南京大學

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目	基于覆盖表示学习的测试用例排序系统
作 者 姓 名	王睿智
学科、专业名称	工程硕士（软件工程领域）
研 究 方 向	软件工程
指 导 教 师	陈振宇 教授，房春荣 助理研究员

2022 年 05 月 20 日

学　　号：**MF20320162**

论文答辩日期：**2022 年 05 月 20 日**

指 导 教 师：　　**（签字）**

The Test Case Prioritization System Based on Code Coverage Representation Learning

By

Ruizhi Wang

Supervised by

Professor Zhenyu Chen

Research Associate Chunrong Fang

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2022

学 位 论 文 原 创 性 声 明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不句含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：_____

日期：　　年　　月　　日

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于覆盖表示学习的测试用例排序系统

工程硕士（软件工程领域）专业2022届级硕士生姓名：王睿智

指导教师（姓名、职称）：陈振宇 教授，房春荣 助理研究员

摘 要

随着软件规模和复杂性的增加，软件迭代过程中产生的错误和缺陷也变得越来越复杂，越来越难以发现。同时，定位错误和缺陷的成本也在增加。在软件迭代过程中，如何保证软件的质量，尽快发现软件的错误和缺陷，已经成为软件开发者面临的一大挑战。软件测试是发现软件问题的关键方法，而回归测试是一种有效的测试手段，可以确保现有代码的正确性。然而，随着软件的迭代，测试用例集的规模逐渐增大，这给控制开发成本带来了巨大的挑战。对测试用例集进行优先级排序可以更快地发现错误，提高缺陷定位速率。这有利于软件开发者尽快修复问题，并降低软件回归测试成本，对软件测试来说具有重大意义。

本系统主要从代码覆盖技术的角度对测试用排序技术进行了实现。由于传统覆盖信息表示只涉及行覆盖信息，而忽略了代码与代码之间的关系属性，因此本系统提出新的关系表示方式，即将代码关系与测试覆盖关系保存在同一矩阵中。其中，使用抽象语法树来提取节点之间信息。传统的抽象语法树是以词为粒度构建树状模型，冗余数据较多，计算成本较高。本系统提出了一种以语句为粒度的抽象语法树。同时，本系统通过变异测试技术，获取测试用例在错误矩阵中的权重集合，并以此为依据获得理想情况下测试用例排序集合。最后，本系统通过神经网络模型对代码以及覆盖信息进行特征提取，将理想情况下的测试用例权重集合作为标签进行学习，获取测试用例排序推荐模型。

最后，本文从系统测试和实验验证两个方面对基于覆盖表示学习的测试用例排序系统进行了检测。通过系统测试，验证了系统的功能性和非功能性都是符合预期的。通过两个主要实验，分别验证了系统在相同程序多个版本，以及多个程序之间，对测试用例的排序都具有较好的预测效果。在多程序实验中，基于覆盖表示学习的测试用例排序系统在排序效果上，较全局贪心算法提升 20% 左右，较额外贪心算法提升 1% 左右，有较好的实用性。

关键词：软件测试，回归测试，测试用例优先级排序，抽象语法树

南京大学研究生毕业论文英文摘要首页用纸

THESIS: The Test Case Prioritization System Based on Code Coverage Representation Learning

SPECIALIZATION: Software Engineering

POSTGRADUATE: Ruizhi Wang

MENTOR: Professor **Zhenyu Chen**
Research Associate **Chunrong Fang**

Abstract

With software's increasing size and complexity, bugs and defects generated during software iterations become more complex and challenging to detect. At the same time, the cost of locating errors and faults is also increasing. In the software iteration process, ensuring the quality of software and finding software errors and defects as soon as possible has become a significant challenge for software developers. Regression testing is an effective means of testing to ensure the correctness of existing code. However, as the software iterates, the size of the test case set gradually increases, which poses a significant challenge to control the development cost. Prioritizing test case sets allows for faster error detection and increases the rate of defect location. This is of great significance for software testing as it facilitates software developers to fix problems as soon as possible and reduces the cost of software regression testing.

This system implements the test ordering technique mainly from the perspective of the code coverage technique. Since the traditional coverage information representation only involves line coverage information and ignores the relationship properties between code and code, this system proposes a new relationship representation, i.e., the code relationship and test coverage relationship are kept in the same matrix. The abstract syntax tree is used to extract the information between nodes. The traditional abstract syntax tree is to build a tree model with a word as granularity, which has more redundant data and higher computational cost. This system proposes an abstract syntax tree with utterance as the granularity. At the same time, this system obtains the weights of test case killing variants by variation testing technique and uses it to get the location of test

case ranking in the ideal case. Finally, this system extracts features from the code and coverage information by neural network model, learns the set of test case weights in the perfect case as labels, and finally obtains the test case ranking recommendation model.

Finally, this thesis tests the test case ranking system based on coverage representation learning from two aspects: system testing and experimental validation. Through system testing, it is verified that the system is functional and non-functional as expected. The two main experiments confirmed that the system has a good prediction for test case ranking in multiple versions of the same program and between various programs, respectively. Compared with the traditional test case prioritization algorithm, the test case ranking system based on coverage representation learning performs well in ranking effectiveness and execution time and has better practicality.

Keywords: Software testing, regression testing, test case prioritization, abstract syntax trees

目录

表 目 录

图 目 录

第一章　引言

1.1　项目背景及意义

近年来，随着互联网行业的迅速发展，软件逐渐进入各行各业 [2]。同时，为了增加软件功能、优化现有功能和修改存在缺陷 [3]，程序开发者对软件进行多次迭代，软件的规模也在逐渐扩大。为了保证软件的有效性和稳定性，软件测试成为软件开发不可或缺的一部分，成为了软件与用户接触的最后一道保障，软件测试的重要性逐渐凸显出来。

在软件迭代开发过程中，虽然早期版本可能已进行了全面测试，但是迭代后的版本无论改动有多少，仍然需要重新测试，这是因为软件的复杂性使得开发者无法保证自己可以纵观项目的全貌，并且在对早期版本的项目修改过程中，可能会影响到已有代码的功能，进而导致错误的发生。所以在程序后一版本的测试阶段，运行全部的测试用例仍有必要。

为了有效地测试后一个版本功能的有效性，多数情况会重用早期版本已有的测试用例，这样可以减少对已有代码测试的测试用例的开发，进而降低测试成本 [4]。但随着软件规模的扩大以及项目的多次迭代，软件的功能点逐渐增多，规模逐渐扩大，复杂性也在日渐提升，这使得软件回归测试在软件开发过程中所花费的成本逐渐增高 [5]。有文献表明，回归测试消耗了总测试预算的 80% 左右 [6]，占软件维护预算的 50% 以上 [7]。以 commons-lang[1]项目为例，在软件迭代过程中，从版本 2.X 到版本 3.9 测试用例的数量从 1975 个增加到 4599 个，增长 57.1%。这使得开发者逐渐探寻降低测试成本的有效方法和解决途径，从而将软件开发成本控制在可以接受的范围之内。

为了削减测试带来的巨大开销，国内外的相关测试人员进行了深入探究，寻找高效且自动化的回归测试方法，包括测试用例约简、测试用例选择以及测试用例优先级排序。其中，测试用例维护策略是一个核心问题，最简单普遍的维护策略即执行所有已有用例，但这种策略存在严重问题，如已有代码修改需要重新设计测试用例，如部分代码修改可能会影响相关接口导致已有测试用例失败，最重要的是随着软件规模的逐渐扩大，在一些测试用例场景因为成本限制可能会不允许执行完所有已有测试用例 [8]，使得这种简单的测试维护策略的弊端逐渐展现出来。

[1] https://github.com/apache/commons-lang

软件开发者在实际的软件开发过程中逐渐产生了想要约减测试用例或者更快的找到代码缺陷的需求，想要通过提前发现问题来尽早地解决，从而提高开发效率 [9]，降低软件错误带来的资损和时间成本。这对于大型的互联网公司尤其关键。在谷歌，源代码提交的频率很高（每分钟近 20 次提交），每天频繁执行 1 亿个测试用例 [10]。因此，如何降低回归测试的成本已经引起了学术界和工业界的广泛关注。学术研究人员和行业从业人员分别从不同方面对节约回归测试成本的方法进行了探索。

为了降低回归测试的成本，许多研究者致力于这一领域，并提出了许多方法，如测试用例约减、回归测试选择和测试用例优先级排序 [11]。测试用例约减 [12] 旨在通过排除冗余测试用例来减少测试用例的数量。回归测试选择 [13] 旨在仅选择和重新运行受代码更改影响的测试用例。测试用例优先级排序 [14, 15] 则旨在更改测试用例的执行顺序，以最大限度地实现早期故障检测。在这三个方面中，测试用例约减和回归测试选择都排除了一些测试用例，这可能会增加系统的不安全因素。相反，测试用例优先级排序是对测试用例所有用例进行操作，只是简单地重新排序，并不丢弃任何测试用例。因此，测试用例优先级排序没有任何故障检测损失，已经在研究中得到广泛研究并在实践中得到应用 [16]。

在此背景下，本文提出一种基于覆盖表示学习的测试用例优先级排序方法，并将该方法实现为测试用例排序系统。通过该系统，软件开发者可以将项目中的测试用例进行排序，并对排序结果效果进行评估，并应用到实际的开发测试中，这对在测试开发中尽早发现软件问题和缺陷具有重大意义，同时，可以降低软件测试时间成本。本文提出的以语句粒度构建代码节点之间关系及覆盖关系具有重大意义，既减少了储存空间和计算成本，同时也可以以有效粒度表示程序内在关系。本系统使用神经网络模型学习挖掘代码与测试用例之间的内在联系，并通过变异测试，获取测试用例的理想权重，进而对模型进行训练，最终达到可以对项目进行解析、自动获取测试用例权重及排序的目的，进而在软件测试过程中，为软件测试者推荐可以更快发现问题的测试用例排序。

1.2 国内外研究现状

测试用例优先级排序（Test Case Prioritization，TCP）通常应用于回归测试 [11, 17]。回归测试即当软件代码有所变动时，需要重新执行原有的测试用例集合，以确保软件的正确性和稳定性。TCP 可以帮助开发者更快地检测到故障，使开发者可以及时解决问题和缺陷。TCP 问题的探索最早可以追溯到在 1997 年 Wong 等人发表的文献 [18]。他们将测试用例历史覆盖信息和代码修改信息进行有效结合，针对特定程序版本提出一种混合方法，并开发出 ATAC 工具。该方法

首先借助测试用例约减技术，识别出冗余的测试用例集合，对余下的非冗余测试用例，根据其覆盖信息进行排序。

1.2.1 基于覆盖的 TCP 技术

首先对 TCP 技术进行深入探究的是 Rothermel 等人 [17]，他们的研究领域主要是根据代码覆盖率信息来获取测试用例优先级的排序。其中提出了两种经典的贪心算法，即全局贪心算法和额外贪心算法。其中，全局贪心算法是根据每个测试用例覆盖语句的多少来进行排序，简单直观；额外贪心算法是迭代地选择覆盖最多代码语句的测试用例，然后调整剩余测试用例的覆盖信息，将所选的测试用例标记为“已覆盖”，并重复这一过程，直到测试用例集中的所有测试用例都标记为“已覆盖”，最终获取测试用例优先级排序。

尽管基于覆盖信息的全局贪心算法和额外贪心算法对测试用例排序较为有效，但是与理想的测试用例排序结果之间仍然存在差距。随着 TCP 技术的发展，逐渐有人提出新的排序算法并逐渐融入新的技术到测试用例排序过程中，如 Li 等人 [19] 提出的 2-Optimal 贪心算法，这是基于语句覆盖和决策覆盖进行测试用例排序的一种算法。又如 Jiang 等人 [20] 提出的自适应随机测试技术，就是给定一组先前执行过并且没有暴露代码问题的测试用例，与这些旧测试用例有较大差异的新测试用例则更有可能揭示代码的错误与缺陷。并且 Jiang 等人认为，输入域中应该均匀的放置测试用例。随着深度学习领域的发展，有些学者也提出利用神经网络学习覆盖信息的想法 [21–23]，通过学习覆盖信息中的有效特征进行故障定位和测试用例排序。

在之后同样有许多学者提出基于覆盖信息的测试用例排序技术，但是全局贪心算法和额外贪心算法仍然是最为流行并且关注度较高的两种排序算法。

1.2.2 基于需求的 TCP 技术

基于需求的测试用例排序技术（Prioritization of Requirements for Test，PORT）是一种价值驱动的系统级测试用例优先级划分方法。PORT 根据四个因素对系统测试用例进行优先级排序：需求波动性、客户优先级、实现复杂性和需求的故障倾向性。这是 Srikanth 等人 [24] 在 2005 年提出的一种技术。Srikanth 等人对高级研究生软件测试班学生开发的四个项目进行了端口案例研究。结果表明，系统级别的端口优先级提高了严重故障的检测率。此外，客户优先级被证明是提高故障检测率的最重要的优先级因素之一。这项实验通过考虑潜在缺陷严重性，以经济高效的方式提高了用户感知的软件质量，同时，在新代码的系统级测试和现有代码的回归测试期间，也提高了软件严重故障的检测率。Krishnamoorthi 等人 [25] 在 2009 年也提出了相似的方法。其提出从软件需求规范中找到一个系

统级测试用例优先级模型，用来提高用户对高质量软件的满意度，该软件同样具有成本效益，并提高严重故障检测率。提出的模型基于六个因素对系统测试用例进行优先级排序：客户优先级、需求变化、实现复杂性、完整性、可跟踪性和故障影响。并采用了两种不同的验证技术对所提出的优先级技术进行了验证，并分三个阶段对学生项目和两组工业项目进行了实验。其方法同样可以提高软件严重故障的检测率。

1.2.3 基于历史的 TCP 技术

基于历史的 TCP 技术是从开发过程所产生的信息文件中收集对排除故障有用的元素，并基于这些信息元素来进行测试用例排序。在开发过程中，软件开发团队将产生各种软件开发过程的中间产品，如需求文档、接口文档等。这些中间产品在软件迭代过程中将记录着软件的改动，并且可能会作为最终产品进行交付 [26, 27]。这些记录文档可以通过检查系统的哪些区域一起发生了变更，以及新的变更集合出现在哪里，从而显示代码源文件之间如何相互交互，或者系统在开发过程中如何演变 [28, 29]。从而这些记录可用于识别软件系统中的关联集群。关联集群由一组文件组成，如果多个文件一起重复更改以修复一组故障，则这些文件将被标识为关联群集，认为这些文件之间存在某种关联关系 [29]，从而产生关联聚类。关联聚类是通过在变更记录日期矩阵上奇异值的分解（Singular Value Decomposition，SVD）生成的 [30]。

Sherriff 等人 [31] 在 2007 提出一种基于奇异值分解（Singular Value Decomposition，SVD）的回归测试优先级排序技术（SVD-based Regression Test Prioritization，SRTP）。SRTP 提供了一个新的框架，主要用于从测试和现场故障中收集更改记录，生成关联集群，并利用这些集群来指导回归测试优先级划分。Sherriff 等人提供一种基于变更记录和奇异值分解的方法，该方法可以优先考虑回归测试，以减少回归测试中出现的故障数量。这种方法会生成历史上倾向于一起更改的文件集群，并将这些集群与测试用例信息相结合，生成一个矩阵，该矩阵可以乘以表示新系统修改的向量，以创建测试用例的优先列表。

1.3 本文主要工作

本文所实现的系统是为了减少回归测试中的时间成本并尽早发现代码缺陷而设计，旨在通过该系统，可以为项目提供一个合理的测试用例执行顺序，从而减少由于项目测试代码量过大而导致的发现错误滞后、等待成本过高等问题。

本文通过以下方式来实现目标：

将代码之间的关系通过关系矩阵进行表示，以语句粒度来定义代码节点，从

而减少单词粒度的数据冗余繁杂的问题，以及方法粒度的数据粒度过粗导致代码之间关系表示不充分的问题。同时将一个测试用例作为一个测试节点，通过覆盖信息来建立覆盖关系矩阵。最后将代码关系矩阵与覆盖关系矩阵进行合并作建立合并矩阵。

通过变异测试，获取测试用例杀死变异的变异信息矩阵，根据测试用例杀死变异体数量作为测试用例的权重值，根据权重对测试用例进行排序，并将其作为测试用例顺序的理想顺序。

通过神经网络模型，将合并矩阵与测试用例排序集合进行输入，通过学习输入信息的内在特征从而获取训练后有效的程序测试用例顺序推荐模型。最后将项目的合并矩阵输入后，获得有效的测试用例推荐排序，并对获取的测试用例进行评估。

综合上述方案和设计，本文主体部分通过使用 Java 基于 SpringBoot 框架进行搭建开发，数据预处理及神经网络模型训练部分使用 Python 语言进行编写，使用 MySQL 数据库对数据进行存储，对外使用 HTTP 方式提供访问接口。

1.4 本文组织结构

第一章为引言，本章主要介绍了软件测试在发展过程中遇到的问题，阐述软件回归测试给软件开发行业带来的巨大挑战，进而引出有效的测试用例排序方法具有重大意义。

第二章为相关技术介绍。本章主要对项目实现过程中使用到的技术做了阐述，包括测试用例排序常用技术、变异测试技术、覆盖测试技术以及系统搭建过程中使用的前后端开发技术。

第三章为本文的需求分析及概要设计。本章重点从设计的角度来阐述系统的样貌。首先分析了项目的功能性需求与非功能性需求，然后详细阐述了系统的总体设计，最后对各个模块进行了详细设计构建。

第四章主要介绍了系统的详细实现。本章主要阐述了第三章设计的实现方式，使用顺序图展示各个模块的调用及运行流程，使用主要代码阐述系统关键部分的实现，最后对系统界面进行了展示。

第五章为系统测试。主要分为系统功能性、非功能性测试和效果测试。通过单元测试和集成测试验证功能性、非功能性需求是否实现。通过两个实验，对系统的效果进行测试，并对实验有效性威胁进行了分析。

第六章为总结与展望。本章主要介绍了本文与本系统的贡献，同时对系统可以进一步提升的方面进行了分析，阐述了系统的不足之处以及改进方向。

第二章　相关技术概述

2.1　测试用例排序

TCP 定义 [17] 为：测试用例优先级排序是将测试用例根据某种策略定义测试用例的优先程度，从而对测试用例进行排序，进而通过该测试用例排序运行测试，可以更快的找到软件缺陷，进行改正。

定义 2.1. 给定一个测试用例集 T，PT 则是 T 的所有可能的排序的集合，目标函数 f 将 PT 映射到具体数值。TCP 问题则是寻找 $P' \in PT$，使得 $\forall P''$，$P'' \in PT(P'' \neq P')$，$f(P') \geq f(P'')$

2.1.1　排序策略

TCP 是一种提前发现代码缺陷的有效方法，其中多数 TCP 技术是根据测试用例覆盖代码信息作为依据，按照某种策略来进行排序的。比较常见的排序策略有自适应算法 [20]、搜索算法 [20]、全局贪心算法 [17] 以及额外贪心算法 [17] 等。因本系统主要涉及到全局贪心算法和额外贪心算法，所以本文仅对以上所述两种算法进行介绍。

（1）全局贪心算法是目前使用较多的一种排序算法。全局贪心算法是根据测试用例覆盖行数的多少进行排序，并获取当前覆盖代码排序最大值的测试用例。如果有 $n(n > 1)$ 个测试用例代码覆盖数相同且最大，则随机选取其中的一个作为本次挑选的测试用例。每轮都按照上述进行选取，最终获取测试用例排序集。

算法1形式化展示了全局贪心算法的排序过程。输入待排序的测试用例集合 $T(t_1, t_2, ...t_n)$ 与测试用例覆盖代码矩阵 M，输出为排序好的测试用例集合 R。其中 $M[i, j](1 \leq i \leq n, 1 \leq j \leq m, m$ 为代码单元数，如语句、分支等) 如果为 1，则表示第 t_i 个测试用例覆盖到第 m_j 行，如果为 0，则表示没有覆盖。

首先获取每个测试用例覆盖语句的情况，即 $Count$。然后根据 $Count$ 逐渐找出当前 T 中覆盖数最大的测试用例 t_k，将其依次放入结果序列 R 中。最后返回排序好的集合 R。

（2）额外贪心算法也是常用的一种测试用例排序算法。它也是根据测试用例的覆盖矩阵信息，选取其中覆盖代码单元最多的测试用例作为本次迭代的输出。但是与全局贪心算法不同的是，额外贪心算法每一轮选取一个测试用例后会有

算法 1: 全局贪心算法

Input: 待排序测试用例集 T，覆盖矩阵 M
Output: 已排序测试用例集 R

$R \leftarrow \emptyset$
for each i $(1 \le i \le n)$ **do**
  for each j $(1 \le j \le m)$ **do**
    if $M[i, j] == 1$ **then**
      $Count[i] \leftarrow Count[i] + 1$
    end if
  end for
end for
while len(T) > 1 **do**
  $maxNum \leftarrow Integer.MIN_Value$
  $index \leftarrow 0$
  for each $tc \in T$ **do**
    $i \leftarrow getIndex(tc)$
    $num \leftarrow Count[i]$
    if $num > maxNum$ **then**
      $maxNum \leftarrow num$
      $index \leftarrow i$
    end if
  end for
  $R \leftarrow R \cup t_k$
  $T \leftarrow T \setminus t_k$
end while
return R

一个动态反馈的过程，下一轮的排序会根据更新后的代码单元覆盖矩阵进行排序，直到所有测试用例完成排序。

额外贪心算法与全局贪心算法最大的不同，在实现上是每一次遍历完矩阵 M 后，都要选取一个最大覆盖行的测试用例，选取后要将该测试所覆盖的行在矩阵 M 中全部去掉。当再一次遍历矩阵时，当前的矩阵会小于等于上一次迭代的矩阵。如算法2所示，前半部分与全局贪心算法相似，后半部分当遍历完一次 M 后，获取到最大的一个测试用例时，会删除掉这个测试用例在 M 中所有涉及的行，也即 $delete(M, i)$ 方法，就是删除掉 M 矩阵中的第 i 行，然后再进行下一次的迭代。通过这种方式最终获取测试用例排序结果。

2.1.2 覆盖准则

对于覆盖准则的研究，Jones 等人 [32] 提出了使用修正条件/决策覆盖（Modified Condition/Decision Coverage，MC/DC）的方式来保证测试的充分性。MC/DC 是一种结构性覆盖率标准，要求通过执行显示决策中的每个条件，以独立且正确地影响决策结果。该标准的制定有助于满足在安全关键应用中对复杂布尔表达式进行广泛测试的需要 [33]。

算法 2: 额外贪心算法

Input: 待排序测试用例集 T，覆盖矩阵 M
Output: 已排序测试用例集 R

while $len(M) > 0$ **do**
  $R \leftarrow \emptyset$
  for each i $(1 \le i \le n)$ **do**
    for each j $(1 \le j \le m)$ **do**
      if $M[i, j] == 1$ **then**
        $Count[i] \leftarrow Count[i] + 1$
      end if
    end for
  end for
  while len(T) > 0 **do**
    $maxNum \leftarrow Integer.MIN_Value$
    $index \leftarrow 0$
    for each $tc \in T$ **do**
      $i \leftarrow getIndex(tc)$
      $num \leftarrow Count[i]$
      if $num > maxNum$ **then**
        $maxNum \leftarrow num$
        $index \leftarrow i$
      end if
    end for
    $R \leftarrow R \cup t_k$
    for each $i \in len(M)$ **do**
      if $M[i, index] == 1$ **then**
        $delete(M, i)$
      end if
    end for
    $T \leftarrow T \setminus t_k$
  end while
end while
return R

具体来说，MC/DC 是将程序中的每个判定分支的所有可能结果至少取值一次，保证每个分支的情况都会涉及到，并且程序中的每个入口方法以及结束方法也要保证执行。同时，每个分支中的判定条件都会独立的对结果产生影响，也就是保证其他条件不变的情况下，一个分支条件改变后，会直接影响最终的结果。

2.1.3 排序评估

软件测试就是要检测出软件存在的问题和缺陷，而为了提高检测效率和质量，就需要高质量的测试用例。所以在对测试用例进行排序时，一个重要指标就是检测该排序是否可以更快的找出代码中的问题所在，也就是检测错误的速率。

（1）APFD(average percentage of fault detection)。Rothermel 等人 [8] 首先提出了 APFD 评测指标。APFD 指标主要是对测试用例检测错误的速率进行评估。对于某一特定的测试用例排序，其 APFD 取值范围是在 0-100% 之间，如果取值较

高，则说明该测试用例执行顺序可以更快的检测到代码中的缺陷，反之，则检测错误的所需的时间较长。通过 Elbaum 等人给出的 APFD 计算公式可知，假设有测试用例集 T，其中包含 n 个测试用例，m 个缺陷，给定一个测试用例执行次序，其中，TF_i 表示首个可检测到第 i 个缺陷的测试用例在该执行顺序中所处的执行位次，则 APFD 的计算公式为:

$$APFD = 1 - \frac{TF_1 + TF_2 + ... + TF_n}{nm} + \frac{1}{2n} \tag{2.1}$$

例如表2.1所示，测试例子中，该实例有 A、B、C、D、E5 个测试用例和 10 个缺陷，假设测试用例的缺陷检测信息已知，例如测试用例 B 可以检测出编号为 2、5、6、7 的缺陷。在该实例中，测试用例排序的目的是最大化缺陷的早期检测能力。

表 2.1: 测试用例检测缺陷表

	1	2	3	4	5	6	7	8	9	10
A		√				√				
B		√			√	√	√			
C		√				√	√			
D	√					√				
E			√	√				√	√	√

若测试用例的执行次序是 A-B-C-D-E，则 APFD 的取值为:

$$APFD = 1 - \frac{4+1+5+5+2+1+2+5+5+5}{5*10} + \frac{1}{2*10} = 35\%$$

若测试用例的执行次序是 E-D-C-B-A，则 APFD 的取值是:

$$APFD = 1 - \frac{2+3+1+1+4+2+3+1+1+1}{5*10} + \frac{1}{2*10} = 67\%$$

所以，在该实例中，第 2 次顺序的执行效果要优于第一次执行顺序。图2.1、图2.2表示不同执行次序下，测试用例执行比例与检测出的缺陷比例之间的关系。以图2.1为例，测试用例 A 执行结束后，也即执行了 20% 的测试用例后，只可以检测出 20% 的缺陷，对应图中点 (20，20)。将所有点连成折线，则 APFD 取值对应的是折线下方阴影占整个面积的百分比。通过对比两图可知，第二种测试用例排序可以尽早的实现错误发现，反映在图中为面积尽早的达到较高的数值。

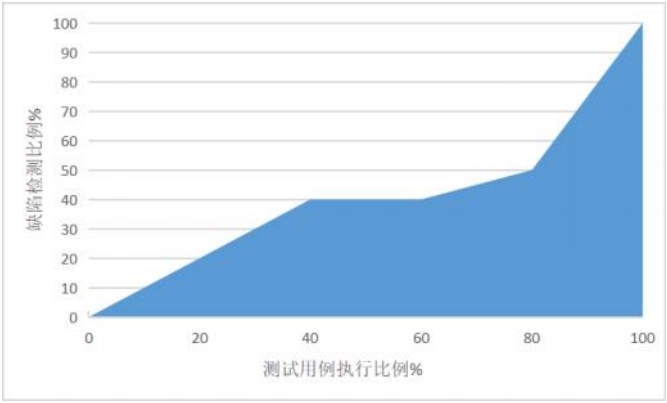


图 2.1: 排序 1 缺陷检测速率图

而在图2.2中，当测试用例 E 执行后，也即执行了 20% 的测试用例后，也已检测出 50% 的缺陷，从而第 2 个测试用例执行顺序将检测错误的时间点提前了，所以第二种顺序更好一些。

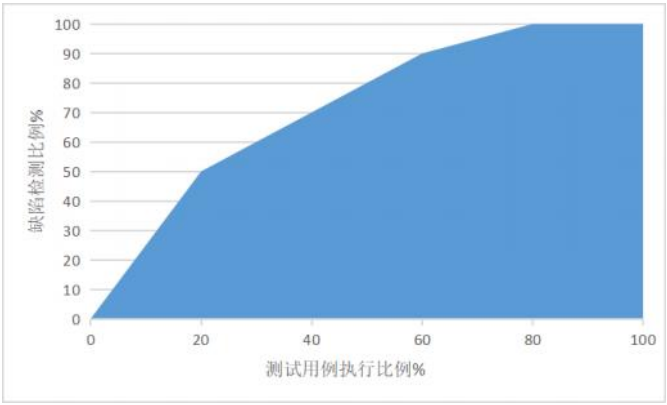


图 2.2: 排序 2 缺陷检测速率图

（2）除此之外，Elbaum 等人还提出一种新的评测指标 $APFD_c$[34]，该指标综合考虑了测试用例的执行开销和缺陷危害程度，其计算公式为

$$APFDC_c = \frac{\sum_{i=1}^{m}(f_i * (\sum_{i=TF_i}^{n} t_j - \frac{1}{2} * t_T F_i))}{\sum_{i=1}^{n} t_i * \sum_{i=1}^{m} f_i} \tag{2.2}$$

其中，t_i 代表第 i 个测试用例的执行开销，f_i 代表第 i 个缺陷的危害程度。这种评估指标将测试用例执行时间纳入的评估因素中，可以从测试用例执行成本的角度对测试用例排序进行评估。

2.2 变异测试

软件测试旨在测试软件的行为，如果测试用例通过，则证明软件运行良好，无缺陷问题，若运行失败，则证明软件存在异常行为。由于其简单性和实用性，工业界将软件测试作为保障软件质量的重要手段。然而，由于测试人员的编程风格及能力不同，最终导致测试用例的质量参差不齐。目前为止，学术界提出过多种度量标准，但大多数指标都依赖于代码覆盖率的概念 [35]，也即测试覆盖了多少源代码。虽然代码覆盖率被广泛应用，但其弊端明显。该方法只关注代码是否被有效执行，却不检测测试用例的质量以及是否通过。

变异测试 [36] 是一种崭新的技术，它在一定程度对测试用例的质量的评估具有指导意义。变异测试是通过修改部分代码的形式来引入缺陷，通过执行测试用例来检测缺陷。如果测试用例不能检测到代码中出暴露的问题，那么开发者有理由认为该测试用例的质量较差，无法识别应查询到的缺陷，需要对测试用例进行改进。近年来，许多突变测试工具逐渐被开发出来，主要用于突变测试方面的研究。在 Java 工具中，最受欢迎的是 MuJava[37] 和 Major[38]。但是这些工具是为了支持特定项目而构建的，因此，它们的普用性受到限制 [39]。

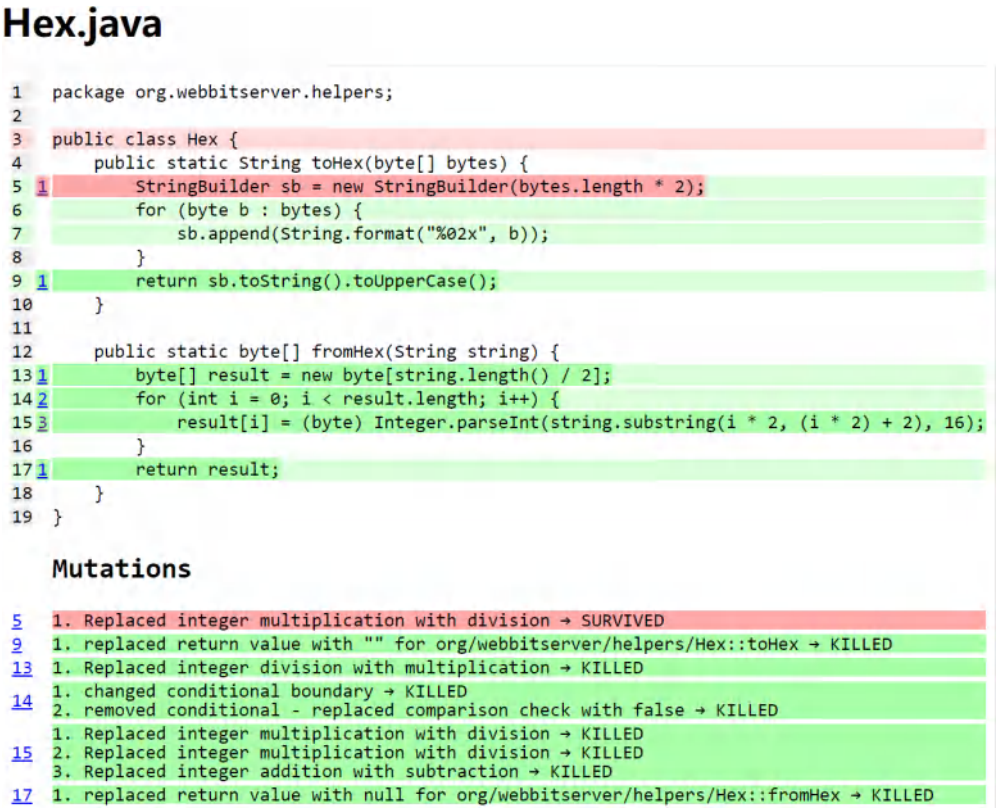


图 2.3: PIT 解析报告

PITest 是一个面向 Java 的变异测试系统。PITest 的运行的速度非常快，因为

它操纵字节码，并且只运行可能会杀死突变体的所有测试。PITest 的主要优点是它健壮、易于使用，并且与开发工具集成良好 [39]，因为它提供了 Maven 插件。最后，PITest 是开源项目，可以对其进行功能性扩展，并且能够积极维护。PITest 工具获得地址：http://pitest.org。

PIT 通过字节码操作产生突变体。与编译的突变文件相比，所采用的方法提供了显著的性能优势，因为字节码操作在计算上消耗较低，所以它将突变生成成本降低到零。此外，PITest 不需要与外界进行交互，这就减少了输入输出的操作，也大大降低了内存的开销。突变体在突变过程中所产生的字节码不需要进行持久化操作，而是储存在内存中。

突变过程主要分为两个阶段。第一个阶段为执行扫描阶段，这个阶段系统会将程序中的所有类进行扫描，并寻找可以进行突变的位置，储存在内存中。此过程会产生突变体的字节码，但是该字节码会被立即丢弃。第二个阶段，也就是产生突变体的阶段，此阶段会给每个突变体分配全局唯一的突变标识，该标识主要由突变的位置和突变的运算符两部分组成。突变体的位置是由方法和类的名称以及方法签名和发生突变的指令所定义的，这就使得系统可以重建多个突变体，大量的突变体描述可以通过这种方式保存在内存中。

PIT 生成的 HTML 报告使用不同的颜色来显示行覆盖率和变异分数，如图2.3所示，展示了应用 webbit-0.4.19[1]的部分报告。浅绿色的线条对应没有生成突变的代码，如图中的第 6、7 行。深绿色线对应测试用例被执行且失败的代码行，即突变体被杀死的情况，如图中的 9、13、14、15、17 行。浅粉色显示没有代码覆盖的行，即在测试范围之外的行，如图中的第 3 行。深粉色线是生成突变体并通过测试的代码，即存活突变体的代码，如图中第 5 行。在每个行号旁边，有一个数字嵌入了一个内部链接，此数字为该行代码生成的突变体的数量。如图2.3中，我们可以看到第 13、14、15 行分别产生了 1、2、3 个突变体，点击连接到 HTML 底部，也即 Mutation 下，可以发现都是深绿色，说明这些变异体都被杀死。而第 5 行的产生了 1 个变异体，点击数字链接，到 HTML 底部，该变异体为深粉色，说明该变异体存活。

2.3 覆盖测试

在覆盖测试发展的几十年间，虽然有人提出过一些质疑 [40, 41]，但是总体上被大多数学者所认可并广泛应用。最早采用代码覆盖率作为测量测试用例指标的是 Piwowarski 等人 [42]。

[1] https://github.com/webbit/webbit

代码覆盖率作为测试用例质量标准的重要指标，在软件工程领域被广泛应用。如 Evosuite[43] 将其用于测试用例生成。其提出了一种新的测试数据生成方法，称之为整体的测试用例生成。不单独优化每个测试用例，而是同时优化测试用例集中的所有测试用例，并且采用适应度函数同时考虑所有测试目标。该技术从随机生成的测试用例开始，然后使用遗传算法进行优化，以满足选定的覆盖率标准，同时使用测试用例大小作为次要目标。最后，将得到最佳的最小化测试用例集。再者，在错误定位方面，基于测试覆盖率解析的故障定位技术因为它的有效性和轻量级，如今得到了广泛的研究。基于频谱的故障定位技术（Spectrum-based Fault Localization，SBFL）[44–46]，成为了最流行的一种基于覆盖率的故障定位技术 [1]，它是通过统计分析失败和通过测试的覆盖率来识别有缺陷的程序实体。另一种较为流行的故障定位方式是基于学习的故障定位 [47] 它利用先进的机器学习技术，更充分地利用覆盖率信息，更细粒度的表示覆盖率，在此基础上，进一步学习测试覆盖率和测试结果之间的因果关系。

而与本文相关的 TCP 领域，大量的研究使用测试覆盖率作为排序测试的依据，并依据代码覆盖率而提出了许多测试用例优先级排序的算法。Rothermel 等人 [48] 提出的贪心算法，其就是将代码单元作为覆盖的单位元素，并根据其进行排序。随后，Zhang 等人 [49] 还将覆盖次数这一指标作为影响错误检测率的重要因素。

图 2.4: OpenClover 解析报告

OpenClover 可以测量 Java 和 Groovy 的代码覆盖率并收集多个代码指标。它不仅展示了应用程序中未测试的区域，而且还结合了覆盖率和指标来查找风险最大的代码。OpenClover 的测试优化功能可以跟踪哪些测试用例与应用程序代

码的每个类相关，这也是本系统主要使用的功能。由于 OpenClover 可以运行与应用程序代码更改相关的测试，从而显著减少了测试执行时间。同时，OpenClover 在定义覆盖测量范围方面的灵活性超过了其他工具，从而使得测试人员可以专注于测试代码的重要部分。OpenClover 不仅记录测试结果，还测量每个测试的单个代码覆盖率。使用者可以找到测试结果，哪些测试涵盖给定的类或源代码行，或者哪些类由测试执行。对于适用性方面，OpenClover 也非常有优势。OpenClover 为 Jenkins、Bamboo 和 Hudson 提供了专用插件。只需单击几下，就可以为构建设置代码覆盖率测量。通过 OpenClover 与 Ant、Maven 和 Grails 的集成，可以获得更大的灵活性。

如图2.4为 OpenClover 解析报告，报告中展示了某个源码文件中每行代码是否被覆盖，以及被执行的测试即覆盖该行的测试用例名称。其中 OpenClover 也对源码进行了有效的解析，点击图中的“-”可以对代码模块进行扩展与收缩。

2.4 抽象语法树

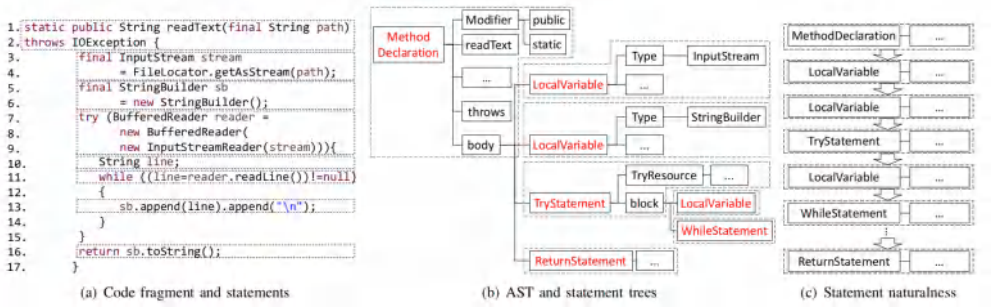


图 2.5: 语句生成树示意图

随着机器学习的发展，许多学者希望将机器学习与程序分析相结合，从而开辟出一个崭新的领域。但在探索过程中，研究者面临一个关键的问题，那就是如何有效的表示代码片段，并以此为基础进行后续的分析。传统的方式是通过将程序视为自然语言文本来进行表示，但这有极大的可能会遗漏源代码之间的语义信息。最新的研究表明 [50–52]，基于抽象语法树（Abstract Syntax Tree，AST）的神经模型可以更好地表示源代码之间的关系结构。

AST 是一种旨在表示源代码抽象语法结构的树 [53]。它已被编程语言和软件工程工具广泛使用。如图2.5[1] 所示，(a) 为源代码的代码示例，(b) 为结点转化的中间产物，(c) 为最终需要获取的语句关系图。AST 的节点对应于源代码的结构或符号。一方面，与普通源代码相比，AST 是抽象的，不包含标点符号和分隔符等所有细节。另一方面，AST 可用于描述源代码的词法信息和语法结构，

如图2.5（b）中所示的方法名称 ReadText 和控制流结构。一些研究在基于令牌的方法中直接使用 AST 进行源代码搜索 [54]、程序修复 [55] 和源代码差异 [56]。

2.5 系统搭建使用技术

2.5.1 Spring Boot

Spring Boot 是在 Spring 基础上的应用框架，Spring Boot 的出现简化了 Spring 应用程序的开发，让所有 Spring 开发者很快的入门，提供开箱即用的功能，提供各种默认配置来简化项目配置。提供内嵌式容器，简化 web 项目的开发过程。同时，也没有冗余的代码生成和 xml 配置要求。核心功能如下：

自动配置：Spring 常见的应用功能，SpringBoot 可以自动提供相关自动配置类，包括 JdbcTemplate、Spring Security、Thymeleaf 等；

导入依赖：如 SpringBoot 的 web 起步依赖是 org.springframework .boot:spring-boot-start-web，会根据依赖传递将其它所需依赖引入到项目中；

Autouator：提供在运行时检查应用程序内部情况的能力，包括 Spring 应用程序上下文配置的 Bean、线程状态、Spring Boot 自动配置情况等。

2.5.2 Layui

Layui 是一套开源的 Web UI 解决方案，采用自身经典的模块化规范，并遵循原生 HTML/CSS/JS 的开发方式，常适合网页界面的快速开发。Layui 区别于那些基于 MVVM 底层的前端框架，它更多是面向后端开发者，可以让没有前端开发经验的程序开发者快速上手，无需涉足前端各种工具，只需面对浏览器本身。同时，Layui 提供丰富的组件库，可以让页面更加美观，交互方式更加生动。Layui 主要有以下优点：

模块化开发：Layui 努力引导用户使用模块化开发的方式，将高效做到极致；

轻量级开发：Layui 的开发无需导入复杂的依赖包，非常适合小型项目开发。

2.6 本章小结

本章主要对本系统主要使用的技术进行了相关说明。首先对测试用例排序技术进行了总体上的概述，其次对覆盖测试进行了简单介绍，再者对变异测试及本系统使用的变异工具进行了介绍与变异报告的部分展示。然后对流行的代码表示方法 AST 做了简单说明，本系统主要在 AST 基础上生成语句粒度的关系树，这对已有的技术进行了借鉴与创新。最后，本文对系统使用的前后端技术进行了简单介绍。

第三章　基于覆盖表示学习的测试用例排序系统需求分析与设计

本章对基于覆盖学习的测试用例排序系统的需求分析与系统设计进行了详细说明。首先，对系统的总体需求进行分析和概述，其中包括对功能性和非功能性需求的阐述说明，同时，对系统中主要模块的需求用例进行了充分的展示。其次，是对系统实现总体架构的介绍，其中详细说明了系统使用的总体流程，以及其设计思路。最后，重点对各个重要模块的设计进行了阐述，包括逻辑设计、数据结构以及模块交互等。

3.1　系统整体概述

在软件开发过程中，随着迭代的进行，软件的体积逐渐增大，测试用例逐渐增多。为了保证软件的正确性，往往需要对测试用例全部执行，以保证原有代码不会因为新增的代码而出现错误。但是庞大的测试用例集往往运行时间较长，从而导致测试人员在提交后无法尽快地修复程序中存在的问题。如何先执行可能会出错的测试用例，成为了解决发现问题滞后的关键。本系统利用神经网络模型提出了一种解决方案。本系统通过解析大量程序结构，包括代码之间关系，测试用例与代码之间关系，作为神经网络的输入参数，得到一种具有较强普用性的测试用例推荐模型。从而当测试人员输入系统程序源码文件后，系统解析程序源码，调用训练好的模型后获取该项目的测试用例推荐，给予系统使用者测试用例执行顺序的推荐，从而使测试人员尽快发现问题，进行修复。

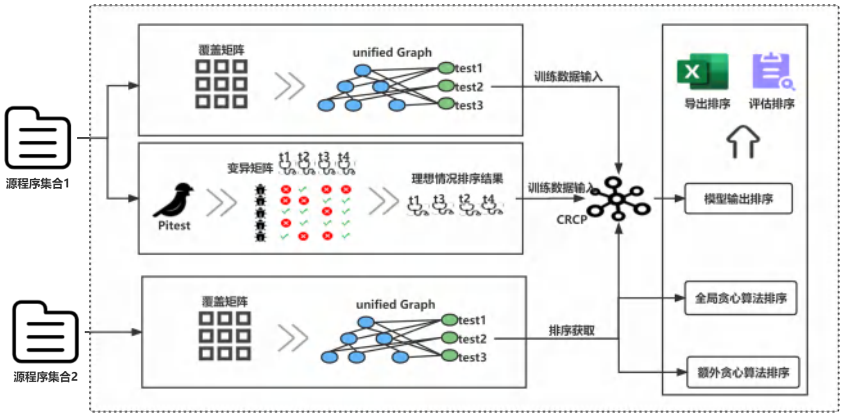


图 3.1: 系统整体流程图

如图3.1是模型的训练过程，而基于覆盖表示学习的测试用例排序系统是对训练完成模型的一种使用，这需要系统使用者提交代码的测试用例集合是完全成功的。本系统不仅是单纯的使用模型对程序的测试用例进行排序，而且还是一个代码的管理系统。用户可以提交代码，解析后可以查看代码的详细信息。当源码解析后，不仅可以获取模型推荐测试用例排序，系统还会根据已有的覆盖矩阵获取全局贪心算法和额外贪心算法计算的测试用例排序序列。这就使得用户可以根据系统所给予的推荐排序以及自身对代码的理解，来导出需要的测试用例排序，这样大大提高的了测试用例排序的灵活性和适用性。

3.2 需求分析

3.2.1 功能性需求分析

如表3.1所示为基于覆盖表示学习的测试用例排序系统的功能性需求分析列表，系统使用人员进入系统后，首先要会查看系统的首页内容。首页中包含已经上传的项目、正在解析的项目，以及最近上传的项目，这可以让用户对项目的情况有一个直观的认识。

系统使用者可以根据自己的需要导入新的项目,导入的项目需要支持Maven,同时需要测试用例完全通过后才可传入。导入项目完成后，用户可以在项目列表中查看已经导入的所有项目，用户可以根据需要编辑项目的基础信息，若项目不再需要，也可以删除项目。

系统使用者导入完项目后，可以对项目进行解析，也可以不进行解析，单纯的作为项目管理系统使用。若对项目进行解析后，系统会展示项目的解析进度。不同规模的项目，系统解析的速度是不同的。系统会对项目的结构进行解析，获取项目的有关信息以及测试相关信息反馈给用户。解析完成后，用户可以在解析项目列表中查看到项目的相关信息，同时可以对解析的项目进行相关操作。

已经完成解析的项目，会增加详情选项。系统使用者可以查看已经完成解析项目的详细情况，其中包含项目的方法数量、测试用例数量、测试用例名称、测试用例执行时间等相关信息。

当系统解析完项目后，用户可以根据项目名称和项目版本获取该项目的测试用例推荐顺序。该顺序是训练完成的神经网络模型给出，也是本系统的核心推荐模块。同时，用户也可以根据自己的需要来自定义测试用例排序顺序，只需要更改测试用例顺序的编号顺序即可完成。在此之余，用户也可以获取通过额外贪心算法以及全局贪心算法获取的测试用例排序。

用户可以使用系统来评估测试用例排序的质量，但是所评估的项目必须在

已经解析完成的列表中，测试数量与项目中测试用例数量要一致。本系统通过计算项目测试用例排序在特定错误矩阵中的 APFD 值反馈给用户。

最后，系统可以对用户选择的排序也即多种排序策略获得的排序进行导出，通过 Excel 表格的形式来交付。

表 3.1: 功能需求列表

需求 ID	需求名称	需求描述
R1	导入项目	系统系使用人员能够向系统中导入项目，导入成功或失败后，系统会给予反馈
R2	查看项目列表	系统使用人员能够查看系统中已经导入的项目，能够查看项目的基本情况，并对项目是否进行解析有直观的认识
R3	解析项目	系统会根据解析项目的路径找到解析项目所在路径，并对项目进行分层次解析处理，获取解析完成的数据后展示给用户，其中包括方法数量、测试用例数量以及测试执行时间等
R4	查看解析项目列表	系统使用人员能够查看，经过解析后的项目的列表解析，解析列表中的项目只有在解析完成后才会放入其中
R5	查看解析项目详情	系统使用人员能够查看具体解析后的项目详情，详情的主要内容包括系统的方法数量、测试用例数量、测试用例执行时间等，并对这些项目的信息进行结构化展示
R6	获取用例推荐排序	系统使用人员可以更据项目名称和项目版本，获取该项目推荐的测试用例排序，以及根据全局贪心算法和额外贪心算法获取的测试用例排序。获取排序的功能也为本系统的主要核心功能点
R7	评估测试用例排序	系统使用人员可以更据解析项目的名称和版本，获取某项目测试用例排序的 APFD 值
R8	项目管理	用户可以对项目进行管理，可以修改或删除项目，并且可以通过 excel 表格导出项目信息。

3.2.2 非功能需求分析

如表3.2是基于覆盖表示学习的测试用例排序系统的非功能性需求列表。系统的可用性是系统的关键，要保持较高的可用性。在系统在各个状态下，如果出现系统崩溃的情况，再次重启系统后，要保证及时恢复到崩溃时已有的状态。同时，必须对数据进行备份，保证系统如果出现故障时，数据不会丢失，并且系

统重启后会将数据恢复到终断时的状态。本系统为初始版本，当本系统作为商用系统，需要重点面向企业用户，这时需要对系统的各个模块进行扩展和升级，所以系统需要具有较高的可扩展性，同时也需要有较高的可读性，以减少他人维护代码的成本。系统需要做好各个模块之间的解耦工作，以及每个模块内部功能的聚合。也要对比较重要的模块在高层次上进行抽象处理，使得模块具有较高的可扩展性。最后，为了方便用户的使用，降低用户的学习成本，尽量做到人机交互的便捷性，对页面的设计要尽量简洁高效，以方便用户的使用以及提高用户的使用效率，便于用户可以尽快的上手该系统。

表 3.2: 非功能需求列表

需求名称	需求描述
可用性	本系统应该具有较好的可用性，应对数据进行备份。如果系统崩溃或出现网络问题时要，要保留已处理好的数据，并当系统进行重启时，会重新运行未完成的部分，并保证运行结果正确。
可扩展性	本系统的各个模块应提供较高的可扩展性，及时处理高层次的抽象功能，做好模块与模块之间的解耦，以方便以后在功能点上进行扩展与算法上的升级
易用性	系统的人机交互界面应尽量的简单美观，设计合理，使系统的使用者容易上手，降低学习成本

3.2.3 用例分析

根据以上对基于覆盖表示学习的测试用例排序系统的功能性需求分析，可以获得如图3.2所示的系统用例图，其中包括导入项目、查看项目列表、解析项目、查看已解析项目列表、查看某项目解析详情、获取测试用例推荐排序以及评估测试用例排序和导出排序 8 个系统用例。

本系统主要解决的是测试用例排序问题，但同时也会提供给用户管理项目的功能。用户可以对导入的项目进行增删改查的操作，同时也可以查看项目的详细信息。系统的核心功能是测试用例的排序，但是在排序之前需要对项目进行有效的解析。为了增加系统的交互性，项目是否解析由用户主动控制触发。对于测试用例排序而言，本系统只是提供推荐的测试用例排序，而非改变项目测试结构，系统并非输出输出项目代码，而是将推荐的测试用例排序结果进行交付。使用户可以对推荐的测试用例进行参考，并在 Junit 等工具中指定测试用例执行顺序达到尽早发现代码错误问题的目的。同时，系统也会根据项目代码进行变异处理，得到变异矩阵，从而提供给用户指定测试用例排序后，获取其评估结果的能力。

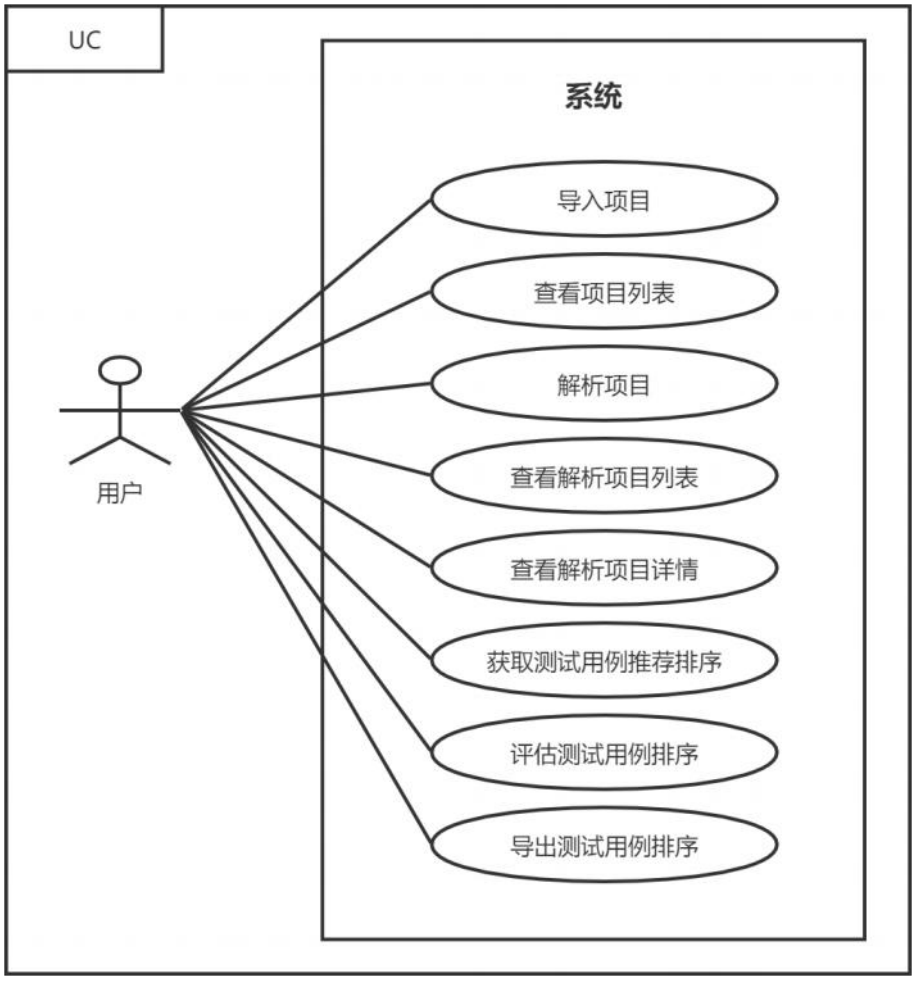


图 3.2: 用例分析图

3.2.4 用例描述

系统首页是系统的入口，首页展示了已经导入的项目、最近导入的项目和正在解析的项目等信息。系统使用者只有将项目导入系统后才能对程序代码进行进一步的操作。所以，导入项目是使用系统的第一步。本系统对项目的要求是支持 Maven 的项目，并需要将项目打成 zip 格式的压缩包后传入系统中。用例描述如表3.3所示。

导入项目后，用户可以查看已导入项目列表，系统使用人员可以对已经导入系统的项目有一个大致的了解，对项目的解析进度有一个直观的认识。导入系统项目的展示是根据导入项目的时间先后进行排序，方便系统使用人员定位最近导入系统的程序。再者，导入项目列表对导入项目的基本信息进行展示，包括项目名称、项目版本以及是否解析等。用例描述如表3.4所示。

表 3.3: 导入项目用例描述

ID	UC1
名称	导入项目
参与者	测试人员
触发条件	用户有新项目需要导入项目
前置条件	用户被授权
后置条件	无
优先级	高
正常流程	1、用户进入系统 2、用户点击导入项目按钮 3、系统跳转到项目导入页面 4、用户输入项目名称 5、用户导入项目文件 6、点击提交按钮 7、系统提示体提交成功

解析项目主要是对导入系统的项目进行解析，也是本系统的核心功能。该功能由系统使用人员在导入项目列表页面点击“解析”按钮触发。触发解析功能后，系统进行一系列自动化处理，如获取代码之间的关系以及代码与测试用例之间的关系等，该过程一般执行时间较长。解析完成后系统会在前端页面进行提示，告知用户是否已解析完成。同时，用户可以根据需要来决定是否对项目进行解析。当项目的格式不符合系统的解析格式或其他原因解析失败时，系统会提示失败原因。用例描述如表3.5所示。

表 3.4: 查看项目列表用例描述

ID	UC2
名称	查看项目列表
参与者	测试人员
触发条件	用户想要查看系统中已有测试项目
前置条件	用户被授权
后置条件	无
优先级	高
正常流程	1、用户进入系统 2、点击项目列表按钮 3、系统展示所有导入的项目，并按照时间降序排序进行展示

表 3.5: 解析项目用例描述

ID	UC3
名称	解析项目
参与者	测试人员
触发条件	用户点击解析按钮
前置条件	用户被授权
后置条件	项目在解析过程中产生的数据存入数据中
优先级	高
正常流程	1、用户进入系统 2、用户点击项目列表按钮，进入项目列表页面 3、用户点击解析按钮 4、系统对项目进行解析，解析完成后给出提示
异常流程	4a、系统出现异常导致解析失败 1、系统给出异常信息 2、在项目列表页面将该项目标红

当某些项目解析完成后，用户可查看已解析项目列表。已解析项目列表展示了已经完成解析项目的信息，用户可以对已经完成解析的项目有一个直观的认识，只有完成解析的项目才可对其测试用例进行排序操作。已解析项目列表按照项目解析完成时间的先后顺序对项目进行排序展示，包含了已经解析项目的基本信息，如项目名称、项目版本、测试用的数量、测试执行时间等。用例描述如表3.6所示。

表 3.6: 查看项目解析列表用例描述

ID	UC4
名称	查看项目解析列表
参与者	测试人员
触发条件	用户点击解析按钮
前置条件	已有解析完成项目
后置条件	无
优先级	高
正常流程	1、用户进入系统 2、点击项目列表按钮 3、系统展示所有导入的项目，并按照时间降序排序进行展示

解析项目详情页面将展示系统在触发解析项目后，生成的已解析完成的项

目信息情况。主要内容包括项目的导入时间，项目解析完成时间，项目方法的数量，测试用例数量以及测试用例执行的时间等。其中，系统还会给出四种测试用例排序的获取按钮，分别为原测试用例排序、推荐测试用例排序、额外贪心算法排序以及全局贪心算法排序等，用户可点击相应的按钮，获取相应的排序。同时，详情页面也展示不同排序所获取的测试用例排序。用例描述如表3.7所示。

表 3.7: 查看解析项目详情用例描述

ID	UC5
名称	查看解析项目详情
参与者	测试人员
触发条件	用户想要查看已经完成解析项目的详细信息
前置条件	该项目已经完成解析
后置条件	无
优先级	高
正常流程	1、用户进入系统 2、用户点击已解析项目列表按钮，进入已经解析项目界面 3、点击需要想要查看项目的详情链接

获取测试用例排序是本系统的核心功能。项目必须经过解析后才可以获取其测试用例的排序。用户可进入该项目的详情页面，根据需要获取不同算法所生成的测试用例排序。其中算法包括原测试用例排序、推荐测试用例排序、额外贪心算法排序以及全局贪心算法排序。点击按钮后，相应的排序会生成到页面右侧的表格中。用例描述如表3.8所示。

表 3.8: 获取测试用例排序用例描述

ID	UC6
名称	获取测试用例排序
参与者	测试人员
触发条件	用户想要获取已经完成解析项目的推荐排序
前置条件	该项目已经完成解析
后置条件	无
优先级	高
正常流程	1、用户进入系统 2、用户点击已解析项目列表按钮，进入已经解析项目界面 3、点击需要想要查看项目的详情链接 4、用户根据需要点击不同算法获取的排序

用户获取完测试用例后，可对获取的测试用例的质量进行评估。用户可跳转到测试用例排序评估页面，对特定项目的测试用例排序进行评估。首先，用户需要选择已经解析完成的项目序号或项目名称及版本来确定项目，输入相应的测试用例排序，然后通过点击评估按钮来获取项目的该测试用例排序的 APFD 值。用例描述如表3.9所示。

表 3.9: 评估测试用例排序用例描述

ID	UC7
名称	评估测试用例排序
参与者	测试人员
触发条件	用户想要评估已知测试用例排序的效果
前置条件	该项目已经完成解析，并已获得排序
后置条件	输出盖排序的 APFD
优先级	高
正常流程	1、用户进入系统 2、点击用例排序评估按钮 3、选择已解析项目的项目序号或项目名称和项目版本 4、输入该项目想要评估的测试用例排序 5、点击评估按钮

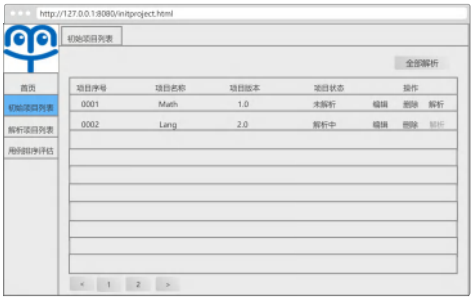
系统使用人员最终可以生成某项目的测试用例排序，导出测试用例排序也是本系统的基本功能。导出测试用例排序为后续使用该测试用力排序提供了便捷。系统以 Excel 格式的形式来导出测试用例排序。用例描述如表3.10所示。

表 3.10: 导出测试用例排序用例描述

ID	UC8
名称	导出测试用例排序用例描述
参与者	测试人员
触发条件	项目完成解析并获取排序
前置条件	点击导出排序按钮
后置条件	无
优先级	高
正常流程	1、用户进入系统 2、点击解析项目列表按钮，进入已解析项目列表 3、点击全部导出或者导出排序按钮 4、系统导出全部或者单个项目的测试用例排序

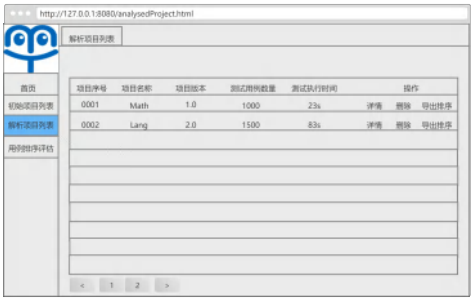
3.2.5 UI 设计

(a) 首页 UI 设计

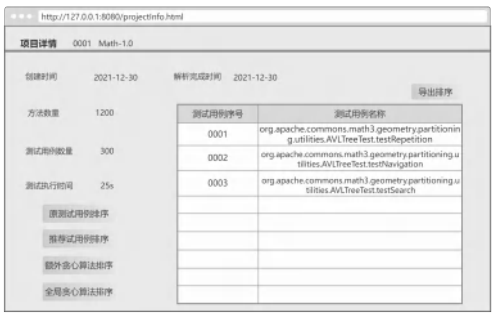


(b) 初始项目列表 UI 设计

(c) 导入项目 UI 设计



(d) 解析项目列表 UI 设计



(e) 解析项目详情 UI 设

(f) 用例排序评估 UI 设

图 3.3: UI 设计图

首页页面展示了所有项目的整体情况，包括最近上传项目，解析项目情况以及正在解析项目进度情况。除此之外，还展示了项目总数，以及解析项目的总数等，使用户对系统中已有项目有一个整体直观的掌握。左侧为导航栏，可以便于用户快速的定位到该系统的主要功能页面。首页尽量以简洁的形式展示较多的内容，让用户可以直观的感受到系统的主要功能，并提供直接点击方式进入功能页面。页面设计如图3.3（a）所示。

初始项目列表页面展示了已经导入系统的程序信息。为符合页面设计一致性原则，页面左侧统一使用导航栏，右侧为页面主要信息内容。页面主体展示了项目的基本信息，其中包括项目序号，也即项目的唯一标识，项目名称，项目版本，项目状态以及相关操作等。为了页面的简洁性，项目信息进行分页展示。页面设计如图3.3（b）所示。

导入项目页面因功能简单，所以设计更加简洁。整个页面只包括项目名称、项目版本、上传的项目以及提交项目按钮。页面设计如图3.3（c）所示。

解析项目列表页面与初始项目列表页面设计相似，主要展示的内容为已经解析完成后的项目信息，其中包括项目序号、项目名称、项目版本、程序的测试用例数量以及其测试执行的时间等。解析项目基础信息同样进行分页展示。页面设计如图3.3（d）所示。

项目详情页面主要展示已解析完成后项目的所有信息。页面主要分为三个部分，第一部分为左上方静态信息展示，该部分展示的信息包括项目创建时间、项目解析完成时间、项目的方法数量、项目的测试用例数量以及项目测试用例的执行时间。第二部分为多种排序获取接口，及左下方的四种测试用例排序获取按钮，其中包括原测试用例排序、推荐测试用例排序、额外贪心算法排序以及全局贪心算法排序。第三部分以及动态展示部分，该部分以表格的形式进行展示，展示内容为测试用例的排序，其中包括测试用例序号和测试用例的名称。该部分的内容会根据用户点击第二部分按钮的不同而展示不同的内容。最后，该页面提供导出排序接口，用户可以点击“导出排序”按钮导出动态表格中展示的测试用例排序。页面设计如图3.3（e）所示。

测试用例评估页面因内容简单，所以使用较为简洁的方式与用户交互。系统提供项目序号、项目名称以及项目版本的下拉框供用户选择评估项目。下拉框展示信息为系统已经完成解析的项目信息。用户可将该项目的测试用例排序输入到“测试排序”对应的文本框中，点击“评估排序”按钮获取该项目的 APFD 值。页面设计如图3.3（f）所示。

3.3 总体设计

3.3.1 总体架构设计

基于覆盖表示学习的测试用例排序系统是一个 Web 服务平台，系统的开发主要使用 Java 和 Python 语言。其中模型训练部分以及训练模型的数据处理部分使用 Python 语言进行编写，并独立为两个不同的服务。而与用户交互的系统部分使用 Java 语言进行编写。系统总体架构设计如图3.4所示。

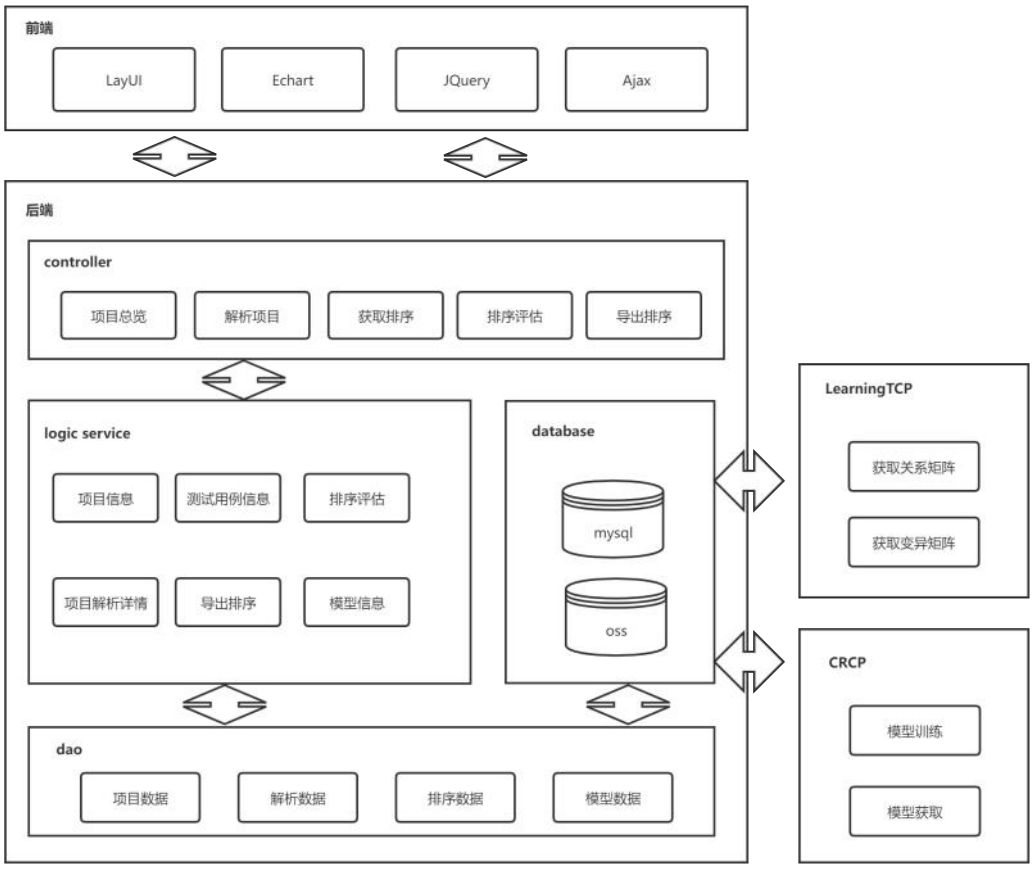


图 3.4: 总体架构设计图

前端方面，本系统主要使用了 LayUI 和 Echarts 框架进行开发。LayUI 框架是一套开源的前端页面解决方案，其采用了经典的模块化规范，并遵循原生 HTML/CSS/JS 的开发方式。其风格简约轻盈，而组件优雅丰盈，从源代码到使用方法的每一处细节都经过精心雕琢，非常适合网页界面的快速开发。前后端交互方面使用 Ajax 进行前后端的异步交互，这使得系统可以根据用户的请求快速加载新的内容到页面上，而无需重新加载页面，大大提高了系统的性能与用户体验。

本系统后端主要使用 Spring Boot 框架。由于国内外活跃的社区支持，Spring Boot 发展迅速，更新频繁，逐渐完善，极大的简化了编程人员的开发过程，同时在配置部署方面也极具高效性，这使得开发人员可以将大部分精力投注于业务本身，而非框架使用，减少了编程人员的学习成本。数据库方面，本系统目前只使用 Mysql 数据库对格式化数据进行持久化操作，但随着项目的迭代，可以增加 OSS 数据存储服务来存储更多的非格式化数据。

此外，对于程序分析部分以及模型训练部分使用 Python 语言编写，并封装为两个独立的服务供系统进行调用，这使不同功能模块之间进行了有效解耦，便于系统的迭代与扩展。

3.3.2 模块划分

如图3.4所示，为降低各个模块之间的耦合性，并增加各个模块之间的内聚性，本系统主要分为三个主要核心服务模块。

第一个模块为交互模块，也即系统的入口，用于与用户进行交互，并分别调用其他模块执行相应的功能。包括接收用户上传的项目，调用数据分析模块对项目进行解析，调用模型模块来获取某项目的测试用例排序，与数据库进行交互，获取解析完成的基础信息等。

第二个模块为数据的处理模块 (LearningTCP)，如图3.5所示。该模块主要用于将原程序源代码处理为模型训练的数据以及展示给用户的信息数据。该模块分为两个子模块，即生成关系图模块和获取变异矩阵模块，获取变异矩阵模块也会生成理想情况下的测试用例权重，该权重只用于模型的训练。

第三个模块即模型训练模块 (Test Case Prioritization with Code Coverage Representation Learning，CRCP)。该模块主要用于模型的训练，以及当系统调用模型时，返回通过模型获取的测试用例排序。其内部又主要分为两大模块，一个是训练模块，一个是模型的使用模块。

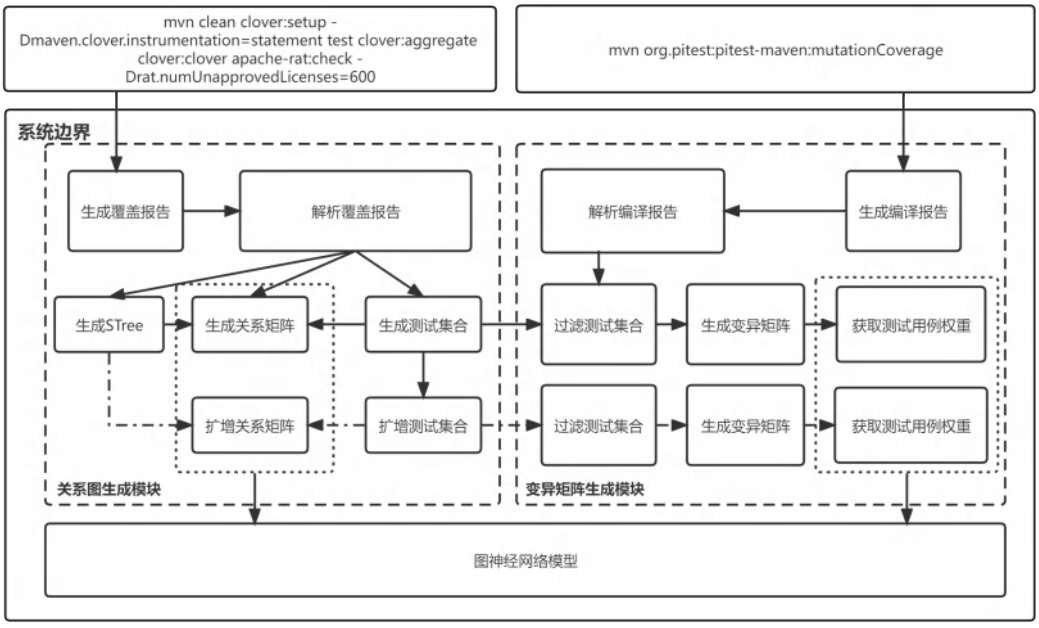


图 3.5: 数据处理模块设计图

3.3.3 整体设计

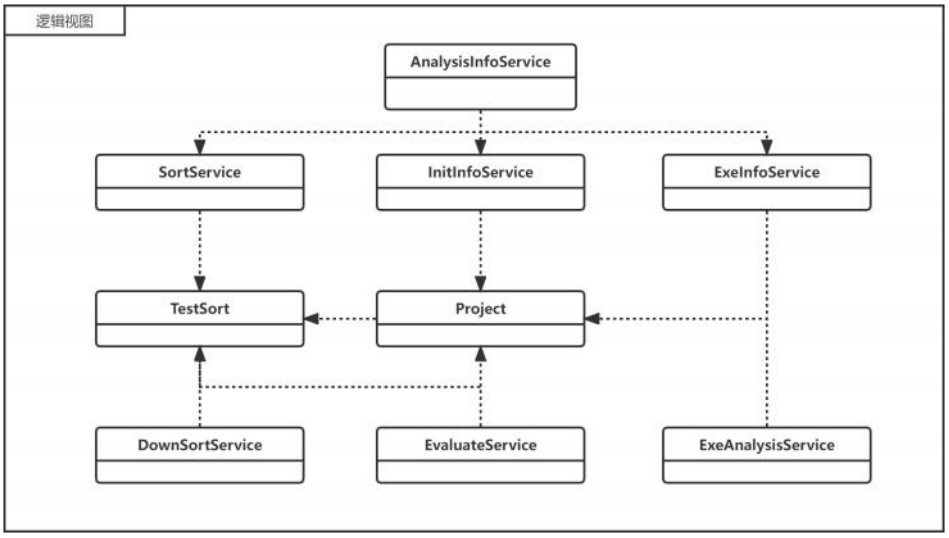


图 3.6: 逻辑视图

本章节分别从逻辑视图、进程视图、开发视图和物理视图的角度来描述并分析系统的总体设计。如图3.6为系统的逻辑视图。逻辑视图面向的是系统使用者，描述的是系统提供给系统使用者的服务，图3.6展示的是系统的核心类及核心类交互部分。

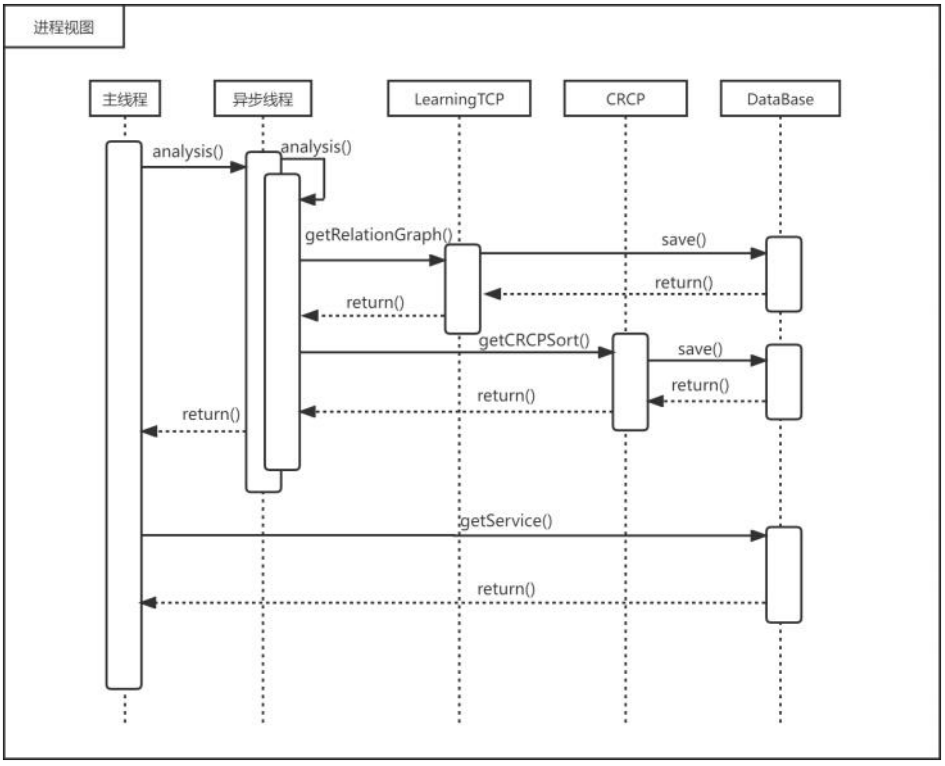


图 3.7: 进程视图

AnalysisInfoService 是展示已完成解析项目详情的核心类，它依赖调用了许多类来完成信息的展示功能。其中 SortService 类用例获取项目测试用例不同算法下的测试用例排序。InitInfoService 获取初始化项目的信息，ExeInfoService 获取执行项目分析过程中的完成信息。ExeAnalysisService 提供了进行项目解析命令的接口，以方便用户主动进行项目的解析行为。EvaluateService 提供了获取测试用例排序质量评估功能，即获取某项目某种算法排序的 APFD 值。DownSortService 则为系统内的排序提供了导出功能。

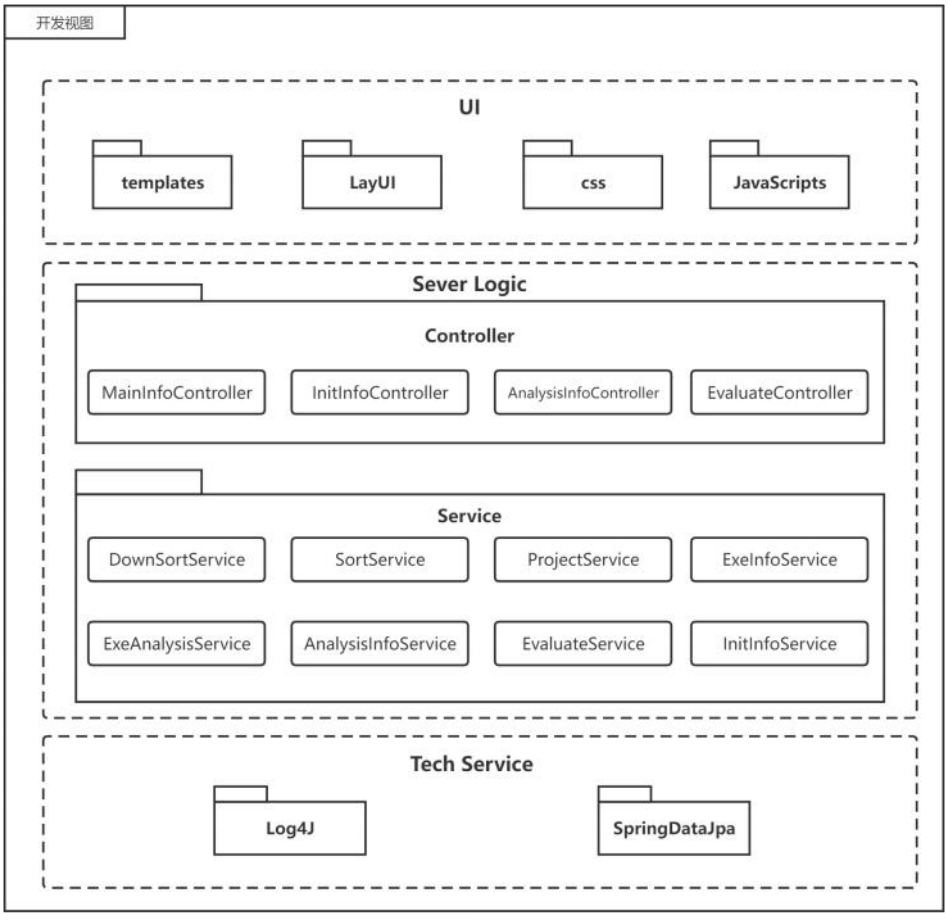


图 3.8: 开发视图

如图3.7为系统的进程视图，主要描述了不同程序模块之间的交互过程。主线程启动后，收到用户解析项目指令，主线程会调用异步线程执行解析操作，异步线程会调用 LearingTCP 模块，获取关系图模块。获取完成后，将得到的信息进行持久化操作。收到 LearningTCP 解析完成信息后，异步线程会调用 CRCP 模块，获取模型推荐的测试用例排序，并将推荐排序进行持久化操作。经过以上两个模块对项目的分析处理及持久化操作，主线程只需要从数据库 (DataBase) 中即可获取所需要的所有数据，后续的操作只需要主线程与 DataBase 交互即可。

如图3.8为系统的开发视图，开发视图主要是面向开发者，描述软件在开发环境下的静态组织。前端 UI 部分主要包括前端模板、LayUI 组件、CSS 样式和 JavaScript 资源文件。逻辑代码部分分为两层，分别为控制器部分（Controller）和逻辑服务部分（Service）。Controller 部分主要负责具体的业务模块流程的控制，因为系统开发初期功能简单，Controller 层效果体现较弱，但是为了系统的以后的可扩展性，遂设计该层。Controller 层主要包括 MainInfoController、InitInfoController、AnalysisInfoController 和 EvaluateController。Service 部分封装了的系统的主要逻辑代码，包括 SortService、AnalysisInfoService、InitInfoService 以及 ExeAnalysisService 等。Tech Service 为系统的基础技术服务，是提供系统技术支撑层，主要包括日志服务 Log4J 以及和数据库进行交互的 SpringDataJpa。

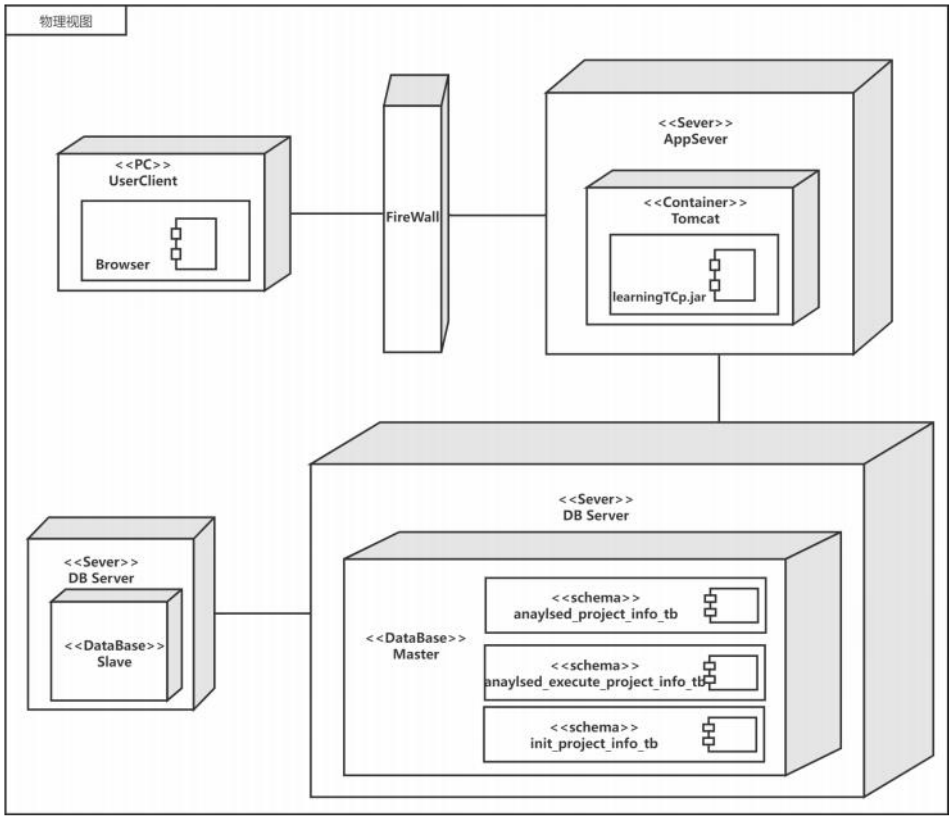


图 3.9: 物理视图

如图3.9为系统的物理视图，物理视图则是面向运维人员，从运维人员的角度出发描述系统的部署与通讯情况。用户使用浏览器发送请求，通过防火墙到达服务端，部署在服务器上的系统 Web 应用接收请求后进行处理。数据库作为单独的服务进行部署，通过使用数据库集群来保证数据的可用性。

3.4 关系图生成设计

3.4.1 代码关系图设计

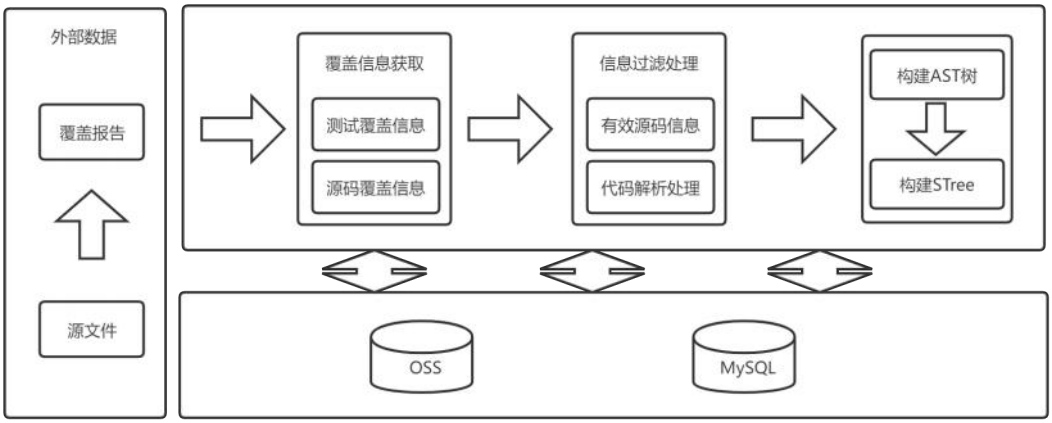


图 3.10: 代码关系获取模块架构图

如图3.10是生成语句粒度关系树（STree）的整体架构，该模块的输入是源代码经过编译测试后，并经过 openclover 工具处理后的覆盖报告，该报告中记录详细的覆盖信息，其中包括测试用例覆盖的代码语句行信息，以及每个方法中的每一行被覆盖的测试用例信息。最初设计过程中，是将整个项目的代码作为关系的获取对象，但是由于系统出发点主要是为了观测测试用例性能的好坏，这就使得源代码中有着许多与该目的无关的代码，也即无效信息，如类所在行、全局变量、注释以及空行等，所以在获得语句粒度的关系树之前，先对要处理的数据做一个过滤处理。

对于数据的存储，使用 OSS 和 MySQL 两种存储方式，OSS 用来存储 pkl 文件数据，而 MySQL 用来存储处理后的数据关系。

由单词粒度的 AST 转化为 STree 是该模块的核心部分。本系统使用 Javalang 开源工具处理过滤后的数据信息，从代码片段构建一个 AST，并将整个 AST 拆分为语句树，也即一棵语句粒度的树是由一条语句的 AST 节点组成，根在语句节点。其次，在总体的语句树上设计了一个递归编码器来捕获语句级的词汇和语法信息，然后用封装的节点类来表示它们。

Javalang 工具解析后的数据是以树的形式存储，也即每个方法为根节点，根节点下存放着语句以及单词的信息，一个方法体为一棵独立的树结构，而我们需要的是语句粒度的树结构，我们要做的就是要将处理后的树结构进行“组装”，将一行代码的节点“组装”成一个节点，并找出父子节点，从而建立关系。

其中最主要的是要识别其中作为语句开始的节点信息，例如“LocalVariable-Declaration”、“IfStatement”或者“ReturnStatement”等信息，这些节点意味着一

个语句的开始，同时也意味着这是需要和上一个节点进行分离，同时记录父子节点信息。

```
public int lengthOfString(String s) {
    if (s.length()==0){
        return 0;
    }
    HashMap<Character, Integer> map = new HashMap<Character, Integer>();
    return s.length();
}
```

图 3.11: 语句代码示例图

如图3.11所示，是一个简单的方法，其中包含简单的 if 语句和变量声明，我们需要捕捉的是代码行之间的父子关系，如第 1 行和第 2 行、第 5 行、第 6 行存在父子关系，第 2 行和第 3 行同样也存在父子关系，那么我们需要将每一行以一个节点的方式单独存储，同时保存各个节点之间的关系。

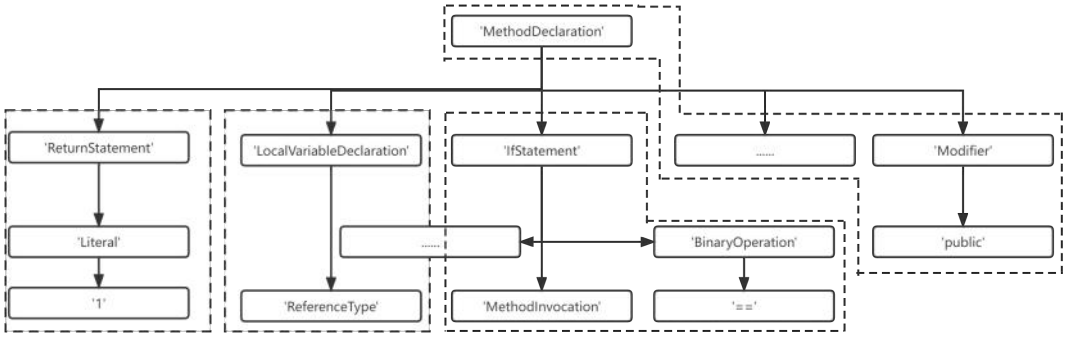


图 3.12: 语句结构解析图

如图3.12是通过 Javalang 工具解析的代码树，即 AST。本系统通过识别代码节点的 token 值来判断其是否还存在子节点，虚线部分是需要进行合并的节点信息，而在 AST 中同时也隐含了合并以后每个节点之间的关系。

3.4.2 测试覆盖图设计

如图3.13为测试覆盖关系获取流程图。测试覆盖图主要是对测试用例和行代码之间的覆盖关系做一个表示。因为在获取测试用例与代码行单元关系之前，需要先使用 OpenClover 工具进行覆盖的测试数据的获取。其中，OpenClover 工具获得的测试覆盖报告有测试覆盖方法和代码行之间的关系。所以，测试用例覆盖图的构建直接从已获得的覆盖报告中获取即可。

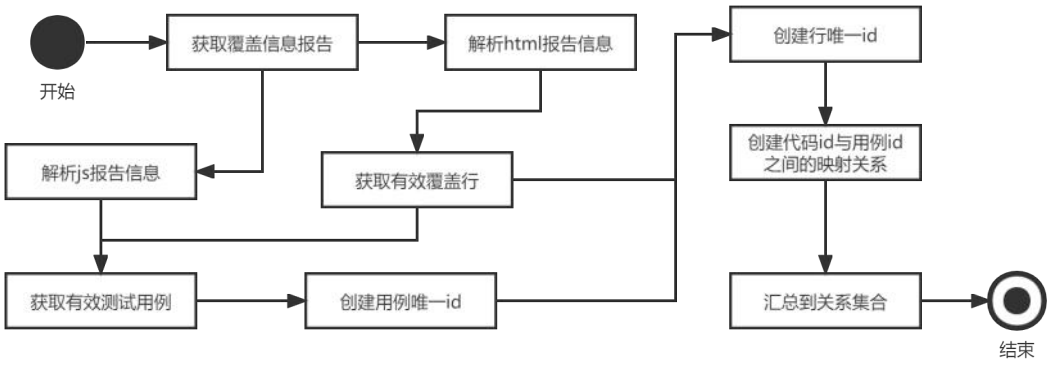


图 3.13: 测试覆盖关系获取流程图

测试覆盖报告中根据每个方法都会有一个 HTML 文件进行展示方法和测试用例之间关系，本系统主要通过解析这个 HTML 文件来获取节点关系。首先，获取被覆盖的行，我们称为有效行。获取到有效行后，在 HTML 定位到该行号，可以找到覆盖该行的测试用例。随后，为了进行统一化表示，需要对行代码与测试用例分配唯一的 id 值，从而进行全局表示。这就使得测试用例的 id 获取要等到所有行数分配完 id 后再进行表示，这样测试用例的 id 数值才会在代码之后，这样便于区分代码信息与测试信息。

将所有的 id 值分配完成后，根据每个方法的 HTML 获取同名称的 JS 文件，在对应的行中找到对应的测试用例编号，如果该测试用例存在，则将相应的代码行与测试用例之间的关系放入关系集合中。

3.4.3 总体关系图设计

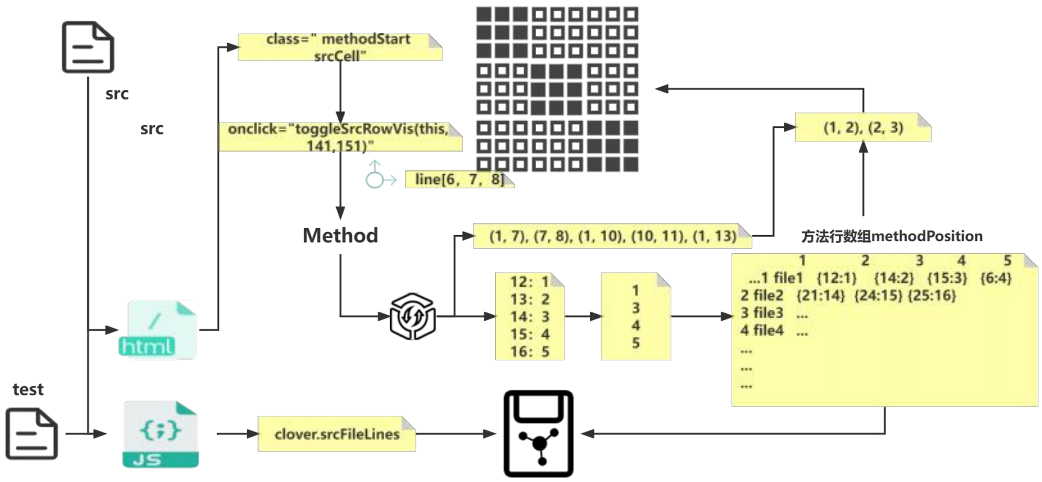


图 3.14: 总关系表示获取流程图

以上对代码关系图和测试用例关系图的获取设计已做阐述，最后一步是将这两个关系图进行合并。如图3.14所示，合并两图最重要的是将代码的表示 id 与测试用例的表示 id 统一起来，使每个节点在全图中有唯一的 id 表示方式。

首先是对代码行数的定义。第一种方案是行数为项目的总行数，其中包含大量注释和无效行，包括类名、包名、导入的依赖名称等。这种方案存在大量冗余代码行，并且测试数量较多，并且一个测试用例可能包含多个参数，不同参数的输入也会算作不同的测试用例。解决方案二次迭代过程中，去掉了注释行数，解析“*”和“/*”，去掉相应的行。因为测试名称大都为“test***”，解析相应的测试名称。但是源码解析还存在大量的冗余，包括“{”行和空格行以及 class 行和全局变量的行。三次迭代中，通过解析.html 文件，以及解析相应行的 onclick 属性，解析到方法所在的行，然后获取方法。这样获取在方法中存在大量的空格，依然存在冗余。最终，决定通过解析.html 文件，获得方法体，将方法体构建 stree，获得有效行数，同时过滤掉空格无效信息，经过这样改动后，冗余较少了。本系统使用语句粒度代码节点为一个最小单位，记录每个方法节点与代码节点的父子关系，同时记录测试节点与代码覆盖关系，最终将所有信息呈现到一个关系图中。

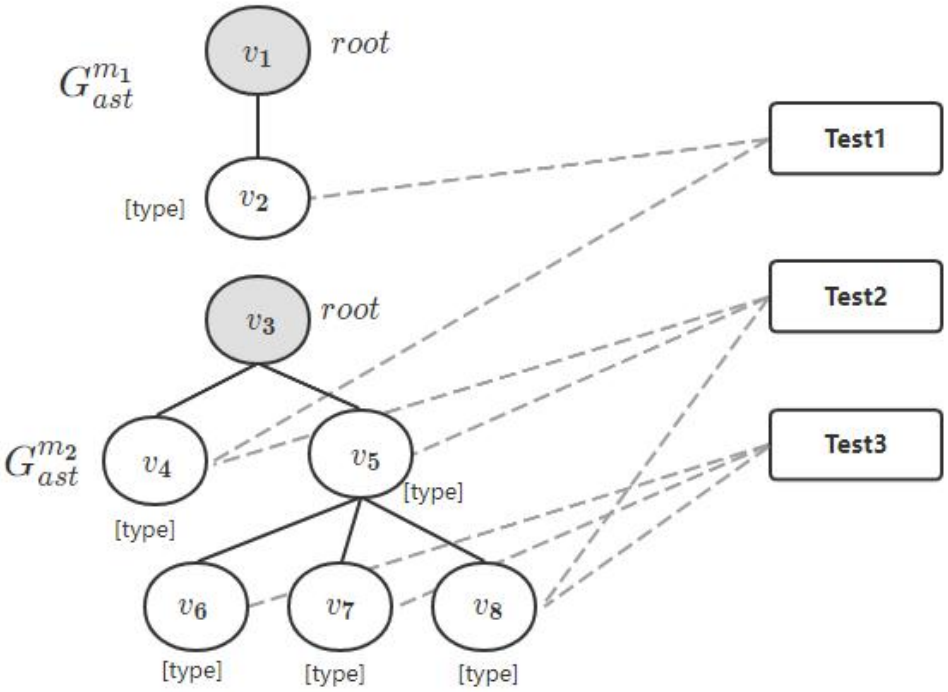


图 3.15: 结点关系图 [1]

如图3.15所示，一个方法语句粒度的关系由图 $\mathcal{G}_{ast}^{m_n}$ 表示，这是单词粒度的 AST 的一个子图，包括语句粒度的节点以及节点之间关系的边，即 $\mathcal{G}_{ast}^{m_n}=(\mathcal{V}_{ast}^{m_n},$

$\mathcal{E}_{ast}^{m_n}$)。而测试用例 $\mathcal{T}$ 与代码节点 $\mathcal{V}$ 的覆盖关系表示为 attr(v_t)，attr(v_t)∈{0，1}，其中 0 表示该测试用例没有覆盖到该节点，1 表示测试用例覆盖到该节点。最后，通过邻接矩阵表示整个图的连接关系。

其中，节点分为三个维度，即方法根节点、测试节点以及代码节点。每个节点都有其唯一表示作为图中关系的定位因素，同时每个关系还有其节点类型信息，即通过 AST 解析的 token 信息。关系同样也分为三个维度，即方法根节点与方法代码几点之间的关系，测试用例与方法根节点之间的覆盖关系以及测试用例与方法非根节点之间的覆盖关系。

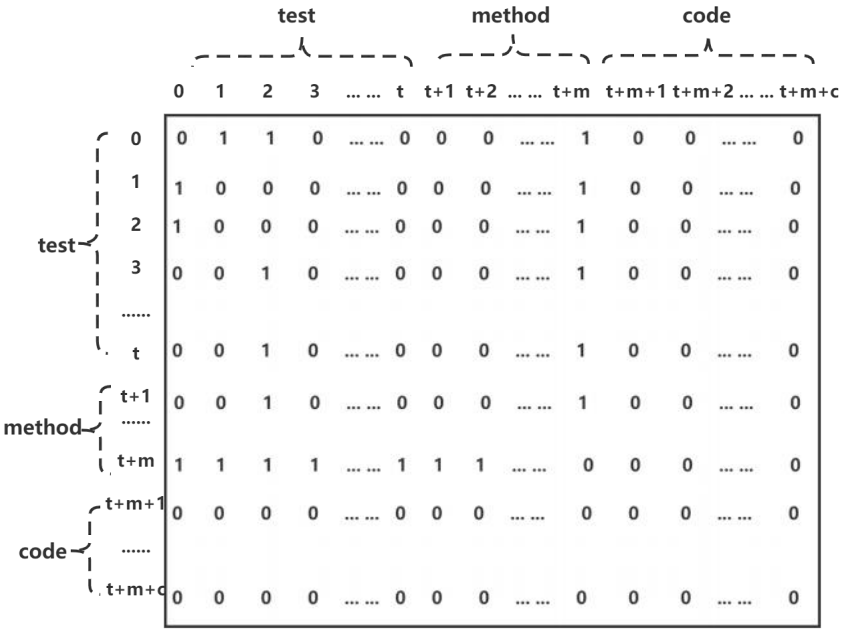


图 3.16: 关系矩阵

最终获取如图3.16所示的关系图，这是一个对称矩阵，从 0 到 t 为测试用例结点序号，从 $t+1$ 到 $t+m$ 为方法结点序号，从 $t+m+1$ 到 $t+m+c$ 为代码结点序号。当测试用例结点与方法结点或测试用例结点与代码结点关系为“1”，则说明该测试用例结点覆盖该方法或代码结点，如果方法结点与代码结点或者代码与代码结点之间的关系为“1”，则说明结点之间存在父子关系。

3.4.4 详细设计

如图3.17是关系矩阵获取模块核心类图，在系统层面则是用户进行解析的动作的触发过程。解析项目的入口是在初始项目列表页面的“解析”按钮，用户点击后，InitProjectController 类会接收请求，并将处理逻辑封装在 InitProjectInfoService 类中。InitProjectInfoService 类主要是处理初始项目请求的逻辑类，而 Ex-

eAnalysisInfoService 则封装了一些解析项目的过程中所需要的方法。InitProject-InfoRepository 和 ExeAnalysisInfoRepository 则是主要处理与数据库进行交互，前者主要是初始项目信息的交互，而后者则是处理在解析项目过程中产生的数据的交互。AnalysedProjectInfoSerivce 则是处理项目解析完成后的一些请求，Anal-ysedProjectInfoRepository 则是处理系统与数据库之间解析完成项目信息的交互。Util 作为工具类，主要封装了一些公共的底层方法，如文件的读取、执行终端命名等。InitProjectInfo 类是初始项目数据的封装类，主要存储了项目在上传到系统后，在没有解析前的一些项目信息。

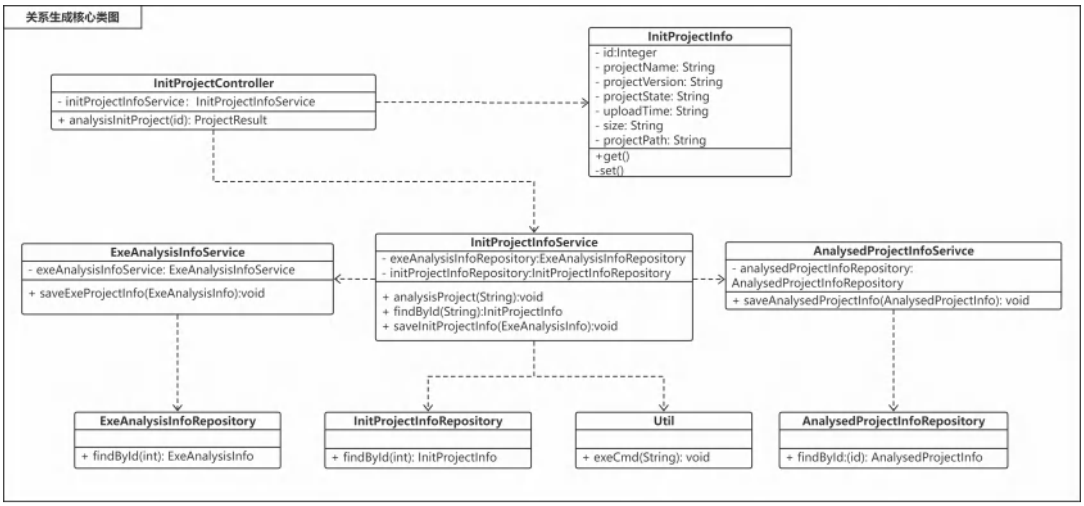


图 3.17: 关系获取模块核心类图

3.4.5 数据库设计

如图3.18所示为获取关系模块的 ER 图。InitProjectInfo 表示系统中导入项目的初始情况下的信息，analysedExecuteProjectInfo 表示项目在解析过程中所产生的信息。InitProjectInfo 和 analysedExecuteProjectInfo 是一对一的关系。

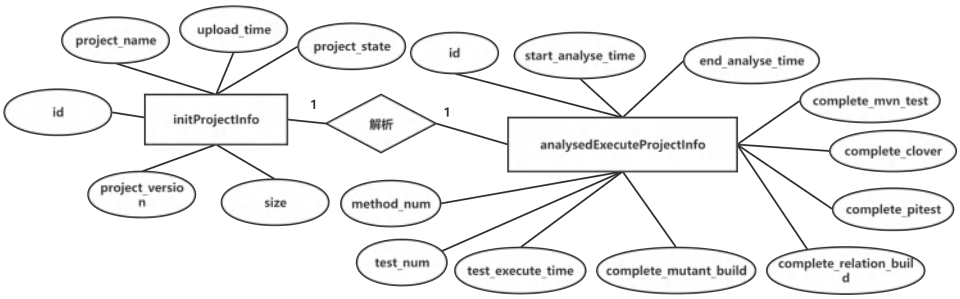


图 3.18: 关系获取模块 ER 图

如表3.11所示是 InitProjectInfo 表字段的详细说明。该表主要记录项目上传

后的详细信息，主要包含 project_name、project_version 等字段。

表 3.11: initProjectInfo 表

字段	含义	类型	说明
id	项目唯一 id	int	id 为该表的主键
project_name	项目名称	varchar	不可为空
project_version	项目版本	varchar	不可为空，默认为 1.0
project_state	项目状态	int	0 未解析，1 解析中，2 已解析
upload_time	项目上传时间	varchar	
size	项目大小	varchar	

如表3.12所示为在获取关系矩阵过程中所产生的信息内容。该表主要记录项目解析时每个过程的完成情况。为了减少存储空间以及提高系统与数据库的交互效率，每个过程的完成与否用 0 和 1 表示。该表主要包含了 complete_mvn_test、complete_clover、complete_pitest、complete_relation_build、、complete_mutant_build 和 test_execute_time 等字段。具体每个字段的含义表3.12进行了详细说明。

表 3.12: analysedExecuteProjectInfo 表

字段	含义	类型	说明
id	项目唯一 id	int	id 为该表的主键
start_analyse_time	项目解析开始时间	varchar	不可为空
end_analyse_time	项目解析完成时间	varchar	不可为空，默认为 1.0
complete_mvn_test	是否完成 mvn test	int	0 未完成，1 完成
complete_clover	是否完成 openclover	int	0 未完成，1 完成
complete_pitest	是否完成 pitest	int	0 未完成，1 完成
complete_relation_build	是否完成图关系构建	int	0 未完成，1 完成
complete_mutant_build	是否变成变异矩阵获取	int	0 未完成，1 完成
method_num	方法数量	int	
test_num	测试用例数量	int	
test_execute_time	测试用例执行时间	varchar	

3.5 用例排序生成设计

3.5.1 架构设计

如图3.19为测试用例排序生成架构设计图。模块中需要两部分外部数据，分别为训练模型使用的训练数据和外部需要排序的项目数据。该模块通过将训练数据进行变异处理，获取变异矩阵，根据变异矩阵获取理想情况下测试用例排

序，然后将该排序作为神经网络模型训练的输入数据，训练模型，另外变异矩阵在排序评估阶段也会使用。

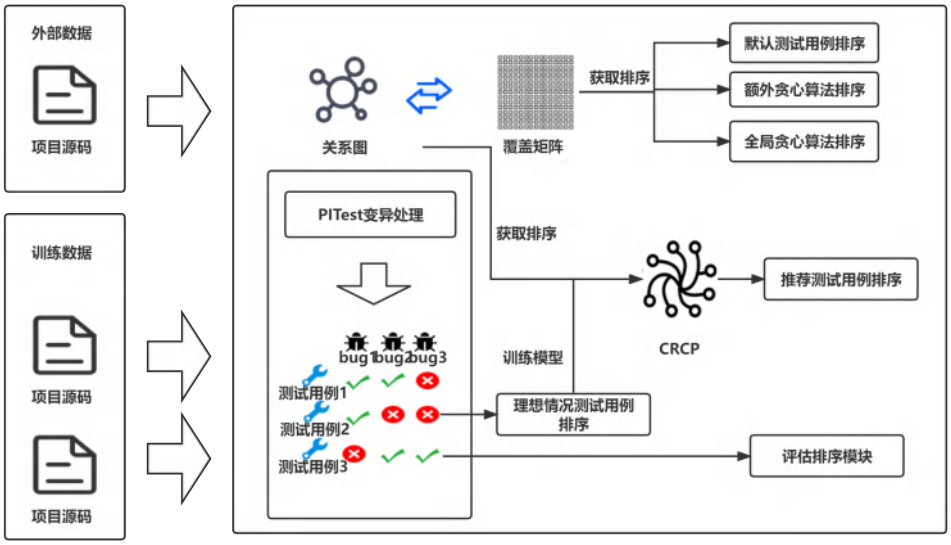


图 3.19: 测试用例排序生成架构设计

完成模型的训练后，对真正需要获取排序的项目进行处理，获取其覆盖矩阵，覆盖矩阵可由前部分获取的关系图转化获得。根据覆盖矩阵可以获取根据额外贪心算法计算的测试用例排序，也可以获得根据全局贪心算法计算的测试用例排序。将关系图输入模型，模型会返回该项目的推荐排序，也即该系统的核心排序推荐。

3.5.2 详细设计

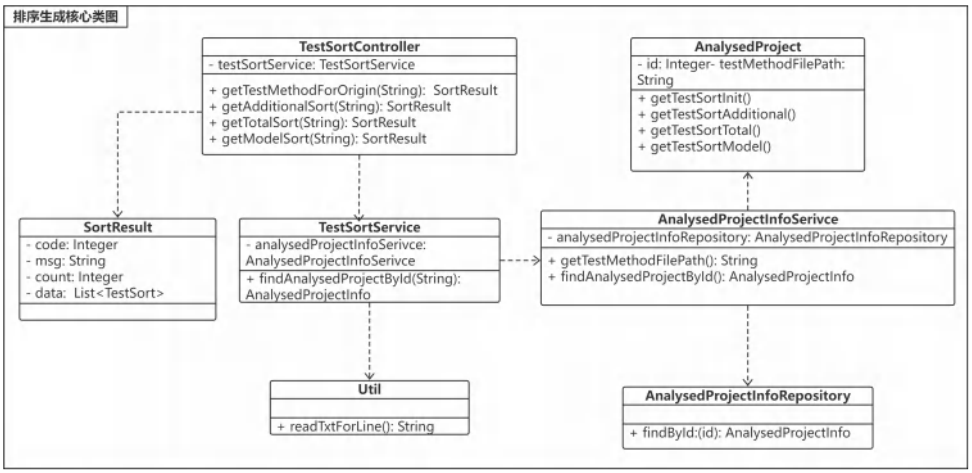


图 3.20: 排序生成模块核心类图

如图3.20为排序生成模块核心类图，该类图主要描述的排序获取过程代码的逻辑实现。映射到系统层面为点击不同算法排序按钮获取相应排序的过程。TestSortController 接收到前端传来的获取测试排序的请求后，会将处理逻辑封装在 TestSortService 进行实现。AnalysedProjectInfoSerivce 和 AnalysedProjectInfoRepository 的主要功能在 3.4 章节的详细设计中进行了阐述，这里不再赘述。AnalysedProject 主要封装了项目解析后的数据信息，包括根据不同算法获取的排序。SortResult 类则是根据前端的需要，分装的便于前端使用的数据传输模型。

3.5.3 数据库设计

表 3.13: analysedProjectInfo 表

字段	含义	类型	说明
id	项目唯一 id	int	id 为该表的主键
model_path	模型路径	varchar	不可为空
data_file_path	data.pkl 文件路径	varchar	不可为空
compare_file_path	compare.pkl 文件路径	varchar	不可为空
weight_file_path	weight.pkl 文件路径	varchar	不可为空
test_method_file_path	testMethodFile.txt 文件路径	varchar	不可为空
test_sort_additional	额外贪心算法排序	text	
test_sort_total	全局贪心算法排序	text	
test_sort_model	模型排序	text	
test_sort_user	用户自定义排序	text	

如表3.13为项目解析后的信息内容。该表主要记录项目解析完成后数据存放的地址以及不同算法获取的测试用例排序内容。因为某些结果文件数据较大，并且为非关系型数据，不利于存入 MySql 数据库中，遂将其保存为 pkl 格式的文件，将路径保存到 Mysql 数据库中。该表的主要字段为 model_path、data_file_path、compare_file_path、weight_file_path、test_method_file_path、test_sort_additional、test_sort_total、test_sort_model 和 test_sort_user。其中个字段的含义及说明表3.13进行了详细阐述。

3.6 图神经网络模块设计

本系统的图神经网络模块设计借鉴了 Grace[1] 的设计方式。Grace 是一种新的基于覆盖的故障定位技术。Grace 将覆盖率视为测试和程序实体之间的连接关系，认为整个关系可以由一个图形结构内在表示：测试和程序实体作为节点，覆盖率和代码结构作为边。

3.6.1 架构设计

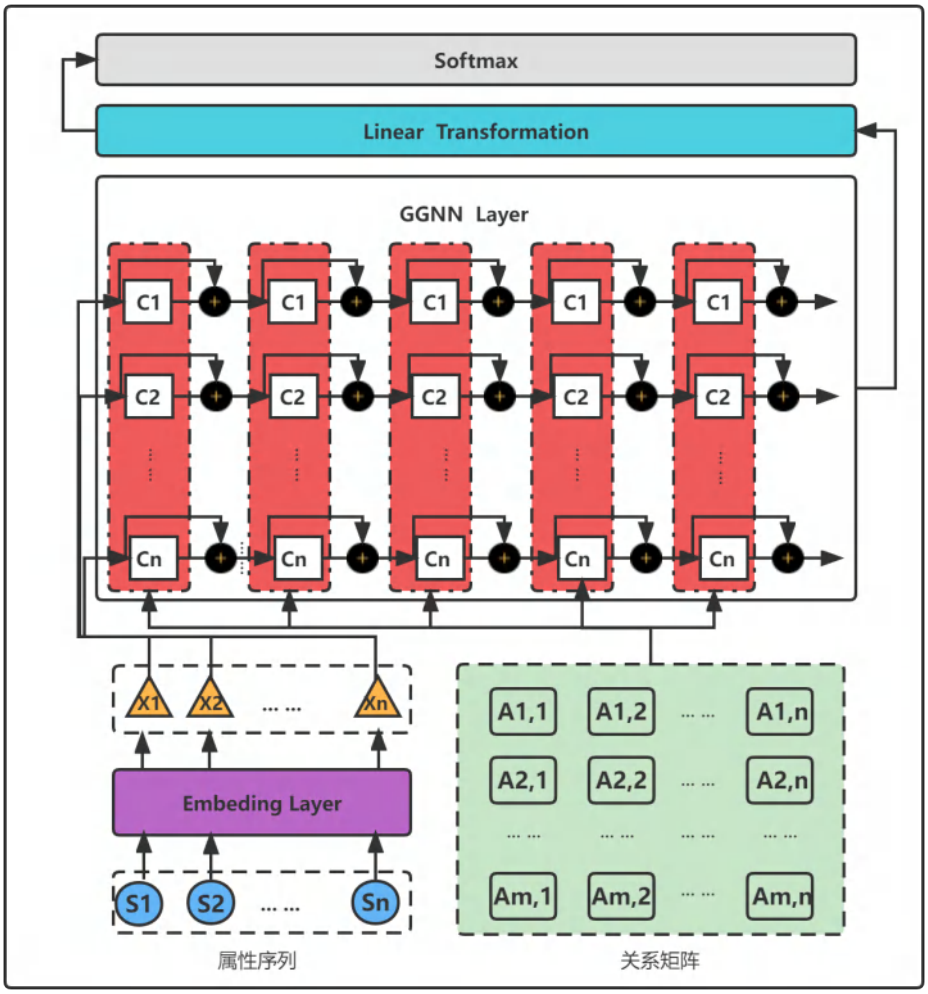


图 3.21: 模型架构设计图 [1]

如图3.21所示，为 CRCP 的模型结构，其主要分为四个某块:

第一个模块即模型的输入层，该层也为数据的预处理层，主要是将项目代码、测试用例覆盖信息进行解析，获得数据节点之间以及测试用例与节点之间的关系矩阵，同时，会获取方法的执行时间以及每个测试用例通过额外贪心算法获取的权重值等信息，将其作为测试用例结点的额外属性传入模型中。

第二层为为嵌入层，即将输入的节点信息表示为向量信息。方法节点和非根代码节点只有一个属性信息，即结点的 token 值，所以直接嵌入为 d 维向量。而对于具有多个属性的测试节点，模型会首先将 AST 节点类型信息嵌入到 $d-n$ 维，然后测试结点的额外的 n 个信息嵌入到 $d-n+1$ 到 d 维度。对于关系矩阵，

直接嵌入为 d 维，从而将所有信息表示出来。

第三层即 GGNN 层，该层通过五次迭代应用 GGNN 层。在第 t 次迭代中，对于每个节点，通过合并来自其相邻节点和来自先前迭代的信息来更新其当前状态。在 GGNN 中，通过利用 LSTM 中的输入门和忘记门来控制单元状态的传播，实现选通机制。

第四层为线性变换层。所有 GGNN 迭代计算后的输出进一步传入线性变换层，然后进行归一化处理。特别是 GGNN 层中最后一次迭代的输出，该输出进一步线性转换为实数。损失函数的计算公式为公式3.1，其中 $g(v_i)$ 表示节点的理想情况下测试用例排序，即 ground truth label，$p(v_i)$ 表示其预测结果。最后利用 sofamax 函数将所有节点的输出标准化，函数如公式3.1所示。

$$\mathcal{L}_{list} = -\sum_{i=1}^{n} g(v_i)log(p(v_i)) \tag{3.1}$$

3.6.2 流程设计

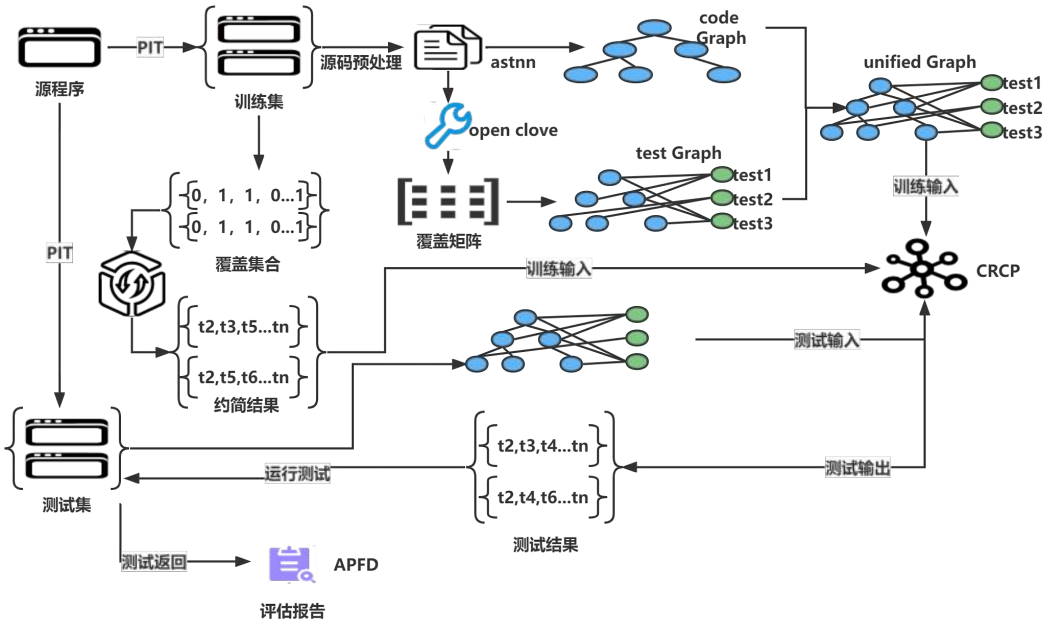


图 3.22: 模型训练流程图

如图片3.22所示是系统为得到训练模型而对源码的训练过程。首先会对源码重跑其测试过程，由于 openclover 工具的因素，需要保证所有的测试用例全部通过的程序集合作为模型训练的原始数据。执行完测试用例集合后，运行 clover 工具，获取其对测试用例的报告，其中包括源码中的每一行是否被测试用例执

行，以及被哪些测试用例执行等。通过解析 openclover 的 xml 报告，一方面获取方法体的起始位置，从而找到方法在源文件中的位置进行解析，获取代码接待之间的关系。另一方面，根据测试用例执行的行从而建立测试用例与代码节点之间的关系。最后将两个关系集合并，建立最终的关系矩阵。

与此同时，需要对源码进行变异处理，获取程序的变异体集合以及变异报告。通过解析变异报告构建变异矩阵，变异矩阵作为错误矩阵，根据错误矩阵使用理想情况下排序算法获取测试用例权重，以此作为模型的输入标签。

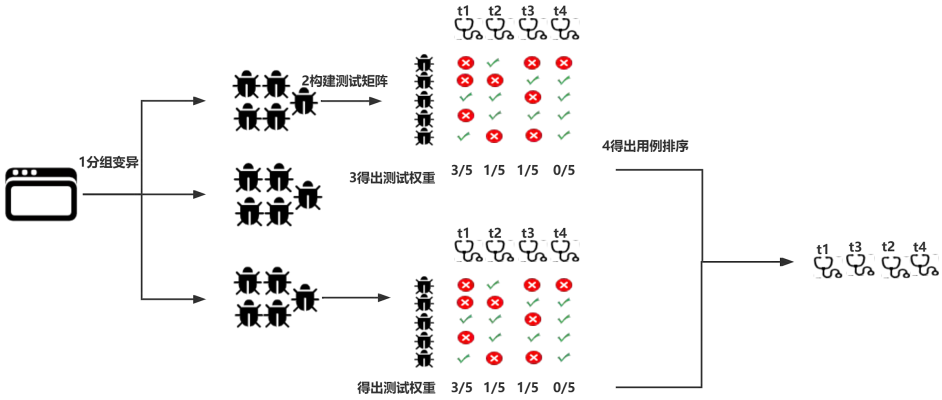


图 3.23: 最优排序设计

如图3.23所示是生成测试用例理想情况下排序的逻辑方案。首先对程序进行变异测试处理，获取程序的变异体，但是由于某些程序的变异体数量巨大，所以要对变异体数量进行裁剪。本系统设置最大值为 500，即如果变异数量超过 500，选取前 500 为该程序的变异体，若小于 500，则为实际的变异体数量。如果某个测试用例杀死了变异体，则记为 1，如果没有则记为 0，从而构建出变异矩阵。对变异矩阵，采用每 5 行为一组，每一组分别获取测试用例权，即每次迭代寻找每一组中测试用例杀死变异体最多的那一个，然后通过额外贪心算法依次获取测试用例权重。最后，将所有的权重进行累计求和求出，按照从大到小的顺序求出测试用力的排序。其中，如果在排序过程中，遇到杀死变异体数量相同的测试用例，则随机选择一个作为本次迭代输入的测试用例。

总体来说，需要将源码分为两部分，一部分作为训练集，一部分作为测试集。训练集部分放入模型中进行训练，获取到较为理想的模型后，使用测试集数据进行测试。最后使用 APFD 指标对模型的效果进行评估。

3.7 排序评估模块设计

3.7.1 架构设计

如图3.24为评估模型架构设计图。系统在该模块的初始输入为程序的源代码项目文件，通过将项目文件进行解析，获取其覆盖矩阵和关系图。图中 CRCP 为已经完成训练的模型部分，可直接输入关系图获取该项目推荐的测试用例排序。再者，系统可以根据关系图或覆盖矩阵，获取项目按照全局贪心算法和额外贪心算法所产生的测试用例排序，以上排序也即我们要评估的对象。

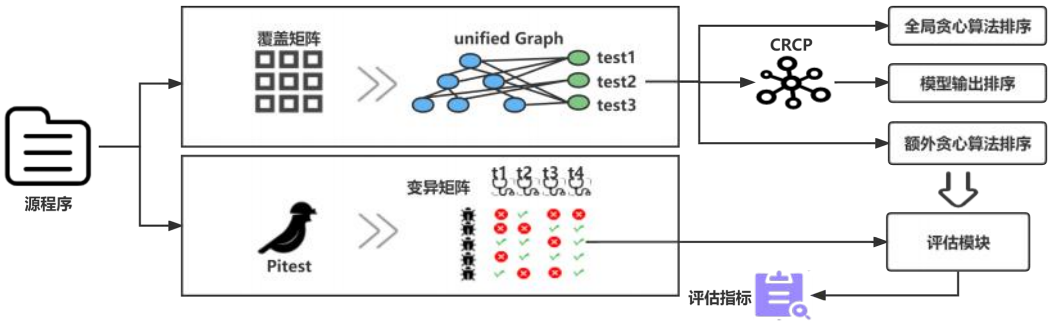


图 3.24: 评估模型架构设计图

随后程序会根据 PITest 工具产生变异矩阵，系统根据测试用例杀死变异体的情况创建变异矩阵，其中横坐标为测试用例编号，纵坐标为变异体编号。变异矩阵获取完成后将其输入到评估模块，评估模块会根据输入的变异矩阵与测试用例排序获取排序的 APFD 值进行返回。

3.7.2 详细设计

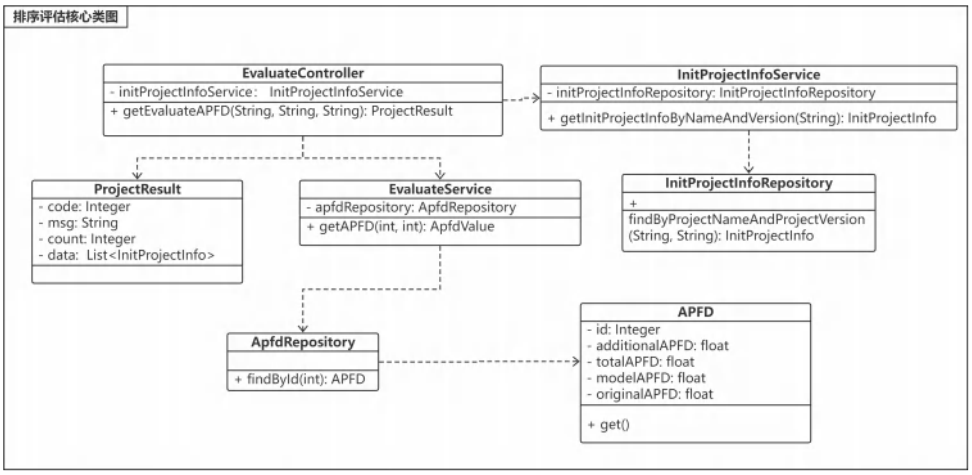


图 3.25: 评估模块核心类图

如图3.25为评估模块核心类图，主要展示了评估模块的核心代码设计。在系统层面则是用户在评估界面，选择相应的项目后，点击“评估”按钮触发。EvaluateController 主要处理前端传过来的评估请求，主要的获取 APFD 的逻辑封装在 EvaluateService 中。InitProjectInfoService 和 InitProjectInfoRepository 的主要功能在 3.4.4 章节中已经说明，在这不再赘述。EvaluateService 通过调用 getAPFD 方法获取某项目某种算法的排序的 APFD，这些信息都封装在 APFD 类中。最后将前端需要的信息封装在 ProjectResult 类中返回给前端。

3.8 本章小结

本章主要对基于覆盖表示学习的测试用例排序系统的需求进行了分析与阐述，包括功能需求、非功能需求等。根据需求分析对系统的框架设计进行了技术选择，并对系统的总体框架进行了分析，从 4+1 视图和模块划分等方面对系统的总体架构进行了详细阐述。系统主要分为了关系图生成模块、测试用例排序模块、图神经网络模块以及排序评估模块。对于每个模块，分别从架构设计、流程设计、类图体系结构以及数据库设计等方面进行了详细的说明。

第四章　基于覆盖表示学习的测试用例排序系统的实现

4.1　关系表示图实现

4.1.1　语句关系图生成实现

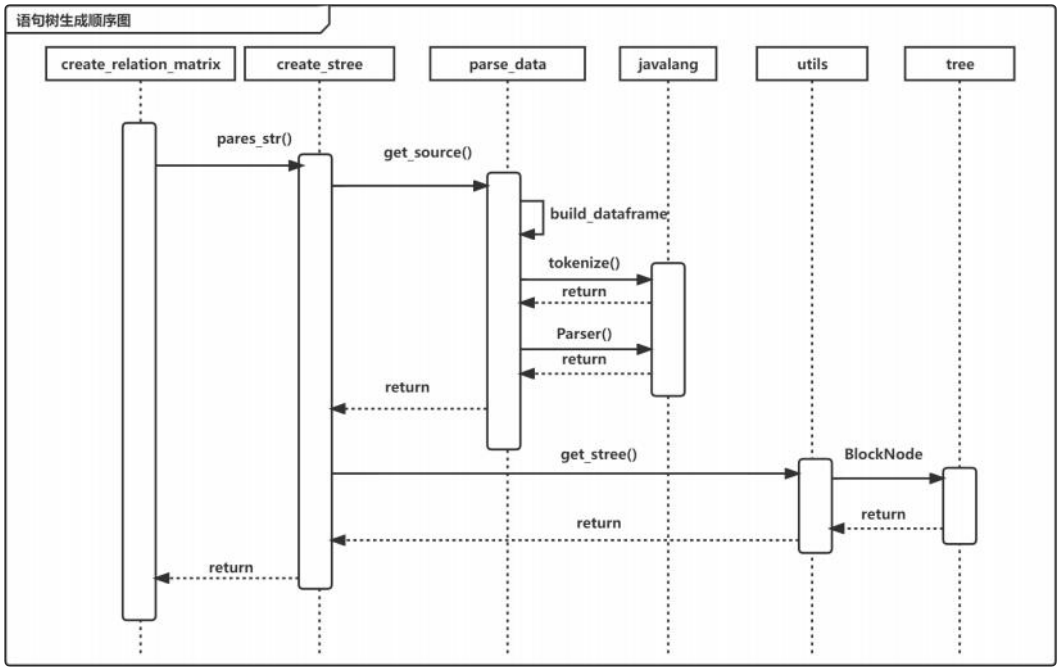


图 4.1: 语句树生成顺序图

语句生成模块执行顺序如图4.1所示，create_relation_matrix 是处理生成语句关系以及测试用例与代码覆盖关系的流程控制模块，该模块调用 create_stree 模块内容创建语句粒度关系树，create_stree 模块是生成语句粒度树的入口模块，其调用 parse_data 模块来对传入数据做二次处理。因为有些语句不是一个完整的方法，可能是 static 代码块，但是这一部分也是非常关键的信息，所以会对 static 代码块做类方法处理，会自动生成方法名称，将其当成方法输入到处理模块。同时，parse_data 模块也会调用 Javalang 方法的 tokenize 和 Parser 方法，将处理好的字符串类型的数据处理为 AST，保存在 dataframe 中返回。

语句粒度树的生成是数据处理的关键部分，是后面获得行与行代码关系的重要步骤。其主要实现是根据节点的类型来进行截断并处理。首先取到当前节点 token，并根据 token 类型进行处理。因为涉及多个递归过程，所以要制定处

理递归的逻辑。原则是先将数据放入 block_sel 和 relation 中，再将当前节点 id 和父节点 id 传入下一次迭代中。

再者是对不同情况的处理逻辑：

如果是 “MethodDeclaration”，“ConstructorDeclaration”，这说明是方法的声明行，是方法的开始，记录当前节点，并获取当前节点 node_id，将当前节点信息放入 block_seqz 中，后遍历其孩子节点，获取孩子节点 child_id，将孩子节点信息放入 block_seqz 中，将父子关系放入 relation 中，如果孩子节点是 “LocalVariableDeclaration” 则跳过，因为 “LocalVariableDeclaration” 是声明类型，不存我们所认为的可以继续分离的孩子节点，也就是代码行，如果孩子节点是其他类型，说明还有其他孩子节点，则继续进行迭代。

关键代码信息如下：

```
def get_stree(node, block_seq, relations, parent_id, present_id):
    name, children = get_token(node), get_children(node)
    ...
    # 如果该行是方法声明的行
    if name in ['MethodDeclaration', 'ConstructorDeclaration']:
        node_id = get_node_id(node)
        block_seq.append((node_id, tree_to_index(block_node(node))[0]))
        body = node.body
        for child in body:
            child_id = get_node_id(child)
            block_seq.append((child_id, tree_to_index(block_node(child))[0])
    )
            relations.append((node_id, child_id))
            # stopToken定义了没有孩子节点的类型信息
            if get_token(child) in stop_token:
                pass
            else:
                get_stree(child, block_seq, relations, node_id, child_id)
    ...
```

代码 4.1: 方法声明语句获取关键代码

如果是 “SwitchStatement”, “IfStatement”, “ForStatement”, “WhileStatement”, “DoStatement” 等节点，说明当前节点是循环体或者判断结构的开始，一定会有孩子节点信息，所以要进行递归调用。首先获取当前节点 node_id，遍历其孩子节点。获取孩子节点 child_id，并将孩子节点信息放入 block_seq，将父子关系放入 relation 中。如果孩子节点是 “LocalVariableDeclaration” 说明该节点没有孩子节点，则跳过。如果孩子节点是其他类型，说明还有其他孩子节点，则将继续进行迭代，直到没有其他孩子节点为止。

关键代码信息如下:

```
# login中定义'SwitchStatement','IfStatement'等信息
if name in logic:
    node_id = present_id
    for child in children[1:]:
        child_id = get_node_id(child)
        block_seq.append((child_id, tree_to_index(block_node(child))[0]))
        relations.append((node_id, child_id))
        if get_token(child) in stop_token:
            pass
        else:
            if get_token(child) == 'BlockStatement':
                get_stree(child, block_seq, relations, node_id, node_id)
            else:
                get_stree(child, block_seq, relations, node_id, child_id)
...
```

代码 4.2: 表达式语句获取关键代码

如果是 “BlockStatement” 节点, 则说明是代码中的 “{” 在 Javalang 中, 会把代码中的 “{” 算作为单独的一部分, 而对于我们的系统来说, 这一部分元素信息是不希望记录的, 所以要删除掉已经记录的信息, 分别删除 block_seq 和 relation 中的上一个元素, 将当前节点的 id 设置为上个该节点的父节点的 id, node_id, 遍历孩子节点, 获取孩子节点 child_id, 将孩子节点放入 block_seq 中, 将父子关系放入 relation 中, 如果孩子节点是 “LocalVariableDeclaration”, 则跳过, 如果孩子节点是其他类型, 说明还有其他孩子节点, 则继续进行迭代。

关键代码信息如下:

```
if name =='BlockStatement':
    block_seq.pop()
    relations.pop()
    node_id = parent_id
    for child in node.statements:
        child_id = get_node_id(child)
        block_seq.append((child_id, tree_to_index(block_node(child))[0]))
        relations.append((node_id, child_id))
        if get_token(child) in stop_token:
            pass
        else:
            get_stree(child, block_seq, relations, node_id, child_id)
...
```

代码 4.3: 代码块处理关键代码

而对于 try...catch... 结构也要单独处理，因为 catch 的数量不确定，所以可能会导致结构的不同，同时有些代码结构中会有 finally 块而有的没有，所以也要进行判断。如果是 “TryStatement”，获取当前节点 node_id，遍历其孩子节点，注意孩子节点分两个集合遍历，如果孩子节点是 “LocalVariableDeclaration”，则跳过，如果孩子节点是其他类型，说明还有其他孩子节点，则继续进行迭代。

例如以下代码：

```
public int demo(int a, int b){
    if(a == 0){
        return b;
    }
    while(b != 0){
        if(a > b){
            a = a - b; // b = a-b
        }else{
            b = b - a;
        }
    }
    return a;
}
```

代码 4.4: 测试示例代码

最终获得的语句关系输出结果如下：

```
node:[(1, “MethodDeclaration”),
      (2, “IfStatement”),
      (3, “ReturnStatement”),
      (5, “WhileStatement”),
      (6, “IfStatement”),
      (7, “StatementExpression”),
      (9, “StatementExpression”),
      (12, “ReturnStatement”)]

relation:[(1, 2), (2, 3), (1, 5), (5, 6), (6, 7), (6, 9), (1, 12)]
```

代码 4.5: 结果展示

4.1.2 测试覆盖图生成实现

测试用例覆盖图主要是获取测试用例的覆盖矩阵。测试用例与代码覆盖关系的获取主要通过解析 OpenClover 生成的报告。OpenClover 会根据每个源码文件生成对应的 html 文件，如图2.4所示。某个源码文件的 OpenClover 报告对每行

代码的覆盖情况进行了展示，同时也展示了覆盖该行的测试用例名称，同时对某个方法的代码块进行了解析。

首先解析 OpenClover 报告获取所有测试用例名称与所有源码的类名，其中最重要的是建立全图的唯一序号映射关系，也即一个存放序号与结点关系的对应字典，主要用于在全局区分每行代码和每个测试用例。获取完映射关系字典后，获取所有的有效行，有效行为所有被覆盖的行。因为一个项目代码庞大，如果所有的行都计算进入覆盖矩阵，其中会存在大量的 0，这是由于源代码中存在许多的空行、注释行已经没有覆盖的行。

最后测试用例与覆盖代码的关系，主要通过解析 OpenClover 中的 js 文件获得的，OpenClover 会将代码与测试用例的覆盖关系通过与源码类类名相同的 js 文件存储。获取完成后将关系转换成覆盖矩阵即可。

其中核心部分构建测试方法和序号之间的映射关系伪代码如下所示：

```
def map_test():
    ...
    test_id_map = {}
    # 遍历测试报告文件，获取报告中测试用例与序号的关系
    for test_file in test_file_list:
        # 解析报告，获取所有的测试的td标签
        td_list = get_test_td()
        for td_item in td_list:
            # 从标签中获取测试用例序号
            test_Id = get_test_id()
            ...
            # 建立报告中的序号与全局序号字典
            test_Id_map[test_id] = valid_line_num
            valid_line_num = valid_line_num + 1
    ...
    return test_id_map
```

代码 4.6: 构建测试方法和序号之间的映射关系核心实现伪代码

4.1.3 总体关系图生成实现

总体关系图主要是将以上两个关系进行合并所获得，其中关键步骤主要对全局的序号进行唯一化处理，也即每个测试用例与代码结点的序号不同出现重复的情况，一个序号可以唯一定位一个结点。再者，要对数据进行有效的持久化操作，并保证数据的一致性。获取的数据通过字典的形式存储在 pkl 文件中，然后将 pkl 文件的地址保存到数据库中。

其中总体关系图生成核心代码如下所示：

```
def anaMain(project):

    # 获取所有测试用例
    test_file_list = get_test_file(xml_file_path, root)
    # 获取所有源文件类名，并对每个方法进行解析
    src_file_list = get_node_info(xml_file_path, root)
    # 统计测试用例，并为其分配唯一id
    test_id_map = map_test(test_file_list, xml_file_path)
    # 获取整体关系矩阵
    relation = fill_matrix_state(src_file_list, xml_file_path, test_id_map)
    ...
    # 保存数据
    data_proess_and_save(relation)
```

代码 4.7: 总体关系图生成核心代码

数据持久化操作接口及其主要内容如下表4.1所示：

表 4.1: data.pkl 数据内容

字段	类型	含义
relationMatrix	list	关系的元组集合
methodAndTestTypeMap	list	方法与测试用例类型映射数字列表
methodAndTestIds	list	方法与测试用例序号列表
codeTypeMap	list	代码类型的数字映射列表
codeIds	list	代码的序号列表
matrix	coo_matrix	关系矩阵
methodNum	int	方法数
codeNum	int	代码数
testNum	int	测试用例数
coverage_weight	int[]	测试用例根据额外贪心算法获取的权重数

如表4.1所示，系统会对测试用例、方法和代码的类型到数字的映射进行持久化操作，这是因为模型训练是需要类型属性，但是会对结点的类型数据进行数字化操作，也就是给所有属性进行编号。为了减少存储的数据量，系统只对结点的类型进行保存以及对每个项目的映射的数字进行保存，这就大大减小了保存的数据大小。

类型与数字映射关系如下表4.2所示：

表 4.2: 结点类型数字映射关系表

字段	数字	字段	数字	字段	数字
UnKnow	1	BreakStatement	14	ForControl	27
Test	2	ContinueStatement	15	EnhancedForControl	28
MethodDeclaration	3	ReturnStatement	16	Expression	29
InterfaceDeclaration	4	ThrowStatement	17	Assignment	30
ConstructorDeclaration	5	SynchronizedStatement	18	TernaryExpression	31
VariableDeclaration	6	TryStatement	19	BinaryOperation	32
LocalVariableDeclaration	7	SwitchStatement	20	MethodInvocation	33
FormalParameter	8	BlockStatement	21	Statement	34
IfStatement	9	StatementExpression	22	Literal	35
WhileStatement	10	TryResource	23	ClassDeclaration	36
DoStatement	11	CatchClause	24		
ForStatement	12	CatchClauseParameter	25		
AssertStatement	13	SwitchStatementCase	26		

4.2 测试用例排序实现

4.2.1 理想排序模块实现

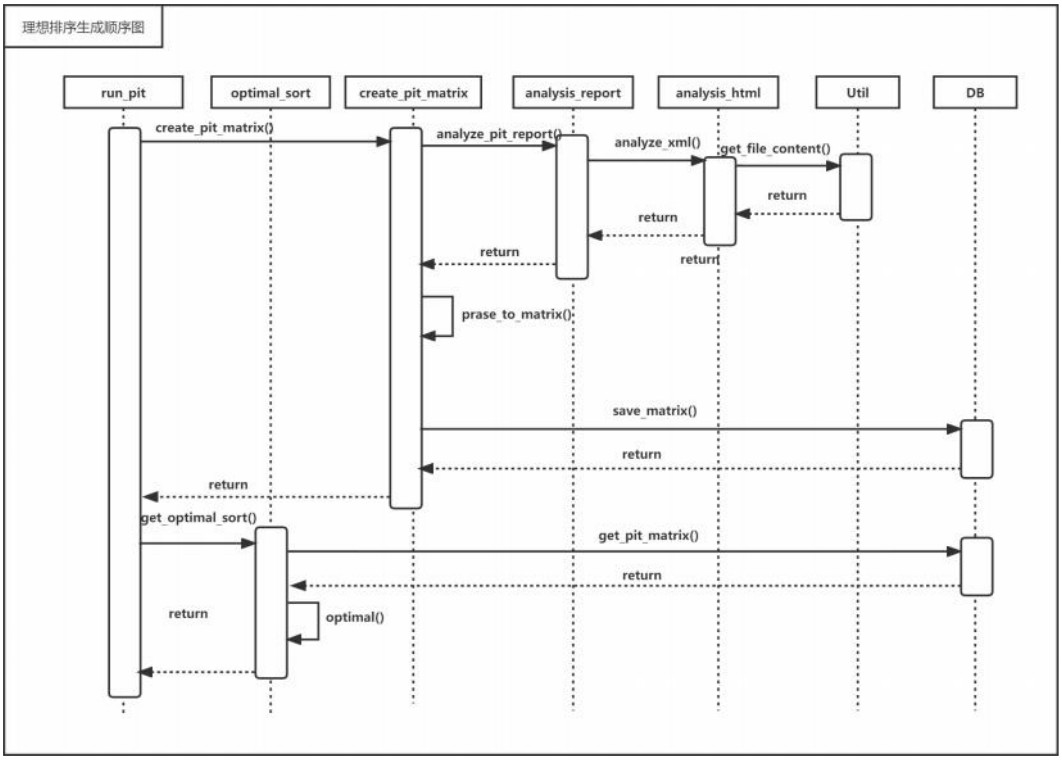


图 4.2: 理想排序生成顺序图

如图4.2为理想排序生成顺序图。理想排序是输入到模型中进行训练的排序集合，主要作为模型的 Ground Truth。理想情况下的测试用例排序是根据变异矩阵获取的。首先通过 Pitest 工具对初始项目进行变异处理，获取变异报告，通过解析变异报告获取变异矩阵，通过额外贪心算法计算测试用例的权重，从而根据测试用例权重进行由高到低的排序。最后进行数据的持久化操作，保存到 weight.pkl 文件中。

获取理想排序最关键的是获取每个测试用例的权重值，按照什么样的算法进行权重的计算非常关键。起初，每五个变异体为一组，对一百组变异矩阵进行分别排序，得到 100 组权重值，获得平均值作为每个测试用例的权重。但是这种算法获取的权重值放入模型中训练效果并不好。随后对算法进行了调整，直接在 500 个（如果变异体数量小于 500，则使用原变异矩阵，如果大于 500，则随机矩阵后选取前 500）使用额外贪心算法获取排序权重，当多个用例覆盖相同的用例时，选取第一个为本次迭代的测试用例。该算法获取的权重放入模型训练效果明显好于第一种。

代码实现思路为先通过一次遍历获取每个测试用例覆盖的矩阵，存储在字典中，每次获取只需要根据 test_id 对应的数组长度大小选出对应的 test，然后将剩下的 test 与当前删除的 test 做差操作。以下为根据变异矩阵获取测试用例权重的核心代码：

```
def optimal(mutant_matrix):
    test_num = len(mutant_matrix[0])
    line_num = len(mutant_matrix)
    test_line_dict = {}
    test_weight = [0 for i in range(test_num)]
    for i in range(test_num):
        lines = []
        test_line_dict[i] = lines
    # 遍历矩阵，列是测试用例id
    for i in range(line_num):
        for j in range(test_num):
            if matrix[i][j] == 1:
                test_line_dict[j] = test_line_dict[j].append(i)
    test_sort = []
    while len(test_line_dict) != 0:
        # 获取覆盖做多行的test
        max_len = 0
        del_index = 0
        for test, lines in test_line_dict.items():
            if len(lines) > max_len:
                max_len = len(lines)
```

```
                del_index = test
        if max_len == 0:
            for test, lines in test_line_dict.items():
                test_sort.append(test)
                test_weight[test] = max_len
            return test_weight, test_sort
        # 将其test与该test做差
        test_weight[del_index] = max_len
        test_lines = test_line_dict[del_index]
        test_sort.append(del_index)
        test_line_dict.pop(del_index)
        for test, lines in test_line_dict.items():
            # 去掉test_lines中有的行
            test_line_dict[test] = set(lines).difference(set(test_lines))
    return test_weight, test_sort
```

代码 4.8: 理想排序生成核心代码

4.2.2 其他排序模块实现

系统使用的排序还有额外贪心算法和全局贪心算法。额外贪心算法只是将 4.2.1 中理想排序生成核心代码中参数由变异矩阵（mutant_matrix）换成覆盖矩阵（coverage_matrix) 即可，所以在这里不再赘述。

以下为全局贪心算法排序的核心代码，主要是根据测试用例覆盖行数的多少进行排序。

```
def total_sort(coverage_matrix):
    test_nums = len(coverage_matrix[0])
    test_dict = {}
    for i in range(test_nums):
        test_dict[i] = 0
    for line in range(len(coverage_matrix)):
        for test in range(len(coverage_matrix[line])):
            if coverage_matrix[line][test] == 1:
                test_dict[test] += 1
    test_weight = [val for val in test_dict.values()]
    test_sort_list = sorted(test_dict.items(), key=lambda x: x[1], reverse=
    True)
    total = []
    for test_id, weight in test_sort_list:
        total.append(test_id)
    return test_weight, total
```

代码 4.9: 理想排序生成核心代码

4.3 图神经网络训练实现

4.3.1 数据处理模块实现

模型训练之前对数据需要进行预处理工作，将数据加载进内存中，并将数据进行格式化处理。模型需要使用语句级别的代码关系结构，并将测试用例结点放入关系结构中一同输入，但是要区分其中的数据结点。对于结点属性，方法和代码结点将 AST 抽象语法数中的 token 值作为结点属性，而测试用例结点数据相对较多，主要有每个测试用例的执行时间和测试用例在覆盖矩阵中通过额外贪心算法获取的测试用例权重。其中也尝试将测试结点的 Jaccard 距离以及每个测试用例单位时间内的覆盖数量作为属性，但是发现效果并不好，遂将这两个属性去掉了。

数据预处理模块核心处理流程代码如下：

```
class SumDataset(data.Dataset):
    ...
    def get_data_list(self, type):
        # 初始化需要的7个参数
        ...
        # 从文件中加载所需要的数据
        ...
        # 对数据格式化处理，并放入数据集data中
        for i in range(len(allProjectTestNode)):
            allProjectTestNode[i] = self.pad_seq(allProjectTestNode[i], self
    .test_len)
        data.append(allProjectTestNode)
        for i in range(len(allProjectMethodNode)):
            allProjectMethodNode[i] = self.pad_seq(allProjectMethodNode[i],
    self.method_len)
        data.append(allProjectMethodNode)
        for i in range(len(allProjectLineNode)):
            allProjectLineNode[i] = self.pad_seq(allProjectLineNode[i], self
    .line_len)
        data.append(allProjectLineNode)
        # 对其他数据做同样的处理
        ...
        return data
```

代码 4.10: 模型数据预处理模块核心代码

为了加快梯度下降的速度，其中时间属性进行了归一化处理，将时间属性值规定在 $[0, 1]$ 之间。再者，将不同项目的同一属性作为一个集合，将不同属性

存放到不同集合作为模型的参数。但是为了保证每个项目相同属性集合长度相同，需要在每个项目属性加载后记录每个属性的长度，找到所有项目每个属性的最大值作为所有属性集合的长度，再对每个项目的属性集合的大小进行扩增，后边少的部分填充 0。

4.3.2 模型训练模块实现

对数据处理完成后，接下来是对数据的训练，获取推荐的测试用例排序。模型的选取主要借鉴了 Boosting Coverage-Based Fault Localization via Graph-Based Representation Learning[1] 论文中所提出的 Grace 神经网络模型。Grace 是一种门控图神经网络模型，是 GNN 的一种变体，它包括门控单元和长短时记忆，以保持长期依赖性。因此，在 Grace 中，设计了一个 GGNN 模型来学习统一覆盖图中的关键特征，并对图中的可疑代码节点进行排序。

模型的核心代码如下：

```
class CRCP(nn.Module):
    def __init__(self, args):
        self.transformerBlocks = nn.ModuleList(
            [TransformerBlock(self.embedding_size, 8, self.
    feed_forward_hidden, 0.1) for _ in range(5)])
        self.token_embedding = nn.Embedding(args.Nl_Vocsize, self.
    embedding_size)
        self.token_embedding2 = nn.Embedding(args.Nl_Vocsize, self.
    embedding_size - 2)
        ...
    def forward(self, test_node, method_node, line_node, test_time,
    test_coverage_weight, test_weight, matrix):
        test_mask = torch.eq(test_node, 2)
        test_em = self.token_embedding2(test_node.to(torch.int64))
        test_em = torch.cat([test_em, test_time.unsqueeze(-1).float(),
    test_coverage_weight.unsqueeze(-1).float()], dim=-1)
        method_em = self.token_embedding(method_node.to(torch.int64))
        line_em = self.token_embedding(line_node.to(torch.int64))
        x = torch.cat([test_em, method_em, line_em], dim=1)
        for trans in self.transformerBlocks:
            x = trans.forward(x, test_mask, matrix)
        x = x[:, :test_node.size(1)]
        res_softmax = F.softmax(self.resLinear2(x).squeeze(-1).masked_fill(
    test_mask == 0, -1e9), dim=-1)
        loss = -torch.log(res_softmax.clamp(min=1e-10, max=1)) * test_weight
        loss = loss.sum(dim=-1)
        return loss, res_softmax, x
```

代码 4.11: 模型核心代码

本系统中的 CRCP 模型是在 Grace 模型基础上进行的调整，以测试用例排序作为模型的输出内容，以测试用例结点的属性作为关键属性。单词嵌入层首先对属性序列进行编码转换为属性矩阵。代码根节点和非根节点只有一个属性，可以直接嵌入到 d 维数向量。对于具有 3 个属性的测试用例节点，我们首先将 AST 节点类型嵌入到 d−3 维，然后将其与测试相关的属性连接成 d 维向量。

再者，GGNN 层通过五次迭代应用 GGNN 层。在第 t 次迭代中，对于每个节点，通过合并来自其相邻节点和来自先前迭代的信息来更新其当前状态。模型中，在 GGNN 中通过利用 LSTM 中的输入门和忘记门来控制单元状态的传播，实现了选通机制。特别是，忘记门决定要从单元状态中排除哪些信息，输入门从当前输入中保留哪些信息，根据新的和遗忘的信息，结点单元状态可以进行更新。为了避免梯度消失的问题，模型利用两个子层之间的残余连接和正则化。所有 GGNN 迭代计算后的输出进一步送到线性变换层，然后进行 softmax 激活。

最后，就是对模型训练时一些参数的控制和调整了，主要包括学习率（lr）、随机种子（seed）以及一次训练所选取的样本数（batch_size）等变量的控制。不同参数可以对模型的训练效果及训练的速度产生较大的影响。

训练控制核心伪代码如下所示：

```
def train():
    设置随机种子
    加载训练集合
    加载验证集合
    optimizer = ScheduledOptim(optim.Adam(model.parameters(), lr=args.lr),
    args.embedding_size, 4000)
    model = model.train()
    for epoch in range(train_epoch):
        for idx, dBatch in enumerate(train_set.Get_Train(args.batch_size)):
            if idx == 1:
                # 使用验证集验证模型效果
                ...
            model = model.train()
            for i in range(len(dBatch)):
                dBatch[i] = gVar(dBatch[i])
            # 输入模型训练，返回损失函数
            loss, pre, _ = model(dBatch[0], dBatch[1],..,dBatch[7])
            optimizer.zero_grad()
            loss = loss.mean()
            loss.backward()
            optimizer.step_and_update_lr()
    ...
```

代码 4.12: 训练控制核心代码

4.4 人机交互模块实现

4.4.1 人机交互流程实现

人机交互模块主要实现了用户与系统交互的主要功能，其中包括项目的上传、项目的查看、项目的解析、测试用例排序的获取以及评估测试用例等功能。人机交互顺序图如图4.3所示。图中用户共产生了七次交互动作，也是系统使用的主要过程。

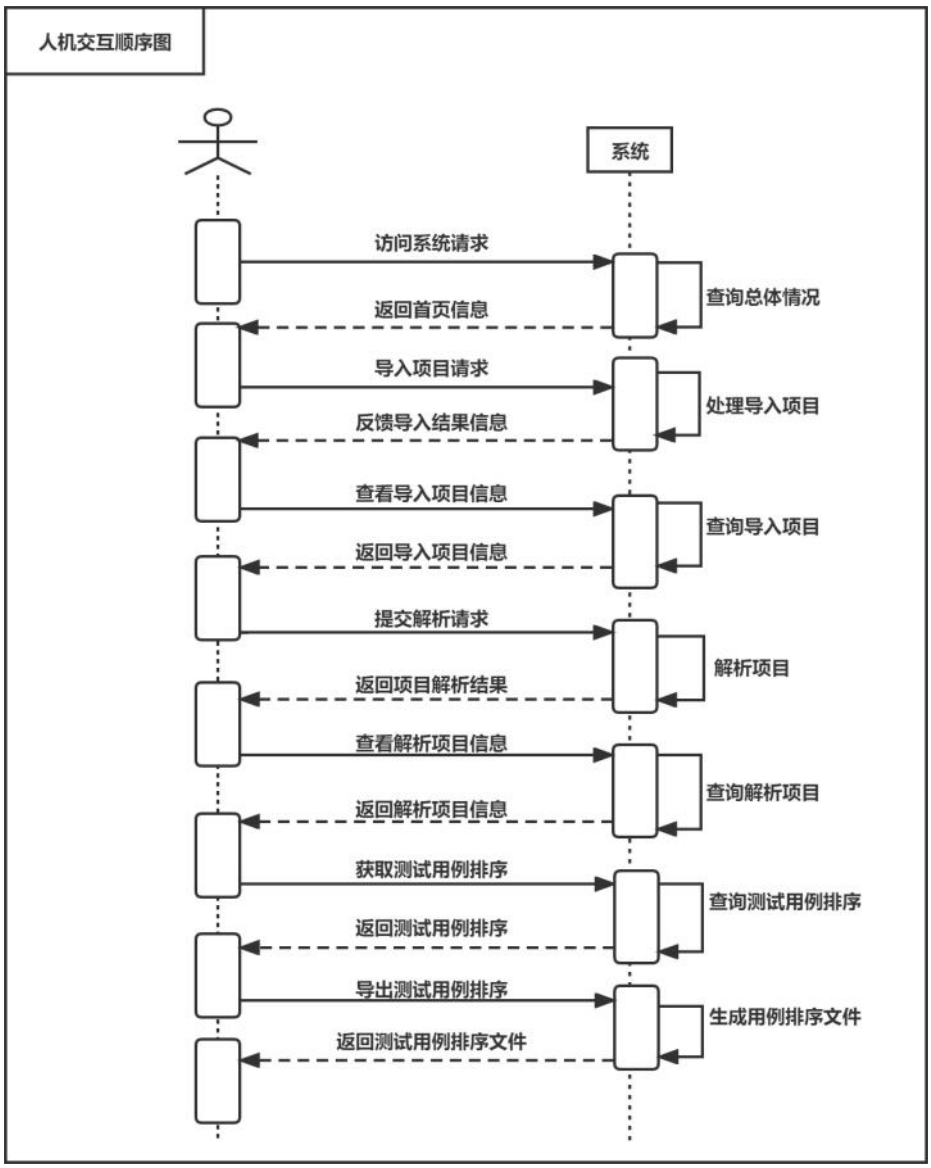


图 4.3: 人机交互顺序图

第一次交互动作由用户发起，用户发起访问系统的请求，系统会查询系统中项目导入以及解析项目的整体情况，以首页的形式展现给用户。第二次交互

动作为用户的提交项目的请求，系统会以表单的形式接收用户提交的项目及项目信息，保存到指定位置，然后返回用户导入结果的反馈信息。第三次交互为用户发起查看导入项目的信息请求，系统会查询所有已经导入项目的信息反馈给用户。第四次为用户解析项目的请求，这也为本系统的核心功能点。系统接收到请求后对需要解析的项目进行解析，并会将解析的结果反馈给用户。解析完成后用户可以查看已经完成解析的项目列表，也即第五次请求。第六次交互中用户发起查看测试用例排序的请求，系统会根据解析项目的情况，返回给用户其需要的测试用例排序。最后，如果用户认为某排序是其理想的排序结果，可以发起导出排序请求，系统会将排序以 Excel 文件的方式导出。

4.4.2 项目解析功能实现

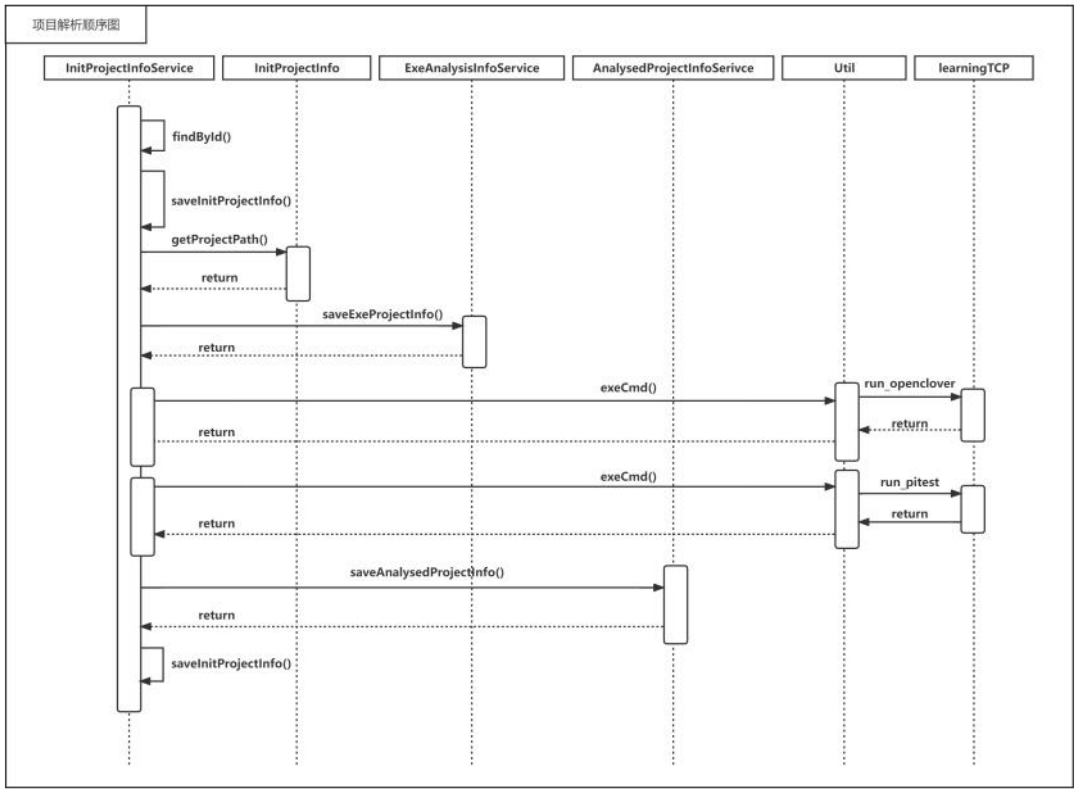


图 4.4: 项目解析顺序图

如图4.4为项目解析顺序图。解析项目是用户在初始项目列表页面点击“解析”按钮触发的。由 InitProjectController 接收到请求后，交给 initProjectInfoService 进行完成。首先会根据项目的唯一标识查询项目的地址，获取项目的路径，然后记录项目的解析状态，通过 Util 模块调用 learningTCP 模块的处理程序的脚本机型获取数据。在每个数据获取完成的结点，都将完成的过程同步到数据库中。

代码4.13为项目解析的核心代码：

```
public void analysisProject(String projectId) {
    // 记录项目解析状态
    InitProjectInfo initProjectInfo = initProjectInfoService.findById(String
    .valueOf(projectId));
    initProjectInfo.setProjectState(1);
    initProjectInfoService.saveInitProjectInfo(initProjectInfo);
    // 解析关系矩阵
    Util.exeCmd('run_openclover')
    // 变异处理
    Util.exeCmd('run_pitest')
    // 保存已解析项目状态
    analysedProjectInfoSerivce.saveAnalysedProjectInfo(analysedProjectInfo);
    ...
}
```

代码 4.13: 项目解析核心代码

4.4.3 排序获取功能实现

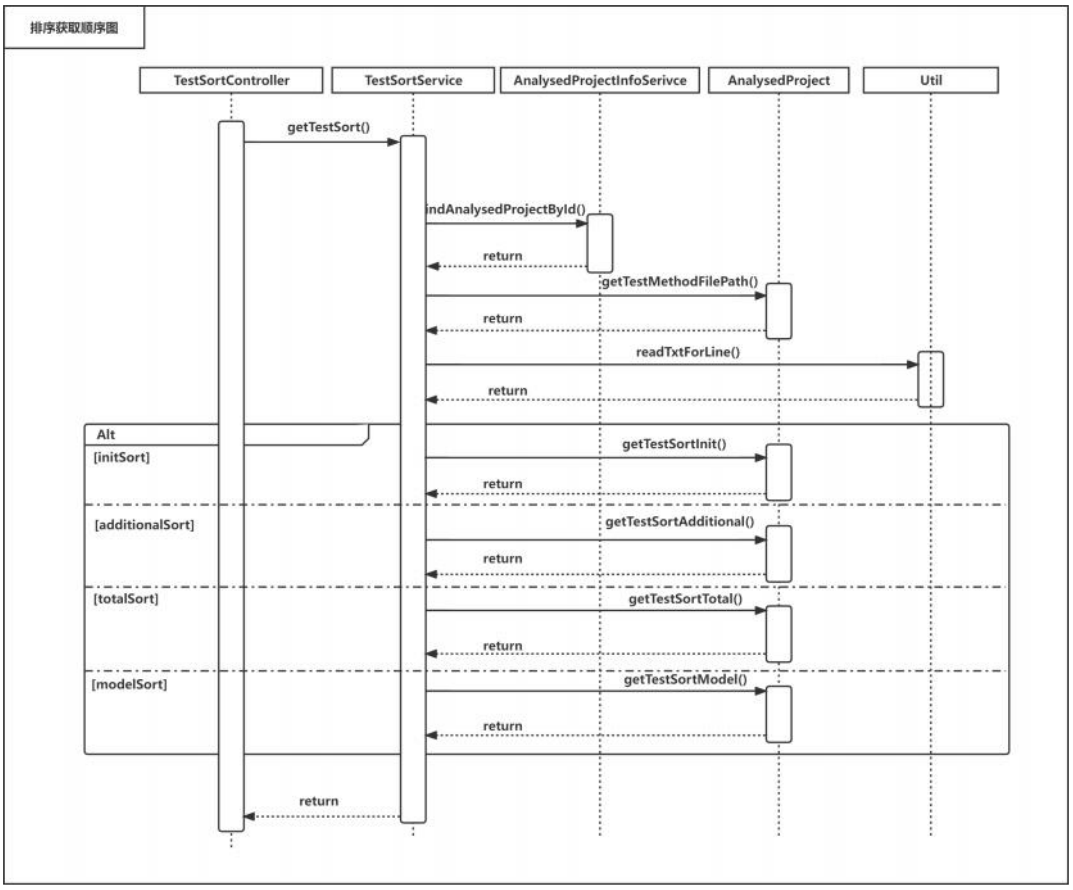


图 4.5: 排序获取顺序图

图4.5为排序获取顺序图。排序获取请求是在项目详情页面触发，用户点击不同算法排序获取按钮触发不同算法获取功能。其中，TestSortController 主要接收前端发送的获取排序请求，主要处理逻辑交由 TestSortService 类实现。TestSortService 先通过 AnalysedProjectInfoService 类获取项目信息，然后通过 AnalysedProject 获取项目的测试用例方法文件存放的地址。通过 Util 工具类读取测试用例方法名称以及唯一标识，最后根据前端不同算法排序的请求，调用自身的函数来获取不同的排序。最后返回给前端进行展示。

关键代码如代码4.14所示：

```
public List<TestSort> getTestSort(String id, String sortType){
    ...
    List<TestSort> testSort = new ArrayList<>();
    AnalysedProjectInfo analysedProject = analysedProjectInfoSerivce.
    findAnalysedProjectById(id);
    String tempFilePath = analysedProject.getTestMethodFilePath();
    Map<Integer, String> indexAndTestMethod = Util.readTxtForLine(
    tempFilePath);
    if(sortType == "0"){//默认排序
        testSort = initSort();
    }else if(sortType == "1"){// 额外贪心算法排序
        testSort = analysedProject.getTestSortAdditional();
    }else if(sortType == "2"){// 全局贪心算法排序
        testSort = analysedProject.getTestSortTotal();
        }
    }else{// 推荐算法排序
        testSort = analysedProject.getTestSortModel();
    }
    return testSort;
}
```

代码 4.14: 排序获取核心伪代码

4.4.4 排序评估功能实现

如图4.6为排序评估顺序图。评估排序主要是获取项目的 APFD 的过程，是在系统的评估节面，用户点击“评估”按钮触发。EvaluateController 接收到用户发出的评估排序请求后，会将处理逻辑交由 InitProjectInfoService 来处理。InitProjectInfoService 先调用 InitProjectInfoRepository 获取项目的信息，然后会调用 EvaluateService 的 getAPFD() 方法获取测试用例排序的 APFD，然后根据前端请求算法的 APFD 进行返回。

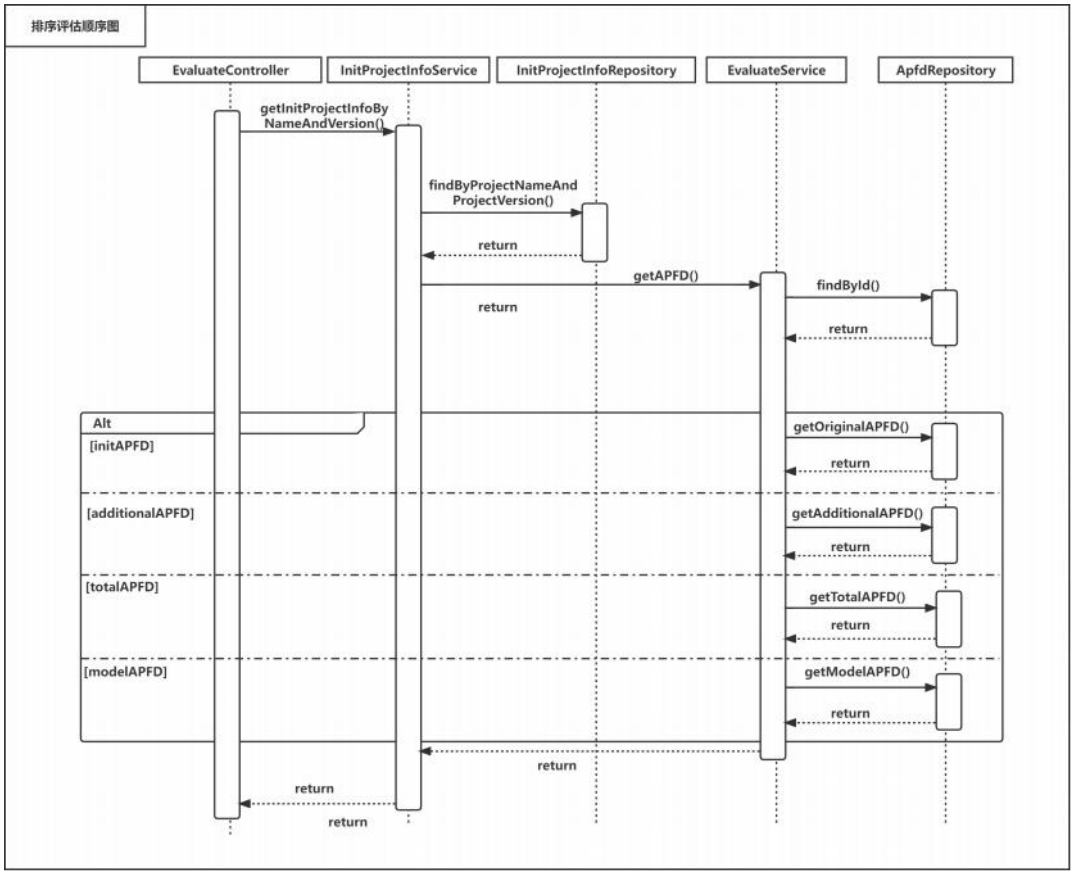


图 4.6: 排序评估顺序图

关键代码如代码4.15所示:

```
public ProjectResult getEvaluateAPFD(){
    ...
    if(projectId == null){
        InitProjectInfo initProjectInfo = initProjectInfoService.
    getInitProjectInfoByNameAndVersion(projectName, projectVersion);
        projectId = String.valueOf(initProjectInfo.getId());
    }
    ApfdValue apfd = evaluateService.getAPFD(projectId,SortType);
    return apfd;
}
```

代码 4.15: 排序评估核心代码

4.4.5 交互页面实现

如图4.7为系统首页页面，展示的内容是系统中项目的更新情况。其中最近上传项目列表模块展示了最近上传的项目列表，一般使用较为频繁的项目就是项目最近上传的项目，这个模块提示用户最近上传项目的情况。首页页面还包括已经完成解析项目的情况和正在解析项目的情况等。首页信息主要是给用户对系统中的项目有一个直观的认识。

图 4.7: 系统首页页面实现

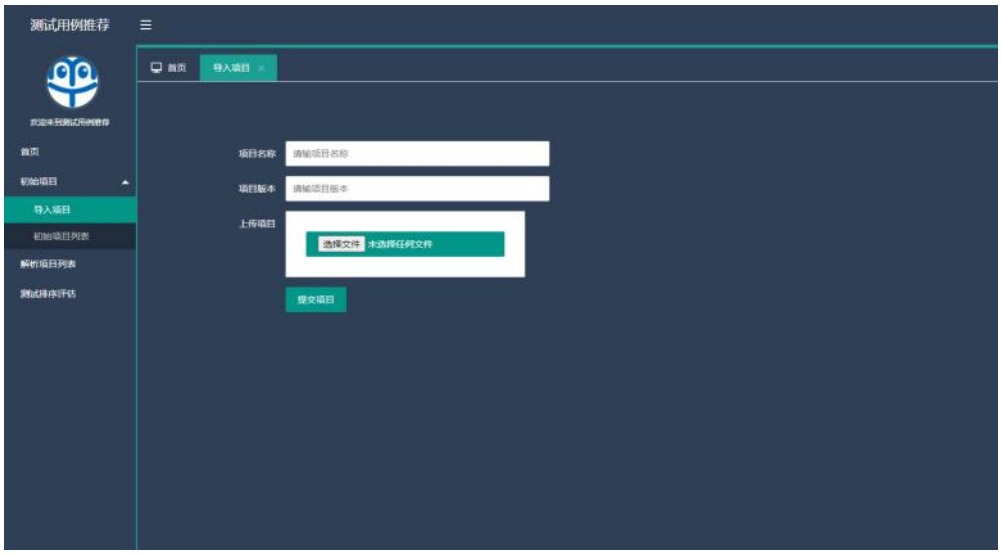


图 4.8: 系统项目导入页面实现

如图4.8为系统项目导入页面。功能较为简单，主要完成用户提交项目请求。

当提交项目完成后，系通过会给出提示提交是否成功。

如图4.9为系统初始项目列表页面，主要展示了系统中导入项目的情况，包括未解析项目、解析中项目以及完成解析的项目。本页面主要使用户对系统中所有项目有一个直观的认识，同时，支持用户对系统中的项目进行编辑、删除以及解析的操作。另外，该页面还展示了项目序号、项目名称以及项目版本等。

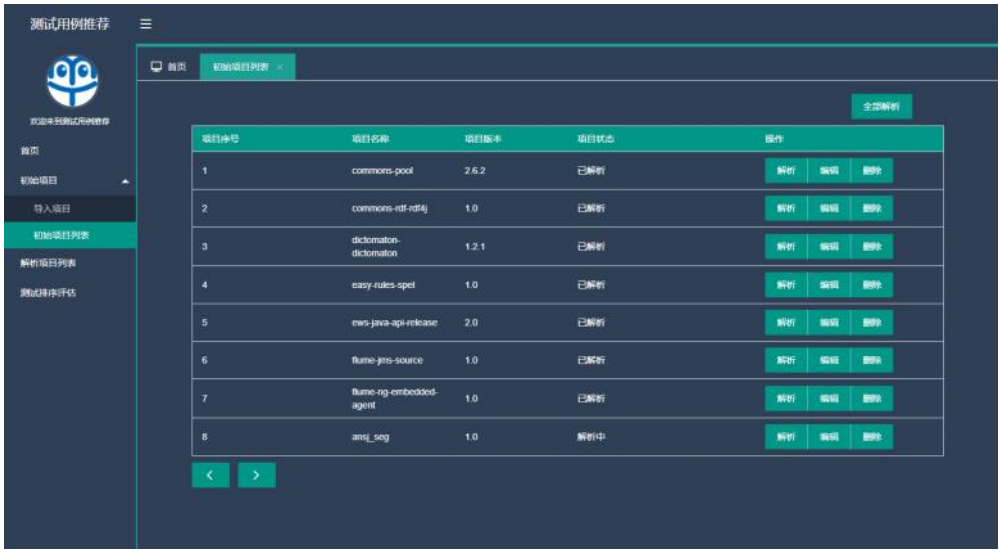


图 4.9: 系统初始项目列表页面实现

如图4.10为系统解析列表页面。主要展示了已经完成解析的项目基础信息，包括项目序号、项目名称、项目版本、项目的测试用例数量以及测试的执行之间等。同时，解析页面还提供展示项目详情的链接按钮，展示项目解析完成的详情信息。最后还包括对解析项目的删除和导出排序操作。

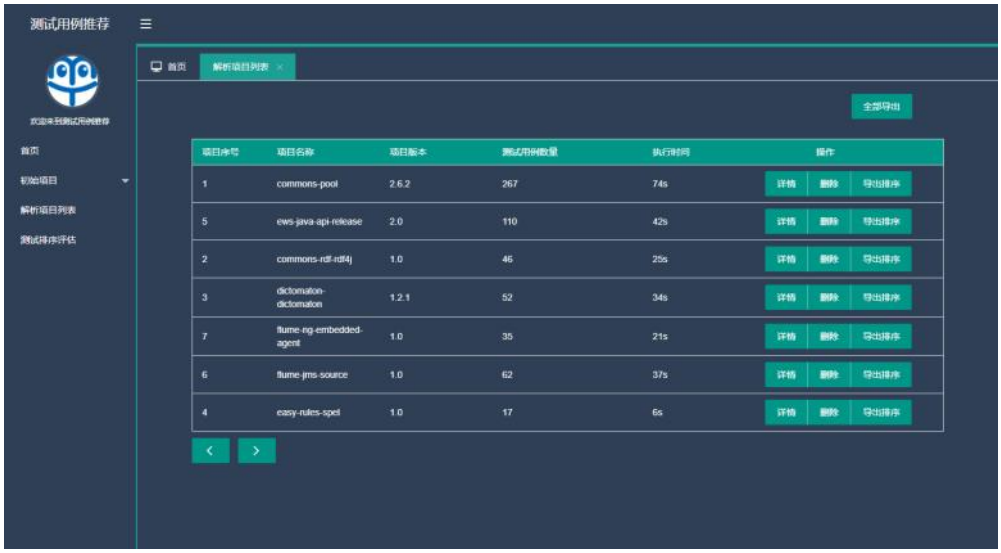


图 4.10: 系统解析列表页面实现

如图4.11为某已解析项目的详情页面。详情页面通过弹出窗口的形式进行展示，展示内容包括项目的上传时间、解析完成时间、测试用例数量等信息。同时在页面布局的左下方，有四个按钮，分别为原始测试用例排序、全局贪心算法排序、额外贪心算法排序以及 CRCP 模型排序，点击不同按钮后，会将不同的排序展示在右侧的表格中，如果用户想要下载该排序，可以点击右上角的“导出排序”按钮。

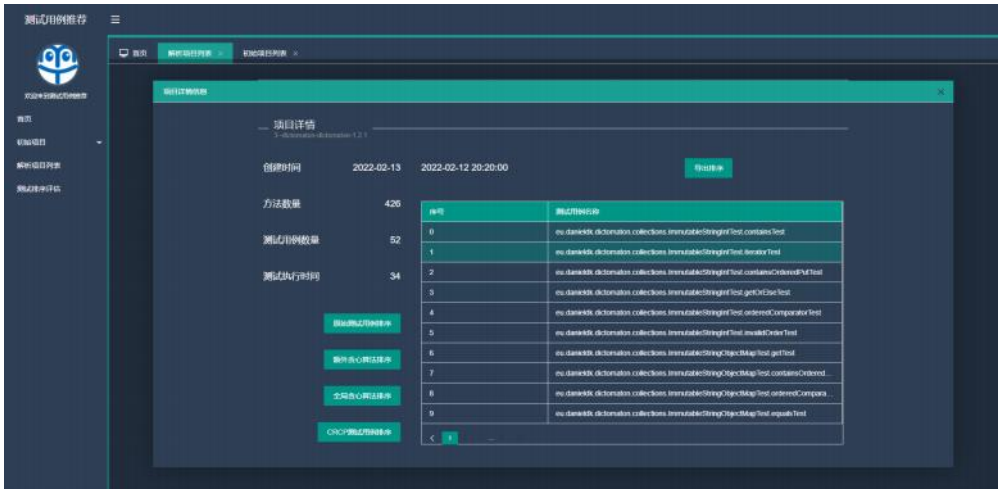


图 4.11: 项目详情页面实现

如图4.12为系统评估页面，该页面主要功能较为简单，即展示某项目中某排序的 APFD 值。

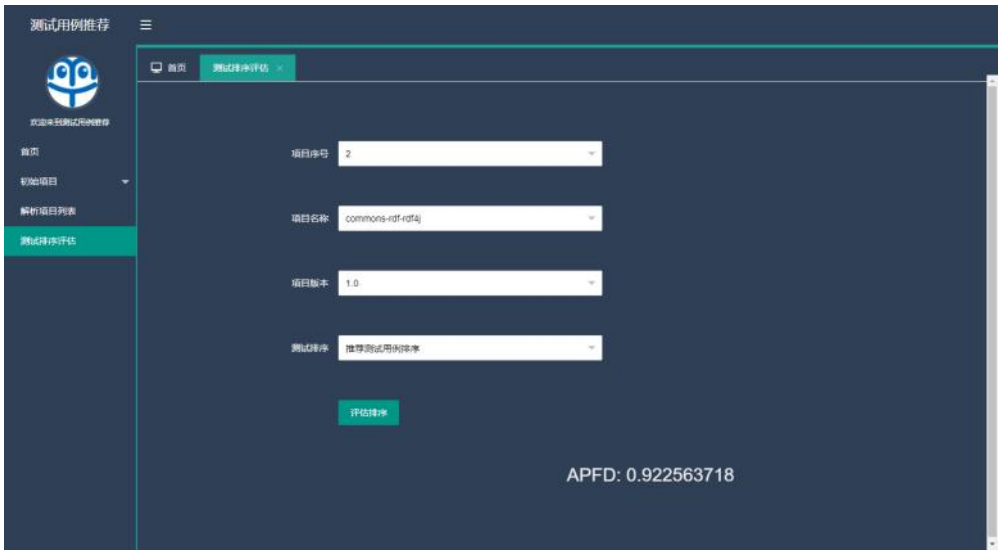


图 4.12: 系统评估页面实现

4.5 本章小结

本章主要对基于覆盖表示学习的测试用例排序系统的实现细节进行了描述。首先对关系表示图的实现进行了详细的阐述，主要包括语句关系生成模块的实现、测试覆盖生成模块的实现以及总体关系图生成模块的实现。其次阐述了各种排序算法的具体实现，然后就模型的各个模块以及实现做了详细说明，最后对人机交互节面进行了展示。

第五章　基于覆盖表示学习的测试用例排序系统的测试

5.1　系统测试

5.1.1　准备环境

本文对基于覆盖表示学习的测试用例排序系统的测试主要分为三方面，首先是对系统进行功能性测试，测试功能点主要为第三章用例描述中的系统测试用例，通过每个测试用例来验证需要实现的需求用例是否达到要求。再者是对系统的非功能性测试，主要包括人机交互时系统的响应时间以及出现系统宕机自动重启时间。最后是对系统的效果测试，主要是将模型推荐测试用例排序的 APFD 与不同 baseline 进行比较。

表 5.1: 系统测试环境表

环境名称	环境信息
操作系统	Ubuntu 20.04 LTS
数据库	8.0.18
其他环境及工具	JDK 1.8.0_261 Python 3.6.8 Git 2.28.0 Spring Boot 2.1.1.RELEASE Layui 2.5.6

如表5.1所示，为系统测试环境表，展示系统测试所用到的环境信息。系统开发是在 Window10 系统上进行的，因为大多服务器环境为 Liunx，遂在 Ubuntu 系统环境下进行测试。使用的数据库为 MySQL8。Java 开发环境为 JDK1.8，Python 版本为 3.6，主要使用后端开发框架 Spring Boot 的版本为 2.1.1 版本，前端开发框架 Layui 的版本为 2.5.6。

5.1.2　功能性测试

本章节根据第三章的功能需求进行分析，对系统用例进行功能测试设计，主要做法是按照用例描述的步骤对指定用例进行操作执行，以验证系统是否符合预期，是否出现功能上的异常，从而判断系统功能是否完善。

表5.2描述了导入项目的测试用例，测试用户将项目导入系统的功能。该测试用例不仅对系统与用户的交互进行测试，同时还要验证文件是否导入系统，并保证导入系统的文件与源文件保持一致。

表 5.2: 导入项目测试用例

测试 ID	TC1
测试名称	导入项目测试用例
待测功能	系统使用人员导入项目，初始项目列表中可以浏览到创建的项目
测试步骤	1、用户进入系统 2、用户点导航栏中的“导入报告”链接 3、用户输入项目内容，并点击“提交项目”按钮 4、用户看到提交成功或提交失败的提示
预期结果	1、系统显示首页页面 2、系统跳转到导入项目页面 3、系统获取用户提交内容 4、系统对用户提交情况给出反馈
实际结果	与预期相符合

表5.3描述了查看项目列表的测试用例，主要对项目列表页面进行测试，以测试系统功能是否完善。主要是对初始项目进行增删改查的操作，并验证是否达到预期效果。

表 5.3: 查看项目列表测试用例

测试 ID	TC2
测试名称	查看项目列表测试用例
待测功能	系统使用人员进入项目列表页面，系统展示项目信息，并可以对项目进行解析、编辑和删除操作。
测试步骤	1、用户进入系统 2、用户点击项目列表按钮，进入项目列表页面 3、用户查看项目列表页面 4、用户点击编辑按钮，可以对项目进行修改 5、用户点击删除按钮，可以对项目进行删除
预期结果	1、系统显示首页页面 2、系统跳转到导入项目页面 3、系统展示所有项目的基本信息 4、系统将可以修改的内容变成文本框，并保存用户更改后的数据 5、系统将用户点击删除的项目进行删除
实际结果	与预期相符合

表 5.4: 解析项目测试用例

测试 ID	TC3
测试名称	解析项目测试用例
待测功能	系统使用人员点击“解析”按钮后，系统会对项目进行解析，并返回解析结果，用户可以在已完成解析页面查看。
测试步骤	1、用户进入系统 2、用户点击项目列表按钮，进入项目列表页面 3、用户点击某项目的“解析”按钮 4、用户等待项目解析完成 5、用户可以在已解析列表中查看
预期结果	1、系统显示首页页面 2、系统跳转到导入项目页面 3、系统接收解析项目请求，对项目进行解析 4、系统解析完成后，将该项目的“解析”按钮置为不可点击 5、系统在已解析列表中添加该项目并进行展示
实际结果	与预期相符合

表5.4描述了解析项目的测试用例，主要测试用户发出解析请求后，系统能否对项目进行解析，并将解析结果展示。

表 5.5: 查看解析项目列表测试用例

测试 ID	TC4
测试名称	查看解析项目列表测试用例
待测功能	系统使用人员进入解析项目列表页面，系统展示已经完成解析的项目信息，并用户可以对项目进行查看详情、导出排序和删除操作。
测试步骤	1、用户进入系统 2、用户点击“解析项目列表”按钮，解析项目列表 3、用户查看解析项目列表页面 4、用户点击“详情”按钮，可以查看项目的详细信息 5、用户点击“删除”按钮，可以对项目进行删除
预期结果	1、系统显示首页页面 2、系统跳转到解析项目列表页面 3、系统展示所有已完成解析项目的基本信息 4、系统跳转到该项目的详情页面 5、系统将该已解析项目删除
实际结果	与预期相符合

表5.5描述了查看已解析项目列表的测试用例，主要对已解析项目列表页面进行测试，以测试系统功能是否完善。

表5.6描述了解析项目详情的测试用例，主要查看该页面的信息展示情况。

表 5.6: 查看项目详情测试用例

测试 ID	TC5
测试名称	查看项目详情测试用例
待测功能	系统使用人员查看项目的详细情况，系统展示已完成解析项目的解析信息内容。
测试步骤	1、用户进入系统 2、用户点击“解析项目列表”按钮，解析项目列表 3、用户点击“详情”按钮，可以查看项目的详细信息
预期结果	1、系统显示首页页面 2、系统跳转到解析项目列表页面 3、系统跳转到该项目的详情页面
实际结果	与预期相符合

表5.7描述了获取测试用例排序的测试用例，主要测试不同算法计算的测试用例排序是否可以获得与展示。

表 5.7: 获取测试用例排序测试用例

测试 ID	TC6
测试名称	获取测试用例排序测试用例
待测功能	系统使用人员获取某项目测试用例排序，包括额外贪心算法、全局贪心算法以及模型推荐测试用例排序算法。
测试步骤	1、用户进入系统 2、用户点击“解析项目列表”按钮，查看解析项目列表 3、用户点击“详情”按钮，可以查看项目的详细信息 4、用户点击“额外贪心算法排序”按钮，可以查看该算法排序 5、用户点击“全局贪心算法排序”按钮，可以查看该算法排序 6、用户点击“推荐测试用例排序”按钮，可以查看该排序
预期结果	1、系统显示首页页面 2、系统跳转到解析项目列表页面 3、系统跳转到该项目的详情页面 4、系统将项目的额外贪心算法排序展示到右侧的表格中 5、系统将项目的全局贪心算法排序展示到右侧的表格中 6、系统将项目的模型推荐排序展示到右侧的表格中
实际结果	与预期相符合

表5.8描述了测试用例评估的测试用例，主要测试排序的 APFD 值是否可以正常获取与展示。

表 5.8: 测试用例评估测试用例

测试 ID	TC7
测试名称	测试用例评估测试用例
待测功能	系统使用人员看项目不同排序的 APFD 值。
测试步骤	1、用户进入系统 2、用户点击“测试用例评估”按钮，查看测试用例评估页面 3、用户选在要查看的测试用例排序信息，点击“评估排序”按钮
预期结果	1、系统显示首页页面 2、系统跳转到测试用例评估页面 3、系统返回用户查询排序的 APFD 值
实际结果	与预期相符合

表5.9描述了导出测试用例排序的测试用例，系统使用人员可以在已解析项目列表页面以及已解析项目详情页面进行测试用例的导出。

表 5.9: 导出测试用例排序测试用例

测试 ID	TC8
测试名称	导出测试用例排序测试用例
待测功能	系统使用人员导出项目的测试用例排序。
测试步骤	1、用户进入系统 2、用户点击“解析项目列表”按钮，查看解析项目列表 3、用户点击“导出排序”按钮，导出项目的所有排序 4、用户点击“详情”按钮，可以查看项目的详细信息 5、用户点击“额外贪心算法排序”按钮，可以查看测试用例额外贪心算法排序，点击“导出排序”按钮，导出该算法的排序 6、用户点击“全局贪心算法排序”按钮，可以查看测试用例全局贪心算法排序，点击“导出排序”按钮，导出该算法的排序 7、用户点击“推荐测试用例排序”按钮，可以查看测试用例模型推荐用例排序，点击“导出排序”按钮，导出该算法的排序
预期结果	1、系统显示首页页面 2、系统跳转到解析项目列表页面 3、系统将该项目的所有测试用例排序文件下载到本地 4、系统跳转到该项目的详情页面 5、系统将项目的额外贪心算法测试用例排序下载到本地 6、系统将项目的全局贪心算法测试用例排序下载到本地 7、系统将项目的模型推荐用例排序展示下载到本地
实际结果	与预期相符合

5.1.3 非功能性测试

根据第三章设计的非功能性测试，对系统的可用性、可扩展性以及易用性进行测试。在系统进行解析时，人为制造服务器宕机情况，分别在不同节点进行

宕机，检测系统重新启动后数据是否依然存在，实验结果如下表所示。

表 5.10: 系统可用性测试结果

宕机结点	数据保存情况	实验结果
运行 OpenClover 阶段	定位到结果并重新运行	符合预期
运行 Pitest 阶段	定位到结果并重新运行	符合预期
获取关系图阶段	已保存报告数据，定位到结果并重新运行	符合预期
获取变异矩阵	已保存报告数据，定位到结果并重新运行	符合预期
获取排序阶段	已保存覆盖数据，定位到结果并重新运行	符合预期
评估排序阶段	已保存排序数据，定位到结果并重新运行	符合预期

再者，系统为了提高可扩展性，对不同的信息进行了抽象处理，并封装不同的功能点，保证单个功能点为一个方法，提高内聚性，从而保证的系统的可扩展性。系统采用简洁的页面设计，每个功能尽量实现平面化展示，也即不用为了实现某个功能而让用户多次点击跳转不同的页面，使用户体验更好，同时也使系统具有较高的以易用性。

5.2 系统实验

本章主要测试基于覆盖表示学习的测试用例排序系统的有效性及效率问题，验证 CRCP 模型对测试用例排序的效果是否好于已有的 Baseline，对比的 TCP 方法中主要为全局贪心算法与额外贪心算法。评估的指标主要是 APFD。

5.2.1 有效性评估实验

有效性评估实验目的是检测在训练集和测试集为不同程序的情况下，CRCP 模型预测的测试用例排序是否有效。本实验提出的主要问题为：对于 AFPD，不同程序情况下，CRCP 模型对测试用例的排序效果是否优于其他的 TCP 技术?

该问题的目的是为了验证 CRCP 模型在训练集和测试集为不同程序的情况下，CRCP 模型对测试用例排序的预测是否有效。解答该问题主要通过将不同的程序集随机分散为训练集和测试集来进行模型训练，最终观察模型在测试集上预测排序的效果如何，评估指标为 APFD。

实验设置。实验是在 Liunx 服务器上进行，CUDA 版本为 10.2，pytorch 版本为 1.10.0+cu102。本实验选取了 198 个程序及程序模块来对以上提出的问题进行验证，实验数据如表5.15、表5.16所示。

实验过程。实验通过使用五倍交叉验证的方式处理训练数据与测试数据的比例，也即选出总数据的 20% 作为测试集合，80% 作为训练集进行 CRCP 模型，

获取测试集的排序结果，然后选取总数据的 20%，并且不同于以上测试集的数据作为下一轮的测试集数据，剩下的作为训练集数据，以此类推，经过 5 轮训练后会获取所有程序的 CRCP 模型推荐测试用例排序，然后获取其 APFD 值作为评估指标。

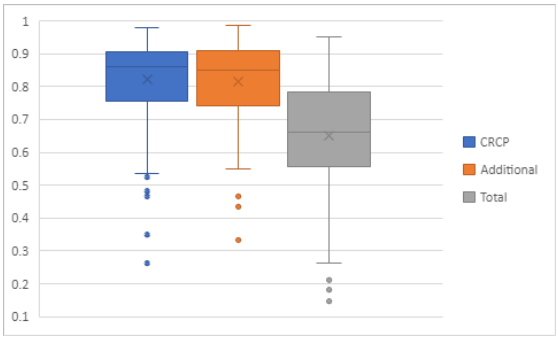


图 5.1: 跨项目程序 APFD 箱型图

实验分析。表5.15和表5.16为选取的 198 个程序及程序模块的名称，以及 CRCP 模型和其他 TCP 技术排序所计算的 APFD 情况。图5.1为上述两表的数据统计箱型图。统计可知，CRCP 模型获取的 APFD 平均值为 0.820998586，Add 算法和 Total 算法的平均值分别为 0.814363294 和 0.650289887，在平均值方面，CRCP 模型效果好于其他 TCP 技术。在中位数方面，CRCP 模型为 0.860332766，Add 算法和 Total 算法分别为 0.850889601 和 0.662681034，同样，在中位数指标方面，CRCP 模型同样好于其他 TCP 技术。通过图5.1可知，CRCP 模型获取的测试用例排序的 AFPD 更为集中一些，大多集中在 0.75 到 0.91 之间，而 Add 算法相较于 CRCP 略差一些，主要集中在 0.73 到 0.91 之间，而 Total 算法最差，数据较为分散，且数值较低。综上所述，与传统的测试用例优先级排序算法相比，基于覆盖表示学习的测试用例排序系统在排序效果上，较全局贪心算法提升 20% 左右，较额外贪心算法提升 1% 左右。这回答了上述提出的问题，即 CRCP 模型在不同程序上具有较好的有效性。

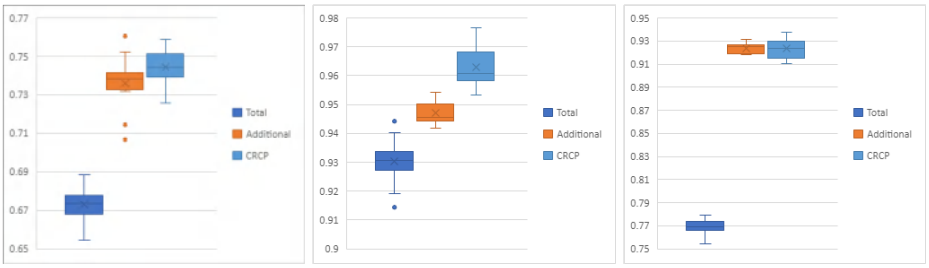
5.2.2 回归场景评估实验

回归场景评估实验目的是检测在训练集和测试集为相同程序，但是版本不同的情况下，对模型预测测试用例排序有效性的影响。本实验提出的主要问题为：对于 APFD，相同程序不同版本的情况下，CRCP 模型对测试用例的排序效果是否优于其他的 TCP 技术?

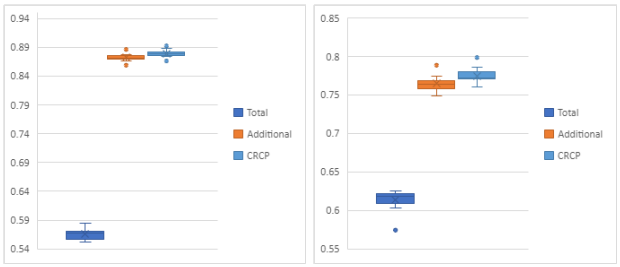
该问题是为了评估在相同程序但是版本不同情况下，基于覆盖表示学习的测试用例系统预测测试用例排序的有效性。为了回答该问题，本实验选取程序的不同版本作为模型的训练数据与测试数据，对测试集的数据进行评估，以此

来验证模型的有效性。

实验设置。实验模型的运行是在 Liunx 服务器上运行，CUDA 版本为 10.2，pytorch 版本为 1.10.0+cu102。本实验随机算取了五个程序进行实验，程序分别为 apollo[1]、cucumber-reporting[2]、commons-validator[3]、common-text[4]以及 javafaker[5]。根据 git 上的提交记录，每 5 次提交认为是一个版本，选取每个程序的最近 100 个版本作为本次实验的实验数据。这 100 个版本的规模范围如表5.12所示，其中包括程序的行数、方法数、类数、测试用例数以及杀死变异体的数量。杀死变异体的数量主要通过解析 Pitest 报告获得，其他属性则是通过 OpenClover 报告获取。每个程序都是随着迭代的增加，代码行数、类数、方法数以及测试用例的数量都在增加。



(a) apfd-apollo (b) apfd-cucumber-reporting (c) apfd-commons-validator



(d) apfd-commons-text (e) apfd-javafaker

图 5.2: 相同程序不同版本 APFD 结果

实验过程。实验将每个程序的最后的 10 个版本作为测试集合，前 90 个版本作为训练集。当第 $90+n(n \in [1,10])$ 个版本作为测试集时，前 $90+n-1$ 个版本进行训练。最后根据模型输出的测试用例排序以及变异矩阵获取 APFD。

[1]https://github.com/apolloconfig/apollo

[2]https://github.com/damianszczepanik/cucumber-reporting

[3]https://github.com/apache/commons-validator

[4]https://github.com/apache/commons-text

[5]https://github.com/DiUS/java-faker

	CRCP	Total	Add		CRCP	Total	Add		CRCP	Total	Add
v1	0.744103	0.671957	0.740010	v1	0.972976	0.940196	0.954178	v1	0.935366	0.776512	0.926999
v2	0.743931	0.654675	0.706783	v2	0.960865	0.933312	0.945569	v2	0.915078	0.769351	0.919299
v3	0.725738	0.677960	0.752122	v3	0.964888	0.934035	0.945569	v3	0.924376	0.767028	0.927151
v4	0.758736	0.662557	0.731611	v4	0.960457	0.927081	0.947169	v4	0.910421	0.770023	0.918536
v5	0.744885	0.675848	0.742092	v5	0.956977	0.929416	0.941966	v5	0.911920	0.775442	0.919035
v6	0.752397	0.671853	0.738403	v6	0.969279	0.932394	0.952756	v6	0.930220	0.778972	0.927805
v7	0.748108	0.685819	0.738403	v7	0.976600	0.944170	0.951296	v7	0.929624	0.754338	0.926421
v8	0.756044	0.688390	0.760553	v8	0.953261	0.914345	0.944604	v8	0.915579	0.763620	0.917965
v9	0.737637	0.676498	0.735622	v9	0.957891	0.919036	0.943772	v9	0.923036	0.769890	0.924519
v10	0.733425	0.667021	0.714516	v10	0.959737	0.928203	0.944374	v10	0.937571	0.765465	0.931399
平均值	0.744500	0.673258	0.736012	平均值	0.963293	0.930219	0.947125	平均值	0.923319	0.769064	0.923913
中位数	0.744494	0.673903	0.738403	中位数	0.960661	0.930905	0.945569	中位数	0.923706	0.769620	0.925470

apfd-apollo　　apfd-cucumber-reportin　　apfd-commons-validator

	CRCP	Total	Add		CRCP	Total	Add
v1	0.877009	0.571480	0.871245	v1	0.781616	0.608155	0.766995
v2	0.873471	0.554169	0.871724	v2	0.786197	0.625406	0.774511
v3	0.866179	0.554541	0.859057	v3	0.773352	0.603486	0.756723
v4	0.879980	0.583075	0.866714	v4	0.772393	0.619455	0.764485
v5	0.878557	0.559863	0.870577	v5	0.770824	0.625705	0.769165
v6	0.881768	0.570099	0.876451	v6	0.765681	0.617476	0.751930
v7	0.879280	0.551210	0.869358	v7	0.760820	0.618762	0.748723
v8	0.874920	0.564625	0.873374	v8	0.798431	0.622963	0.788768
v9	0.892076	0.570928	0.886823	v9	0.770749	0.612737	0.763513
v10	0.887618	0.585167	0.886056	v10	0.771530	0.574644	0.762794
平均值	0.879086	0.566516	0.873138	平均值	0.775159	0.612879	0.764761
中位数	0.878919	0.567362	0.871484	中位数	0.771962	0.618119	0.763999

apfd-commons-text　　apfd-javafaker

表 5.11: 相同程序不同版本 APFD 结果

实验分析。程序 apollo、cucumber-reporting、commons-validator、commons-text 以及 javafaker 后 10 个版本 CRCP 预测排序以及其他 TCP 技术获取排序的 APFD 如表5.11所示，箱型图如图5.11所示。这里的 Add 算法的 APFD 的获取是根据额外贪心算法对测试用例排序 100 次，计算 APFD，然后取平均值作为 Add 算法的 APFD。

总体来看，CRCP 模型所预测的测试用例排序整体好于其他 TCP 排序技术。对于 apollo、cucumber-reporting 和 javafaker，CRCP 模型的平均值和中值都好于其他 TCP 技术，其次是额外贪心算法。CRCP 模型比 Add 算法的平均值和中值平均高 1% 左右。但对于 commons-validator 和 commons-text 程序，CRCP 模型的排序效果也好于其他 TCP 技术的排序效果，但相比于 Add 算法，CRCP 的效果好的并不是非常明显。通过表5.12可知，commons-validator 和 commons-text 程序的规模较大，所包含的方法及测试用例较多，并且随着各项技术的迭代，项目使用的框架这在不断更新，使得相同两个版本之间可能存在较大的变化，使得 CRCP 模型对程序共同特征提取效果较差，导致出现以上的情况。综上所述，在相同程序不同版本上，CRCP 模型具有较好的有效性。

表 5.12: 程序规模统计

	Loc	Method	Classes	Test(Class/Method)	Mutants(killed)
apollo	3730–5909	237–342	57–65	21/63–39/125	134–204
cucumber-reporting	2084–4221	222–363	31–58	53/282–94/626	193–552
commons-validator	15271–18152	649-711	65-76	76/528-86/655	1285–1389
commons-test	25899–27814	1042–1152	120–132	94/1152–107/1350	3169–3482
javafaker	2017–5627	239–689	35–98	37/212–106/574	245–808

5.2.3 效率评估分析

效率评估主要是对以上两个实验进行进一步分析，主要探索两个问题：（1）CRCP 模型对测试用例的排序效率是否优于其他的 TCP 技术?（2）程序中测试用例的规模对 CRCP 模型预测是否产生影响?

问题（1）的目的是为了评估模型的排序效率是否优于其他 TCP，是否具有更好的使用价值。为了回答问题（1），使用实验二的相关数据，通过获取各项 TCP 技术以及 CRCP 模型对测试用例排序的时间来进行解答。问题（2）主要探寻的是 CRCP 模型主要适用于什么样规模的程序。问题（2）的回答使用实验一的相关数据，主要通过获取上述程序的测试用例数量于其 APFD 之间的内在关系进行解答。

表 5.13: 不同 TCP 技术执行时间统计

	Total	**Add**	**CRCP**
apollo	0.006373	0.140884	0.068551
cucumber-reporting	0.653147	0.718608	0.062668
commons-validator	2.794812	4.625782	0.056601
commons-text	11.244214	24.144822	0.045270
javafaker	1.610369	3.407485	0.058139

对于问题（1）CRCP 模型以及其他 TCP 技术排序的效率问题，表5.13和表5.12给出了回答。如表5.13所示，为 5 个程序后 10 个测试版本不同 TCP 技术以及 CRCP 模型执行排序所需时间的平均值，可以看出 CRCP 排序时间在 0.045270 到 0.068551 之间，项目规模对 CRCP 模型的排序效率影响较小。而对于其他 TCP 技术来说，测试用例排序所需要的时间是与程序规模成正比的。apollo 项目中，Total 算法的耗时要小于 CRCP 模型耗时，通过表5.12可知，这是因为 apollo 程序测试用例数较少所导致。总体来说，CRCP 模型测试用例排序耗时整体小于其他 TCP 技术，其次是 Total 算法。

表 5.14: APFD 与测试用例数量相关性

测试用例数区间	AFPD 平均值
[0-50)	0.785101796
[50-100)	0.813971717
[100-200)	0.855026635
[200-300)	0.898775218
[300-400)	0.925276934
[400-500)	0.894679003
[500-1000)	0.895929174
[1000,+∞)	0.842506311

对于问题（2）探索，如表5.14所示，程序根据测试用例数两划分为不同的部分，每一部分取 CRCP 模型预测排序所获取的 AFPD 的平均值。图5.3为表5.14的可视化展示。从表5.14和图5.3中可知，对于测试用例较少的程序，模型的预测效果较差，测试用例在区间 $[0, 50)$ 和 $[50, 100)$ 的程序的 APFD 只有 0.78 和 0.81，小于平均值，而随着测试用例的增多，CRCP 模型的预测效果逐渐变好，到区间 $[300, 400)$ 平均值达到最大。而随着测试用例的继续增大，CRCP 模型的预测效果在逐渐变差，但随后的 APFD 依然高于平均值。综上所述，程序的测试用例规模会影响 CRCP 模型测试用例排序预测的效果，当测试用例太小时，预测效果较差，测试用例较大时，预测效果较好，当程序的测试用例数量在 $[300, 400)$ 区间时，CRCP 模型的预测效果最好。

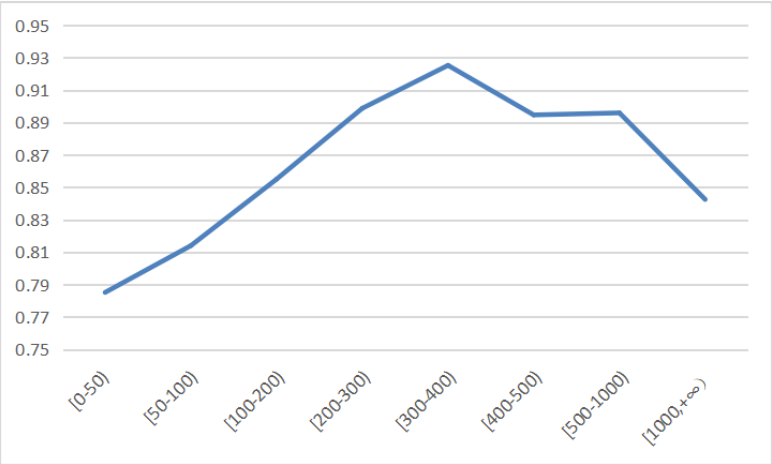


图 5.3: APFD 与测试用例数量相关性

表 5.15: 跨程序 APFD 统计（1）

程序名	CRCP	Add	Total	程序名	CRCP	Add	Total
activemq-jaas	0.876263	0.874768	0.529776	gson-fire	0.936458	0.933681	0.731791
activemq-junit	0.348889	0.333211	0.720000	hadoop-auth	0.841301	0.808775	0.557332
activemq-perf-maven-plugin	0.750781	0.765625	0.568382	hadoop-nfs	0.858756	0.845806	0.629427
android-volley	0.906137	0.919602	0.441645	hadoop-resourceestimator	0.970567	0.971256	0.479515
ansj-seg	0.757628	0.796092	0.560189	http-proxy-servlet	0.956297	0.953613	0.913296
apollo-core	0.743175	0.700311	0.682627	hutool-cron	0.914211	0.912316	0.771861
AxonFramework-messaging	0.850127	0.860009	0.592250	hutool-crypto	0.838910	0.842530	0.706186
commons-cli-cli	0.948789	0.946418	0.740220	hutool-db	0.865144	0.869715	0.752435
commons-codec	0.965286	0.973489	0.779141	hutool-extra	0.543231	0.577014	0.528037
commons-compress-rel	0.822583	0.826603	0.480225	hutool-http	0.896387	0.892593	0.261413
commons-dbutils	0.954634	0.950786	0.599640	hutool-json	0.851255	0.832361	0.758467
commons-email	0.899178	0.885443	0.638185	hutool-jwt	0.855926	0.860167	0.744792
commons-exec	0.810103	0.847984	0.577255	hutool-poi	0.682704	0.643326	0.574136
commons-fileupload	0.934490	0.921100	0.863605	jackson-core	0.904479	0.897929	0.465182
commons-functor	0.806912	0.852110	0.915738	jackson-dataformat-csv	0.880734	0.885564	0.691560
commons-geometry-core	0.912788	0.921251	0.694672	jackson-datatype-guava	0.820179	0.821219	0.573401
commons-geometry-io-core	0.886485	0.885056	0.532139	jackson-datatype-joda	0.846121	0.846445	0.771562
commons-geometry-io-euclidean	0.934560	0.933941	0.802086	JActor-jactor	0.695326	0.690078	0.403932
commons-geometry-spherical	0.891782	0.929080	0.875602	JAdventure	0.751407	0.744031	0.540385
commons-io-commons-io	0.840557	0.836719	0.623772	jarchivelib	0.896032	0.896032	0.885338
commons-number-quaternion	0.923775	0.912930	0.822658	jasmine-maven-plugin	0.669217	0.703172	0.594683
commons-numbers-complex	0.898744	0.899417	0.902463	java-apns	0.930229	0.924738	0.556642
commons-numbers-fraction	0.862045	0.868409	0.858060	javaewah	0.913813	0.928344	0.737994
commons-pool	0.922547	0.915477	0.707874	javafaker	0.743208	0.733912	0.344694
commons-rdf-api	0.604167	0.615850	0.357787	javapoet	0.839402	0.885508	0.788159
commons-rdf-jena	0.899877	0.894256	0.635230	java-uuid-generator	0.863877	0.875567	0.833920
commons-rdf-jsonld-java	0.946023	0.944330	0.630980	jprotobuf-jprotobuf	0.908177	0.893377	0.633312
commons-rdf-rdf4j	0.922564	0.926672	0.745609	jsoup-learning-master	0.901544	0.916665	0.714141
commons-rdf-simple	0.823112	0.825892	0.714341	jsprit-core	0.898507	0.854858	0.657777
commons-rng-core	0.903598	0.906705	0.500067	JustAuth	0.890275	0.868632	0.812742
commons-text-commons-text	0.886751	0.887170	0.687967	la4j	0.938772	0.935593	0.676681
commons-validator	0.943837	0.946668	0.867969	lambdaj-master	0.907409	0.909455	0.568506
cucumber-reporting	0.958492	0.935142	0.899171	languagetool-core	0.825870	0.887103	0.857775
dictomaton-dictomaton	0.925106	0.920146	0.794726	LastCalc	0.818688	0.824028	0.677106
easy-rules-core	0.863206	0.839600	0.650284	Liqp	0.913981	0.901581	0.725118
easy-rules-jexl	0.794118	0.741765	0.435185	logstash-logback-encoder	0.905051	0.901632	0.766498
easy-rules-mvel	0.700000	0.672000	0.181250	low-gc-membuffers	0.910280	0.906804	0.815804
easy-rules-spel	0.829412	0.806118	0.875926	metrics-core	0.934713	0.941451	0.885550
easy-rules-support	0.889600	0.892000	0.566154	metrics-graphite	0.719718	0.717040	0.594188
efflux-master	0.882504	0.875041	0.576200	metrics-healthchecks	0.720285	0.722024	0.741463
elasticsearch-analysis-pinyin	0.943361	0.942723	0.922619	metrics-jersey2	0.864352	0.850463	0.791111
ews-java-api	0.719831	0.729585	0.351410	metrics-jvm	0.848045	0.854947	0.566342
exp4j	0.941743	0.913297	0.715359	mp3agic-mp3agic	0.933409	0.941181	0.318055
failsafe	0.825770	0.806016	0.646048	Mybatis-PageHelper	0.943644	0.934977	0.780738
flume-http-sink	0.878667	0.781539	0.488095	mybatis-spring-boot-autoconfigure	0.879082	0.831380	0.740887
flume-jms-source	0.871429	0.866938	0.677902	nlp-lang	0.863556	0.837006	0.738710
flume-ng-configuration	0.939286	0.939286	0.780296	nv-websocket-client	0.979046	0.955091	0.211912
flume-ng-embedded-agent	0.809286	0.654857	0.505556	otto	0.896626	0.893991	0.911073
flume-ng-log4jappender	0.794444	0.820422	0.777895	pentaho-kettle-dbdialog	0.537879	0.590909	0.149524
flume-ng-morphline-solr-sink	0.760000	0.770768	0.655238	pentaho-kettle-engine-configuration-impl	0.926365	0.929680	0.473098
flume-ng-node	0.907692	0.911538	0.886905	pentaho-kettle-jms	0.788889	0.726984	0.586364
gdx-artemis	0.903584	0.863139	0.671373	pentaho-kettle-json-core	0.872165	0.912376	0.825579
geojson-jackson	0.789474	0.770865	0.597605	pentaho-kettle-meta-inject	0.820241	0.825658	0.670590
graphhopper-web-api	0.811696	0.851502	0.336277	pentaho-kettle-mqtt	0.754605	0.800794	0.899786
greenhouse	0.850618	0.826566	0.637353	pentaho-kettle-repositories-core	0.262500	0.340100	0.146429
gremlin-java	0.896397	0.897820	0.672875	pentaho-kettle-salesforce-core	0.784206	0.769483	0.321226
gson-fire	0.936458	0.933681	0.731791	pentaho-kettle-xml-core	0.912552	0.905423	0.453130

表 5.16: 跨程序 APFD 统计（2）

程序名	CRCP	Add	Total	程序名	CRCP	Add	Total
protoparser-protoparser	0.962753	0.962607	0.604234	tutorials-core-java-exceptions-3	0.681818	0.721273	0.503333
pusher-websocket-java	0.891274	0.849165	0.648443	tutorials-core-java-io-apis	0.780000	0.780000	0.706000
QRGen-core	0.823292	0.828875	0.771646	tutorials-core-java-lambdas	0.940920	0.936377	0.908201
raml-java-parser-1	0.943622	0.946923	0.926596	tutorials-core-java-lang	0.620139	0.606750	0.625421
raml-parser-2	0.898606	0.954652	0.873887	tutorials-core-java-lang-2	0.959459	0.986486	0.807229
RestFixture	0.873863	0.863836	0.396915	tutorials-core-java-lang-3	0.721176	0.696282	0.351556
rome	0.958992	0.921015	0.813384	tutorials-core-java-lang-math	0.466209	0.466209	0.861875
scribejava-core	0.859328	0.878506	0.615651	tutorials-core-java-lang-oop-methods	0.838286	0.776526	0.884066
sentinel-apache-dubbo-adapter	0.706085	0.736397	0.684596	tutorials-core-java-lang-oop-modifiers	0.734783	0.750565	0.568548
sentinel-cluster-server-default	0.608872	0.636241	0.627857	tutorials-core-java-lang-oop-patterns	0.907407	0.907407	0.826667
sentinel-dashboard	0.632867	0.668791	0.592808	tutorials-core-java-lang-oop-types	0.936000	0.932480	0.698656
sentinel-parameter-flow-control	0.772565	0.758304	0.657994	tutorials-core-java-lang-syntax	0.957333	0.942773	0.751938
skype-java-api-skype-java-api	0.841310	0.833369	0.289216	tutorials-core-java-persistence	0.687500	0.667500	0.732143
spring-retry	0.874823	0.852798	0.623881	tutorials-core-java-security-2	0.700000	0.645938	0.834821
stateless4j	0.948590	0.941335	0.839306	tutorials-core-java-streams	0.802165	0.790234	0.888338
statsd-jvm-profiler	0.884402	0.851316	0.615288	tutorials-core-java-string-algorithms	0.524508	0.554281	0.636486
stream-lib	0.973353	0.967805	0.786408	tutorials-core-java-string-algorithms-2	0.682955	0.732298	0.610025
struts-plugins-convention	0.861338	0.885113	0.502978	tutorials-core-java-string-conversions-2	0.781046	0.806510	0.727337
struts-plugins-javatemplates	0.970016	0.965898	0.679334	tutorials-design-patterns-behavioral	0.629412	0.434706	0.422222
struts-plugins-json	0.945575	0.945651	0.845728	tutorials-design-patterns-creational	0.584722	0.652236	0.369333
struts-plugins-portlet	0.857940	0.860441	0.545828	tutorials-hibernate-mapping	0.882000	0.883760	0.933571
struts-plugins-rest	0.864646	0.854079	0.605719	tutorials-jackson	0.734545	0.721336	0.470556
struts-plugins-spring	0.956508	0.939181	0.802279	tutorials-jackson-annotations	0.837778	0.765289	0.729333
tamper-tamper	0.881639	0.875638	0.847839	tutorials-jackson-conversions	0.782851	0.770955	0.571165
tutorials-algorithms-miscellaneous-3	0.481522	0.473357	0.586077	tutorials-jackson-conversions-2	0.534943	0.553030	0.672973
tutorials-algorithms-miscellaneous-4	0.705080	0.685658	0.307673	tutorials-jackson-simple	0.623016	0.648724	0.487222
tutorials-algorithms-miscellaneous-5	0.768288	0.701600	0.360744	tutorials-java-collections-conversions	0.822619	0.724738	0.787963
tutorials-algorithms-miscellaneous-6	0.725619	0.762135	0.565984	tutorials-java-numbers	0.799145	0.808171	0.778736
tutorials-algorithms-searching	0.747550	0.726344	0.502857	tutorials-java-numbers-2	0.482576	0.473133	0.770106
tutorials-assertion-libraries	0.755172	0.755552	0.470339	tutorials-java-numbers-3	0.841290	0.860227	0.710673
tutorials-cdi	0.657895	0.657895	0.058333	tutorials-json	0.618519	0.569622	0.704321
tutorials-core-java-8	0.763866	0.805737	0.663362	tutorials-libraries-4	0.905000	0.771325	0.493662
tutorials-core-java-8-2	0.961538	0.887692	0.950000	tutorials-libraries-5	0.863636	0.710909	0.910000
tutorials-core-java-8-datetime	0.741534	0.655571	0.633796	tutorials-libraries-data-io	0.887500	0.831250	0.656250
tutorials-core-java-arrays-convert	0.571830	0.576859	0.564387	tutorials-mapstruct	0.875877	0.843394	0.555299
tutorials-core-java-arrays-operations-advanced	0.754762	0.739690	0.613000	tutorials-mockito-2	0.840571	0.795360	0.840179
tutorials-core-java-arrays-operations-basic	0.834472	0.808379	0.358080	tutorials-orika	0.861806	0.894931	0.662000
tutorials-core-java-arrays-sorting	0.597403	0.625143	0.398942	tutorials-vavr	0.966667	0.966667	0.658974
tutorials-core-java-collections	0.604000	0.612000	0.655000	vraptor-core	0.780835	0.779333	0.478575
tutorials-core-java-collections-list	0.780124	0.714444	0.520952	webbit	0.879160	0.879015	0.419937
tutorials-core-java-collections-list-3	0.526829	0.579075	0.837019	webcam-capture	0.912320	0.886133	0.891819
tutorials-core-java-collections-maps-2	0.586957	0.548348	0.822414	WorldGuard	0.964079	0.961074	0.740666
tutorials-core-java-exceptions	0.878947	0.876895	0.795238				

5.2.4 有效性威胁

本小节主要分析实验中可能会对实验效果产生影响的因素：

内部有效性威胁在于技术的实现于脚本的运行。为了减少该威胁带来的影响，我们进行了有效的沟通，并请求除开发者以外的多位软件工程师检查了脚本。同时我们在最先进的框架上构建了他们，例如 PyTorch[6] 。我们还尽量复用已有代码以减少内部威胁。对于外部威胁主要是 Baseline 的获取。为了减少对比方法带来的偶然性，我们对 Add 算法进行了 100 次排序，并获取 100 次排序的平均值作为 Add 算法的数值，以此来减少外部威胁对实验的影响。

5.3 本章小结

本章主要对基于覆盖表示学习的测试用例排序系统进行了测试，测试指标为第三章中描述的系统功能需求、非功能需求及效果测试。从功能需求测试、可用性测试及效果测试的角度对系统进行了严格的测试。测试结果表明，系统功能测试已经达到规格要求，系统功能运行正常。效果测试结果展示系统所给出的测试用例排序极具借鉴价值，可以有效在测试过程中及早发现问题并解决。

[6] https://pytorch.org.

第六章　总结与展望

6.1　总结

随着互联网行业的兴起，软件开发逐渐进入应用于各行各业。随着软件功能的逐渐增多，软件代码规模必然呈上升趋势。保证代码的正确性成为了重中之重。软件测试是保证代码的正确性关键手段，而面对庞大的软件测试用例集合，如何有序的运行测试用例可以尽快发现问题，成为了现如今学者研究的热点问题。现如今多数传统测试用例排序技术仅仅通过分析测试用例代码覆盖信息而对测试用例进行优先级排序，而忽略的代码本身存在的有效关系。所以，本文提出了基于覆盖表示学习的测试用例排序系统，主要工作如下：

首先对现如今已有的测试用例优先级技术进行了调研，分析可能存在的问题，并提出的解决该问题的思路。本文中，我们提出了一种新的基于覆盖的测试用例排序技术 CRCP，该技术充分利用了覆盖信息和基于图的表示学习。我们首先提出了一种新的基于图的表示，将所有详细的覆盖信息和语句粒度的代码结构保留到一个图中：将测试和程序实体作为节点，而将覆盖和代码关系作为边缘。然后，我们利用门控图神经网络从基于图的覆盖表示中学习有价值的特征，并对程序实体进行排序。我们对五个程序的一百个版本和近二百个程序进行了评估。评估表明，CRCP 优于传统的额外贪心算法与全局贪心算法。其中，对于较大程序，CRCP 在排序效率上明显优于传统的额外贪心算法与全局贪心算法。

最后，我们将 CRCP 技术核心功能进行封装，使用 SpringBoot 以及 Layui 前后端开发框架，搭建起一个具有实用价值的 Web 应用。本应用提供 CRCP 技术的测试用例优先级排序以及额外贪心算法和全局贪心算法测试用例优先级排序功能，用户可以根据需求自主选择测试用例排序算法，大大增加了获取排序的灵活性。同时，系统也提供了测试用例的评估功能，评估指标为 APFD。并通过系统测试，证明系统功能完备，具有较高的使用价值。

6.2　展望

本文设计并实现了基于覆盖表示学习的测试用例排序系统，是在测试用例优先级领域的一次探索，同时也是在企业软件开发领域的一次实践。但是由于时间较短，以及个人涉足该领域较浅，本文提出的基于覆盖表示学习的测试用例排序系统在技术和实现上还有许多需要改进和优化的地方。

（1）技术方面，本系统虽然对语句粒度的代码关系实现了有效获取，但是可以进一步对方法之间以及类之间的关系进行探索。在测试用例方面，本文虽然对方法级别的测试用例进行了探索，并取得了较好的效果，可进一步对类粒度的测试用例进行尝试。总之在后续探索中，可对不同粒度的代码关系以及测试用例关系进行尝试，评估在不同粒度上，CRCP 技术是否具有较好的效果。

（2）数据方面，本文的数据涉及近二百个程序及模块，但在后续的发展中，可进一步收集更多的程序进行有效训练，选取高质量代码进行处理，或者，对高质量数据进行有效的数据扩增从而提高数据规模。

（3）系统方面，本文对系统的实现模型，主要为跨程序模型，现如今虽然具有较好的适用性，但不同程序之间代码差异性较大。后续随着系统中相同程序不同版本的增加，可以增加不同程序不同模型的功能，每个程序使用其历史版本训练本程序模型，只用于本程序的测试用例推荐，每个程序都具有其单独的推荐模型，从而增加模型的推荐效果。

参考文献

[1] LOU Y, ZHU Q, DONG J, et al. Boosting coverage-based fault localization via graph-based representation learning[C] // Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021 : 664 – 676.

[2] 杨芙清. 软件工程技术发展思索 [J]. 软件学报, 2005, 16(1).

[3] DOWSON M. Iteration in the software process; review of the 3rd International Software Process Workshop[C] // Proceedings of the 9th international conference on Software Engineering. 1987 : 36 – 41.

[4] LOU Y, HAO D, ZHANG L. Mutation-based test-case prioritization in software evolution[C] // 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE). 2015 : 46 – 57.

[5] NERUR S, BALIJEPALLY V. Theoretical reflections on agile development methodologies[J]. Communications of the ACM, 2007, 50(3) : 79 – 83.

[6] CHITTIMALLI P K, HARROLD M J. Recomputing coverage information to assist regression testing[J]. IEEE Transactions on Software Engineering, 2009, 35(4) : 452 – 469.

[7] LU Y, LOU Y, CHENG S, et al. How does regression test prioritization perform in real-world software evolution?[C] // Proceedings of the 38th International Conference on Software Engineering. 2016 : 535 – 546.

[8] 陈翔, 陈继红, 鞠小林, et al. 回归测试中的测试用例优先排序技术述评 [J]. 软件学报, 2013, 8.

[9] CHEN J, LOU Y, ZHANG L, et al. Optimizing test prioritization via test distribution analysis[C] // Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018 : 656 – 667.

[10] KUMAR A. Development at the speed and scale of google[J]. QCon San Francisco, 2010.

[11] YOO S, HARMAN M. Regression testing minimization, selection and prioritization: a survey[J]. Software testing, verification and reliability, 2012, 22(2): 67 – 120.

[12] CRUCIANI E, MIRANDA B, VERDECCHIA R, et al. Scalable approaches for test suite reduction[C] // 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). 2019: 419 – 429.

[13] BRIAND L C, LABICHE Y, HE S. Automating regression test selection based on UML designs[J]. Information and Software Technology, 2009, 51(1): 16 – 30.

[14] MAIA C L B, do CARMO R A F, de FREITAS F G, et al. Automated test case prioritization with reactive GRASP[J]. Advances in Software Engineering, 2010, 2010.

[15] 顾庆, 唐宝, 陈道蓄. 一种面向测试需求部分覆盖的测试用例集约简技术 [D]. [S.l.]: [s.n.], 2011.

[16] ELBAUM S, ROTHERMEL G, PENIX J. Techniques for improving regression testing in continuous integration development environments[C] // Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2014: 235 – 245.

[17] ROTHERMEL G, UNTCH R H, CHU C, et al. Prioritizing test cases for regression testing[J]. IEEE Transactions on software engineering, 2001, 27(10): 929 – 948.

[18] WONG W E, HORGAN J R, LONDON S, et al. A study of effective regression testing in practice[C] // PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering. 1997: 264 – 274.

[19] LI Z, HARMAN M, HIERONS R M. Search algorithms for regression test case prioritization[J]. IEEE Transactions on software engineering, 2007, 33(4): 225 – 237.

[20] JIANG B, ZHANG Z, CHAN W K, et al. Adaptive random test case prioritization[C] // 2009 IEEE/ACM International Conference on Automated Software Engineering. 2009 : 233 – 244.

[21] SHARIF A, MARIJAN D, LIAAEN M. DeepOrder: Deep Learning for Test Case Prioritization in Continuous Integration Testing[J]. arXiv preprint arXiv:2110.07443, 2021.

[22] WONG W E, DEBROY V, GOLDEN R, et al. Effective software fault localization using an RBF neural network[J]. IEEE Transactions on Reliability, 2011, 61(1) : 149 – 169.

[23] ZHANG Z, LEI Y, MAO X, et al. CNN-FL: An effective approach for localizing faults using convolutional neural networks[C] // 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). 2019 : 445 – 455.

[24] SRIKANTH H, WILLIAMS L, OSBORNE J. System test case prioritization of new and regression test cases[C] // 2005 International Symposium on Empirical Software Engineering, 2005.. 2005 : 10 – pp.

[25] KRISHNAMOORTHI R, MARY S S A. Factor oriented requirement coverage based system test case prioritization of new and regression test cases[J]. Information and Software Technology, 2009, 51(4) : 799 – 808.

[26] KROLL, KRUCHTEN. The Rational Unified Process Made Easy[M]. [S.l.] : Pearson India, 2003.

[27] KRUCHTEN P. The rational unified process: an introduction[M]. [S.l.] : Addison-Wesley Professional, 2004.

[28] BALL T, KIM J-M, PORTER A A, et al. If your version control system could talk[C] // ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering : Vol 11. 1997.

[29] BEYER D, NOACK A. Clustering software artifacts based on frequent common changes[C] // 13th International Workshop on Program Comprehension (IWPC'05). 2005 : 259 – 268.

[30] DEMMEL J W. Applied numerical linear algebra[M]. [S.l.] : SIAM, 1997.

[31] SHERRIFF M, LAKE M, WILLIAMS L. Prioritization of regression tests using singular value decomposition with empirical change records[C] // The 18th IEEE international symposium on software reliability (ISSRE'07). 2007 : 81 – 90.

[32] JONES J A, HARROLD M J. Test-suite reduction and prioritization for modified condition/decision coverage[J]. IEEE Transactions on software Engineering, 2003, 29(3) : 195 – 209.

[33] CHILENSKI J J, MILLER S P. Applicability of modified condition/decision coverage to software testing[J]. Software Engineering Journal, 1994, 9(5) : 193 – 200.

[34] ELBAUM S, MALISHEVSKY A, ROTHERMEL G. Incorporating varying test costs and fault severities into test case prioritization[C] // Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001. 2001 : 329 – 338.

[35] COLES H, LAURENT T, HENARD C, et al. Pit: a practical mutation testing tool for java[C] // Proceedings of the 25th international symposium on software testing and analysis. 2016 : 449 – 452.

[36] JIA Y, HARMAN M. An analysis and survey of the development of mutation testing[J]. IEEE transactions on software engineering, 2010, 37(5) : 649 – 678.

[37] MA Y-S, OFFUTT J, KWON Y-R. MuJava: a mutation system for Java[C] // Proceedings of the 28th international conference on Software engineering. 2006 : 827 – 830.

[38] JUST R. The Major mutation framework: Efficient and scalable mutation analysis for Java[C] // Proceedings of the 2014 international symposium on software testing and analysis. 2014 : 433 – 436.

[39] DELAHAYE M, DU BOUSQUET L. Selecting a software engineering tool: lessons learnt from mutation analysis[J]. Software: Practice and Experience, 2015, 45(7) : 875 – 891.

[40] ELMENDORF W R. Controlling the functional testing of an operating system[J]. IEEE Transactions on Systems Science and Cybernetics, 1969, 5(4) : 284 – 290.

[41] INOZEMTSEVA L, HOLMES R. Coverage is not strongly correlated with test suite effectiveness[C] // Proceedings of the 36th international conference on software engineering. 2014 : 435 – 445.

[42] PIWOWARSKI P, OHBA M, CARUSO J. Coverage measurement experience during function test[C] // Proceedings of 1993 15th international conference on software engineering. 1993 : 287 – 301.

[43] FRASER G, ARCURI A. Whole test suite generation[J]. IEEE Transactions on Software Engineering, 2012, 39(2) : 276 – 291.

[44] ABREU R, ZOETEWEIJ P, GOLSTEIJN R, et al. A practical evaluation of spectrum-based fault localization[J]. Journal of Systems and Software, 2009, 82(11) : 1780 – 1792.

[45] JONES J A, HARROLD M J. Empirical evaluation of the tarantula automatic fault-localization technique[C] // Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. 2005 : 273 – 282.

[46] WONG W E, DEBROY V, GAO R, et al. The DStar method for effective software fault localization[J]. IEEE Transactions on Reliability, 2013, 63(1) : 290 – 308.

[47] BRIAND L C, LABICHE Y, LIU X. Using machine learning to support debugging with tarantula[C] // The 18th IEEE International Symposium on Software Reliability (ISSRE'07). 2007 : 137 – 146.

[48] ROTHERMEL G, UNTCH R H, CHU C, et al. Prioritizing test cases for regression testing[J]. IEEE Transactions on software engineering, 2001, 27(10) : 929 – 948.

[49] MCINTYRE D. Bridging the gap between research and practice[J]. Cambridge Journal of education, 2005, 35(3) : 357 – 382.

[50] MOU L, LI G, ZHANG L, et al. Convolutional neural networks over tree structures for programming language processing[C] // Thirtieth AAAI Conference on Artificial Intelligence. 2016.

[51] WHITE M, TUFANO M, VENDOME C, et al. Deep learning code fragments for code clone detection[C] // 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). 2016 : 87 – 98.

[52] WEI H, LI M. Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code.[C] // IJCAI. 2017 : 3034 – 3040.

[53] BAXTER I D, YAHIN A, MOURA L, et al. Clone detection using abstract syntax trees[C] // Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272). 1998 : 368 – 377.

[54] PAUL S, PRAKASH A. A framework for source code search using program patterns[J]. IEEE Transactions on Software Engineering, 1994, 20(6) : 463 – 475.

[55] WEIMER W, NGUYEN T, LE GOUES C, et al. Automatically finding patches using genetic programming[C] // 2009 IEEE 31st International Conference on Software Engineering. 2009 : 364 – 374.

[56] FALLERI J-R, MORANDAT F, BLANC X, et al. Fine-grained and accurate source code differencing[C] // Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. 2014 : 313 – 324.

简历与科研成果

基本情况 王睿智，汉族，1996 年 9 月出生，山东省滨州市人。

教育背景

2020.09 ~ 2022.06 南京大学软件学院 硕士

2015.09 ~ 2019.06 山东农业大学经管学院 本科

致　　谢

时光荏苒，岁月如梭，两年的研究生时光即将结束，回首来时路，心中不禁感慨万千。这一路走来有欢笑，有唏嘘，有失落，有赞叹。在硕士即将结束之际，我想对这一路帮助过我的人表示最真挚的感谢与祝福。

首先，我想要感谢在我硕士期间给予我极大帮助的陈振宇导师以及房春荣老师，感谢老师们在我刚踏入软件工程领域时对我的指导和照顾，让我建立起对这片领域的认知，这使我受益匪浅。同时也感谢慕测平台的黄勇老师，让我对实际的开发有了初步的认识与了解，为我在技术上的提升做了很好的铺垫。

其次，我要感谢张犬俊学长为我的毕业设计提供方向与思路，他细致入微的思维与严谨不苟态度深深的影响了我，为我树立了很好的榜样。我还要感谢李文龙学长、孙伟松学长、郭安同学、葛修婷学姐、刘佳玮学姐以及 iSE 实验室的所有同学，他们在我研究生阶段给予了我太多的帮助。

再者，我要感谢我的室友王舒遥、王擎宇和王佩旭，感谢他们在生活和学习上给予我无私的关怀，让我在学习之余有了更多欢乐的时光。同时也感谢我的好友恽叶霄、吴青衡，是你们让我的研究生生涯变得难忘。

这里，我要特别感谢我的家人和女朋友，在我最难过、最失望的时候，永远是你们在我的背后给予我最大的支持，你们是我不懈努力的动力，感谢你们给予的爱与关怀。

最后，对能在百忙之中审阅我论文的各位老师、专家表示由衷的感谢！

学位论文出版授权书

版权与原创性说明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期：2022 年 05 月 22 日