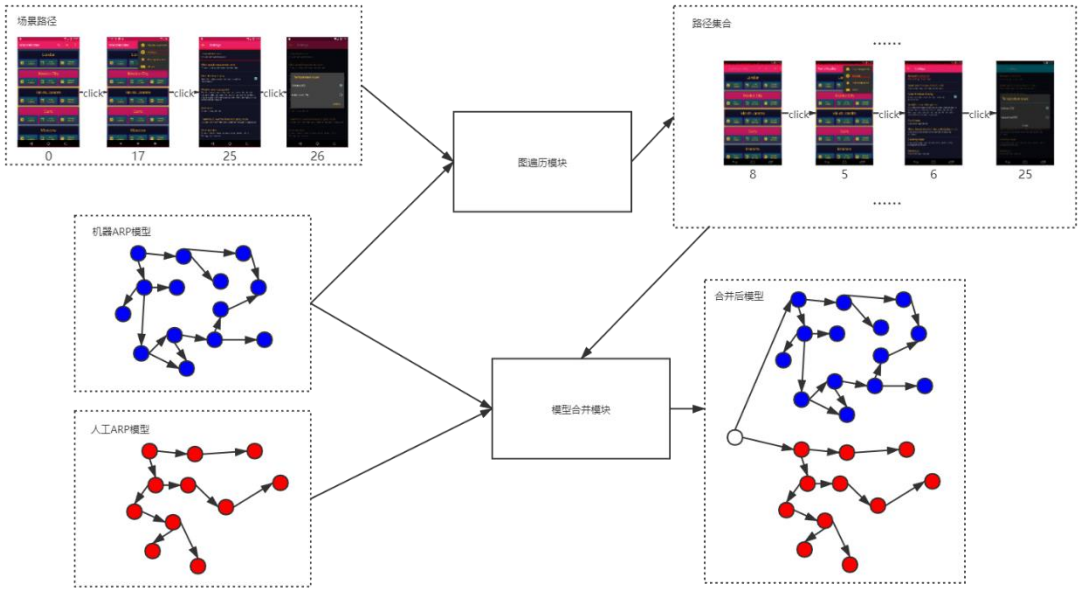

图 14 模型合并过程

模型合并模块读取之前预处理得到的两个 ARP 模型，同时也读取了图遍历模块得出的场景路径集合，然后将两个 ARP 模型合并，并且将场景路径标签泛化到合并后的模型上去。在实际运行时一个人工模型会有不止一条标注过的场景路径，我们在预处理后会对每条路径进行上述的泛化过程，并且用源路径的功能标签和功能描述来标注得出的路径集合，最后再进行模型的合并，合并过程中将得出的所有结果路径一次性泛化到合并的模型上。

4.5 总结

我们使用 Python 语言实现了 SMF 工具，具体体现为一系列可执行脚本文件，其中主要有三个模块：预处理模块，模型合并模块，图遍历模块。这三个模块都可以单独调用实现其功能，也可以组合起来实现本文所述的模型融合功能。我们实现了状态相似度模块与跳转相似度模块，图遍历模块依赖于这两个模块，且状态相似度模块实现了可插拔可配置，可以在不修改源代码的情况下通过命令行参数或配置文件的方式更换工具使用的相似度算法，增加工具的灵活性。

我们也进行了实例研究，以具体的模型的合并以及场景路径的泛化展示了工具整体的运行过程，以及各部分的运行细节。

除此之外，我们还基于现有的模块实现了一些辅助的脚本工具，如基于状态相似度模块实现的脚本工具，可以输出与指定状态相似的状态并按照相似度排序，或是基于预处理模块的脚本工具，可以将场景路径进行一定程度的可视化，便于观察与分析模型数据。

这些模块集合组成了 SMF 工具，我们将在接下来的章节对该工具进行测试，观察其对移动应用模型的融合效果。

5 实验评估

在前面的章节中，我们介绍了一种面向场景的模型融合算法 SMF，并且介绍了其具体的工具实现，包括其中较为重要的模型合并模块与图遍历模块。在本章中，我们设计了实验用于测试 SMF 工具的标签泛化效果，并且通过配置不同的相似度判断算法，测试不同的算法在时间、泛化效果等方面的表现差异。

5.1 实验目标

我们设计的实验主要关注两方面的内容：

- SMF 工具泛化标签的效果
- 不同的相似度算法的表现差异

相应地，我们提出了两个研究问题

RQ1：面向场景的模型合并：SMF 是否能通过合并的方式，泛化得到拥有更多场景路径信息标注的模型？

RQ2： 相似度算法差异：不同的相似度算法在运行时间，泛化效果等方面有怎样的差异？

接下来我们将围绕这两个研究问题，设计相应的实验，并且对实验得到的结果进行分析。

5.2 实验设计

我们选择开源的 Android 应用作为实验的对象，一共收集了 5 个不同类型的 App 的模型数据，每个 App 有三个版本的模型，其中有两个版本由测试人员手动生成，分别由具有一定移动应用使用经验与开发经验的硕士生和本科生收集，我们将这两个版本记为 MS 与 UG。测试人员使用 Appium 作为驱动测试的框架，使用 UI Automator viewer 获取 App 的截屏文件与布局文件，并且根据布局信息编写相应的 Python 测试脚本。我们基于截屏文件与布局文件作为模型的状态，基于测试脚本的定位和调用信息作为状态间的跳转。测试人员在测试的同时也给出了相应的场景标注信息，标注信息由功能标签，功能描述以及对应的场景路径组

成。

表 6 场景路径示例

1	[激活][进入主页面][38-39-40-45]
2	[设置][设置锁屏][1-2-3-4-5-1]

除此之外，每个 App 还有一个无标注的机器探索的模型，该模型由上文提到的 Q-testing 工具探索生成，我们将这个版本记为 QT。工具使用 UI Automator 作为驱动测试的框架，使用 adb 调用 uiautomator 命令行工具获取运行时 App 的截屏文件与布局文件，工具记录下了跳转时执行的不同交互动作与交互对象。

我们将上述实验数据进行预处理，得到了 15 个不同的 ARP 模型。在 10 个人工生成的模型中，我们筛选出了 32 条功能各异的场景路径，路径长度从 2 个状态到 6 个状态不等。实验时我们选择对每个人工模型，将其与对应的机器生成的模型合并，并且泛化该人工模型附带的场景路径标注信息，一共得到 10 个合并后的 ARP 模型。由于我们实现了 3 种不同的状态相似度判断算法，实验一共进行了三组，用以对比不同算法的表现差异。

5.3 实验数据及分析

为了能够定量地分析工具的标签泛化的结果，我们定义了一套标签泛化的评估规则。对于一条有 n 个状态的场景路径：

- 如果其泛化的某条路径有 m 个状态与该场景匹配，则该路径的得分为 m/n（如果完全匹配，该路径得分为 1.0）
- 该场景路径泛化的所有路径得分的平均即为该场景路径的得分（如果没有输出泛化路径则得分为 0.0）
- 所有 32 条场景路径得分的平均即为该算法的得分

具体实验中，除了上述匹配分数，我们还统计了泛化的路径总数，差路径（泛化路径匹配的状态数小于等于 1）以及优路径（泛化路径完全匹配）占路径总数的比例。不仅如此，我们还统计了采用不同相似度算法时合并工具的运行时间，

这一部分实验在安装了 Windows 10 的笔记本电脑（Intel Core i5 2.5GHz）上进行。

在统计匹配分数时，我们采用人工验证的方法去统计输出路径的匹配程度，为了加快评估的速度，我们开发了脚本工具用于辅助评估，具体而言工具会读取一个快查表，查询对应路径的匹配分数，如果没有分数则由人工进行评估并添加进快查表，这样下一次遇到同样的路径结果就可以直接得出分数。上文中提到了不同的相似度算法需要配置不同的相似度阈值，我们进行了多轮实验，最终为每个算法确定了恰当的阈值。实验数据如下表所示：

表 7 实验数据

相似度算法	相似度阈值	匹配得分	路径总数	优路径比	差路径比	运行时间/秒
基于语义和结构	1.15	0.4732	258	0.3333	0.2248	5143
基于树编辑距离下界	1.3	0.6516	400	0.4275	0.0700	43
基于 LSTM	1.15	0.8043	268	0.4701	0.0560	898

RQ1：面向场景的模型合并。以基于 LSTM 的相似度算法得出的数据为例，按照我们定义的评估标准，算法的匹配得分超过了 0.8，且泛化的路径数有 268 条，远超原本人工模型作为输入的 32 条路径。在泛化的路径中，超过四成是完全匹配，而匹配程度过低的路径占比不超过 6%。

总的来说，我们开发的 SMF 有较好的效果，能够将现有模型的场景路径标记通过合并的方式泛化到规模更大的无标注模型上去，得到含有更多场景标注信息的模型。

RQ2： 相似度算法差异。实验中我们更换不同的相似度比较算法，进行了三组实验用于对比其效果差异，不难看出基于树编辑距离下界的方法和基于 LSTM 的方法效果较好。

三种算法中基于 LSTM 的相似度算法匹配得分最高，且有着最高的优路径比和最低的差路径比，其运行时间中等，融合 10 组模型用时约 15 分钟；基于树编

辑距离下界的方法匹配得分稍低，但是匹配了最多的路径，并且有着接近 LSTM 方法的优路径比与差路径比，不仅如此，基于树编辑距离下界的方法运行速度最快，仅为基于 LSTM 方法的 1/20；基于语义和结构的相似度算法效果最差，且运行时间最长（运行时间超过一小时），这可能是因为该方法主要关注从根至叶的路径的相似度，且需要对两个状态间所有的路径交叉进行比较。对于布局文件而言，单独的从根到叶的路径并不能很好地反映出布局的结构，反而是同层级或相近层级的兄弟节点对布局的影响更大。这也为未来添加新的算法时判断算法与移动应用状态相似度的适性提供了灵感。

在控制了泛化算法与跳转相似度算法不变的情况下，我们改变了状态相似度算法，比较了不同算法在各个维度的表现差异，这同样也体现了 SMF 的灵活性，在未来可以实现更多的针对不同场景的算法，根据需要进行配置使用。

6 总结与展望

本文提出了一种面向场景的模型合并算法 SMF（Scenario-oriented Model Fusion），并且开发出了对应的工具，实现将测试人员手工探索的模型与自动化工具探索的模型合并，并且将场景标注信息泛化到合并后的模型上。SMF 包含两个相似度判断模块：跳转相似度与状态相似度，其中状态相似度模块实现了可配置可插拔，提高了可扩展性。工具共实现了三种不同的相似度模块并对其设计实验进行比较。

算法还有许多可改进的部分，如跳转相似度模块判断策略是基于交互区域，有时候测试人员与自动化工具使用的不是同一个版本的应用，有些相同功能的跳转在不同的版本中控件位置发生了改变，我们的跳转相似度算法还不能处理这种情况。状态相似度算法主要是基于布局结构的相似度，而有些状态布局结构差异较大，但仍然是相似甚至相同的状态，如一个文件管理应用，在展示目录文件时其中文件的数量与文件名等信息在不同情况下会有较大差异，但这些状态在语义上来说是相似的，我们也需要更加准确的相似度判断算法处理这种情况。

工具也仍有可改进的部分，如可以将跳转相似度模块也配置成可插拔的形式，可以更换多种跳转相似度算法，组合不同的跳转相似度算法与状态相似度算法进行遍历。SMF 也需要一个更加用户友好的前端，目前使用的是脚本文件操控工具，输出结果主要体现为 JSON 文件，可以在后续的研究中为其开发 web 前端，以及将输出结果可视化等。

参考文献

[1] GOOGLE. UI/Application Exerciser Monkey | Android Developers [EB/OL].: https://developer.android.com/studio/test/monkey, 2021.
[2] GOOGLE. UI Automator | Android Developers [EB/OL].: https://developer.android.com/training/testing/ui-automator, 2021.
[3] DEKA B, HUANG Z, FRANZEN C, et al. Rico: A mobile app dataset for building data-driven design applications[C]//Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology.2017:845-854.
[4] DEKA B, HUANG Z, KUMAR R. ERICA: Interaction mining mobile apps[C]//Proceedings of the 29th Annual Symposium on User Interface Software and Technology.2016:767-776.
[5] FU K, WANG S, ZHAO H, et al. A Recommendation Algorithm for Collaborative Conceptual Modeling Based on Co-occurrence Graph[C]//Requirements Engineering in the Big Data Era.Springer,2015:51-63.
[6] JIANG Y, ZHANG W, ZHAO H. Stigmergy-based collaborative conceptual modeling[C]//2014 IEEE International Conference on Global Software Engineeering Workshops.2014:45-50.
[7] JIANG Y, WANG S, FU K, et al. SCCMT: A Stigmergy-Based Collaborative Conceptual Modeling Tool[C]//2016 IEEE 24th International Requirements Engineering Conference (RE).IEEE,2016:401-404.
[8] THAKARE B S, DUBE M R. New approach for model merging and transformation[C]//2012 International Conference on Computer Communication and Informatics.IEEE,2012:1-5.
[9] WIKIPEDIA. Model-driven engineering - Wikipedia [EB/OL].: https://en.wikipedia.org/wiki/Model-driven_engineering, 2021.
[10] DAM H K, EGYED A, WINIKOFF M, et al. 2016. Consistent merging of model versions. Journal of Systems and Software [J], 112: 137-155.
[11] BASHIR R S, LEE S P, KHAN S U R, et al. 2016. UML models consistency management: Guidelines for software quality manager. International Journal of Information Management [J], 36: 883-899.
[12] CHONG H, ZHANG R, QIN Z. Composite-based conflict resolution in merging versions of UML models[C]//2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD).IEEE,2016:127-132.
[13] DI RUSCIO D, ETZLSTORFER J, IOVINO L, et al. Supporting variability exploration and resolution during model migration[C]//European Conference on Modelling Foundations and

Applications.2016:231-246.
[14] ABOUZAHRA A, SABRAOUI A, AFDEL K 2020. Model composition in Model Driven Engineering: A systematic literature review. Information and Software Technology [J]: 106316.
[15] XIAOCONG. xiaocong/uiautomator: Python wrapper of Android uiautomator test tool [EB/OL].: https://developer.android.com/training/testing/ui-automator, 2021.
[16] APPIUM. appium/appium: Automation for iOS, Android, and Windows Apps [EB/OL].: https://github.com/appium/appium, 2021.
[17] PAN M, HUANG A, WANG G, et al. Reinforcement learning based curiosity-driven testing of Android applications[C]//Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis.2020:153-164.
[18] 宋玲, 吕强, 邓薇, et al. 2012. 基于语义和结构的 XML 文档相似度的计算方法. 中文信息学报 [J], 26: 59-65.
[19] LI F, WANG H, LI J, et al. 2014. A survey on tree edit distance lower bound estimation techniques for similarity join on XML data. ACM SIGMOD Record [J], 42: 29-39.
[20] 赵艳妮, 郭华磊 2016. 基于有效路径权重的 XML 树匹配算法. 计算机工程与设计 [J], 4.
[21] NETWORKX. NetworkX — NetworkX documentation [EB/OL].: https://networkx.org/, 2021.
[22] JSON. JSON [EB/OL].: https://www.json.org/json-en.html, 2021.

致谢

本次毕业设计，从开始选题到完成论文用了半年多的时间，在这半年多内，我从前期调研，反复实验到设计算法，完成工具，最终得出毕业论文，一步一个脚印走到了现在，在这一过程中，我经历了挫折与迷茫，一次次尝试，一次次克服困难，能走到现在与身边人的帮助是分不开的。

首先要感谢我的导师张天副教授，在整个毕业论文的工作中张老师都给予了我巨大的帮助，启发我的思路，指导我完成整个工作，张老师的学术态度深深地影响了我。

其次要感谢王国新师兄，师兄在实验数据与设计思路等方面给了我莫大的帮助。

最后要感谢我的父母与朋友，他们在我大学四年中一直以来给我以支持和鼓励，没有他们就没有今天的我。