软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(6):2562−2585 [doi: 10.13328/j.cnki.jos.006853]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

结合情节挖掘的软件实体演化耦合分析方法*

张鑫雨[1], 晋武侠[1], 刘靖雯[2], 范 铭[2], 刘 烃[2]

[1](西安交通大学 软件学院, 陕西 西安 710049)

[2](西安交通大学 网络空间安全学院, 陕西 西安 710049)

通信作者: 晋武侠, E-mail: jinwuxia@mail.xjtu.edu.cn

摘 要: 软件系统的实体演化耦合分析有助于共同变更预测、软件供应链风险识别、代码漏洞溯源、缺陷预测、架构问题定位等分析活动. 两个代码实体之间存在演化耦合(evolutionary coupling)是指在软件修订历史中, 这对实体倾向于共同变更(共变). 已有的演化耦合分析方法难以准确检测软件维护历史中频繁发生的、有“距离”的共变. 为了解决这一问题, 提出了基于关联规则挖掘、情节挖掘、潜在语义索引模型相结合的演化耦合分析方法(association rule, MINEPI and LSI based method, AR-MIM), 以挖掘有“距离”的共同变更关系. 实验收集了 58 个 Python 项目、242 074 条训练数据、330 660 条 ground truth 的数据集, 与已有的 4 种 baseline 方法进行了比较, 验证了 AR-MIM 的效果. 结果表明: 在预测共同变更候选项场景上, AR-MIM 的准确性、召回率、*F*1 分数均优于已有方法.

关键词: 提交历史; 演化耦合; 情节挖掘; 潜在语义索引; 关联规则挖掘

中图法分类号: TP311

中文引用格式: 张鑫雨, 晋武侠, 刘靖雯, 范铭, 刘烃. 结合情节挖掘的软件实体演化耦合分析方法. 软件学报, 2023, 34(6): 2562–2585. http://www.jos.org.cn/1000-9825/6853.htm

英文引用格式: Zhang XY, Jin WX, Liu JW, Fan M, Liu T. Evolutionary Coupling Analysis Method of Software Entity Based on Episode Mining. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2562–2585 (in Chinese). http://www.jos.org.cn/1000-9825/6853.htm

Evolutionary Coupling Analysis Method of Software Entity Based on Episode Mining

ZHANG Xin-Yu[1], JIN Wu-Xia[1], LIU Jing-Wen[2], FAN Ming[2], LIU Ting[2]

[1](School of Software Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

[2](School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: The entity evolutionary coupling analysis of software systems is helpfulfor analysis activities such as co-change candidate prediction, risk identification of software supply chain, code vulnerability traceability, defect prediction and architecture problem localization. The evolutionary coupling between two entities indicates that these entities tend to be changed together in the software revision history. Existing methods present a low accuracy to detect the frequent “having distance” co-change in the revision history. To address this problem, this study proposes an evolutionary coupling analysis method based on the combination of association rule mining, episode mining and latent semantic indexing (association rule, MINEPI and LSI based method, AR-MIM), which mines co-change relations of “having distance”. The experiment verified the effectiveness of AR-MIM by compared with the four baseline methods on the dataset, collecting 58 Python projects, 242 074 pieces of training data, and 330 660 pieces of ground truth. The results show that the

* 基金项目: 国家重点研发计划(2018YFB1004500); 国家自然科学基金(61902306, 62002280, 61721002, 61833015, 62272387); 中央高校基本科研业务费专项资金; 中国博士后基金(2020M683507, 2019TQ0251, 2020M673439); 西安市科协青年人才支持计划(095920201303)

本文由“软件可信性与供应链安全前沿进展”专题特约编辑向剑文教授、郑征教授、申文博研究员、常瑞副教授、田聪教授推荐.

收稿时间: 2022-09-05; 修改时间: 2022-10-10, 2022-12-14; 采用时间: 2022-12-28; jos 在线出版时间: 2023-01-13

precision, recall, and *F*1 score of AR-MIM are better than those of existing methods in co-change candidate prediction.
Key words: commit history; evolutionary coupling; episode mining; latent semantic indexing (LSI); association rule mining

软件系统的实体演化耦合分析是共同变更候选项预测[1−7]、软件供应链风险识别[8,9]、缺陷预测[10−17]、架构问题定位[18−22]、横切关注点检测[23−25]等软件分析任务的基础. 软件演化耦合分析的相关研究包含两类: 编程语言相关的方法和编程语言无关的方法. 编程语言相关的演化耦合分析方法通过分解或者合并变更提交记录(例如 git commit)来实现精准的演化耦合度量[26,27]. 这类方法假设: 单一提交可能包含了多个不同的维护活动, 分解单一记录提升修改的原子性, 可以提高演化耦合分析的准确性; 不同开发者之间经常协作, 多次不同提交可能都是在修复相同缺陷, 因而合并多条相关提交记录可以提高演化耦合分析方法的效率. 但是, 这些方法需要解析代码修改片段的语法信息, 复杂度较高. 另一类编程语言无关的演化耦合分析方法利用挖掘技术从软件修订历史记录中挖掘潜在的共同变更实体. 这种基于挖掘的方法可扩展性较强, 能够灵活应用到不同编程语言实现的软件项目中. 已有研究工作[7,28,29]利用关联规则挖掘算法分析 Java、C++、C#项目修订记录中的历史耦合关系, 预测当一个代码实体发生修改时可能受到影响的实体, 即预测共同变更候选项. 关联规则方法认为, 频繁地在同一 commit 中发生共变的实体存在耦合关系. Islam 等人[2]提出了传递性关联规则方法, 用于检测没有在同一 commit 直接发生共变的文件之间的传递性关联. Kruizinga[19]提出的模糊重叠方法用来检测“模糊”的共变. 但是, 大部分基于历史挖掘的研究工作主要关注在“0 距离”的演化耦合, 不能很好地挖掘“有距离”的历史共变关系, 且没有考虑到开发活动记录由于工程实践问题导致分散在多次提交中的情况.

为了解决上述问题, 本文提出了基于情节挖掘的演化耦合分析方法(association rule, MINEPI and LSI based method, AR-MIM), 利用关联规则、情节挖掘、潜在语义索引模型的互补特征, 挖掘软件修订历史中实体之间频繁发生的共变, 即演化耦合关系. 为了验证所提方法的有效性, 本文收集了 Python 项目上的 benchmark 数据集, 包含 58 个项目和 330 660 条 ground truth. 首先, 在预测共同变更候选项场景上, 将所提方法与已有的关联规则方法、传递性关联规则方法、模糊重叠方法、情节挖掘方法进行比较; 然后, 与已有方法相比, 分析本文方法捕获显式依赖和隐式依赖的能力; 最后, 在不同领域的 Python 项目上, 分别比较 AR-MIM 与已有方法的准确性.

实验结果表明:

(1) 在预测共同变更候选项场景上, AR-MIM 的准确率、召回率、*F*1 值均高于已有 4 种方法;

(2) 情节挖掘捕获隐式依赖和显式依赖的能力较好;

(3) 与现有方法相比, AR-MIM 方法在各领域的 Python 项目上表现得都是最好的.

此外, 各个方法在不同领域的 Python 项目上所表现出的性能存在差异, 这一点揭示出未来可以结合领域信息以进一步提升演化耦合分析方法的性能.

总之, 本文的主要贡献如下:

(1) 提出了关联规则、情节挖掘、潜在语义索引模型相结合的演化耦合分析方法(AR-MIM), 挖掘出软件修订历史中频繁发生共同变更的文件之间的演化耦合关系;

(2) 与已有方法相比, 在预测共同变更候选项场景上验证了本文方法的准确性, 评估了不同演化耦合分析方法捕获 Python 代码中显式依赖和隐式依赖的能力差异, 比较了 5 种演化耦合分析方法在不同领域的 Python 项目上的性能差异, 并指出潜在的改进方向;

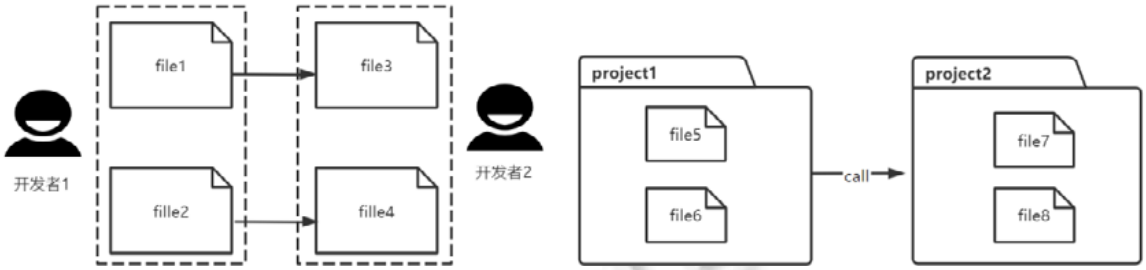
(3) 公开了所收集的数据集(https://github.com/zxylz/AR-MIM.git), 可继续用于演化耦合分析、预测共同变更候选项等研究工作中.

本文第 1 节介绍相关背景. 第 2 节详细介绍提出的关联规则、情节挖掘、潜在语义索引模型相结合的演化耦合分析方法. 第 3 节描述实验设置. 第 4 节展示实验结果. 第 5 节总结相关工作. 第 6 节对本文研究结果作进一步讨论. 第 7 节总结全文.

1 背景介绍

本节介绍软件实体演化耦合的概念、应用场景, 并通过案例解释本文的研究动机.

两个实体之间存在演化耦合(evolutionary coupling)[29]是指在软件修订历史中, 这对实体倾向于共同变更(共变). 实体 X 与 Y 的演化耦合关系可用 $X \rightarrow Y$ 表示, 如果实体 X 发生了变更, 那么实体 Y 也有变更的趋势. 其中, X 和 Y 可以是源文件、类、模块、方法、变量等粒度. 目前, 演化耦合分析已被广泛应用于变更影响分析[1,3,29]、故障定位[16]、供应链风险识别[8,9]等领域. 图 1 展示了相关的应用实例.



(a) 代码变更影响分析和缺陷预测 (b) 软件供应链风险识别

图 1 演化耦合分析应用实例

图 1(a)通过示意图介绍演化耦合分析在代码变更影响分析和缺陷预测中的应用. 图 1(a)中存在 file1–file4 这 4 个源文件, file1、file2 分别与 file3、file4 文件存在演化耦合关系. 演化耦合分析可用于变更影响分析, 当第 1 个开发者修改 file1 和 file2 后, 通过演化耦合分析, 可以得到 file3 和 file4 可能会受到影响的结论, 也需要得到开发者的关注. 演化耦合分析也可用于缺陷预测场景. 假设开发者在一个提交中修改了 file1 和 file2 中的代码, 此次修改引入了缺陷, 那么可以通过分析 file1、file2 分别与 file3、file4 之间的演化耦合关系, 预测出 file3 和 file4 可能受到前者文件修改的影响, 造成发生缺陷或者功能失效.

图 1(b)通过示意图介绍演化耦合分析在供应链风险识别中的应用. 该例子中包含项目 project1 和第三方组件包 project2. project1 包含源文件 file5 和 file6, project2 包含源文件 file7 和 file8. project1 中的文件 file5 和 file6 分别调用了 project2 中的文件 file7 和 file8. 很多供应链攻击[30]将恶意代码注入流行的软件包的代码中, 以感染供应链中依赖这些软件包的其他系统. 因此在供应链风险识别中, 可以利用演化耦合分析识别出 project1 的 file5、file6 与 project2 的 file7、file8 有耦合关系, 进而预测上游 project2 中的漏洞风险可能影响下游的 project1.

下面我们将以项目 scikit-learn 为例来介绍本文的研究动机. 例如: 在 scikit-learn 项目的 19 432 条修订历史记录中, sklearn/utils/fixes.py 和 sklearn/utils/tests/test_utils.py 文件在前 75%的修订历史中没有发生过在同一提交中的共同变更(即, 均为“0 距离”的共变), 这样会导致现有方法无法捕获它们之间的演化耦合关系. 然而, 本文发现: sklearn/utils/fixes.py 和 sklearn/utils/tests/test_utils.py 经常发生有“距离”的共变, 即 sklearn/utils/fixes.py 发生变更后, sklearn/utils/tests/test_utils.py 随后在之后的 commit 中也发生了变更. 例如, 表 1 中的第 1 行数据显示: sklearn/utils/fixes.py 在 commit 5709c5 发生变更后, sklearn/utils/tests/test_utils.py 在之后的 commit f6089f 发生了变更, 时间相差 45.5 分钟. 在前 75%的修订历史中, sklearn/utils/fixes.py 和 sklearn/utils/tests/test_utils.py 一共发生了 3 次有“距离”的共变, 距离时间差最长为 1.25 天, 最短为 45.5 分钟. 在之后 25%的提交历史中, 由于修复相同功能的缺陷, 两个文件发生了共变. 如图 2 所示: 为了修复 Issue #11211:SimpleImputer (strategy=“constant”), 同时修改了文件 sklearn/utils/fixes.py 和 sklearn/utils/tests/test_utils.py.

表 1 sklearn/utils/fixes.py 和 sklearn/utils/tests/test_utils.py 变更的信息

sklearn/utils/fixes.py		sklearn/utils/tests/test_utils.py	
commitID	Time	commitID	Time
5709c5	2011-04-25 13:04:53	f6089f	2011-04-25 13:50:24
a7a051	2014-09-22 17:23:51	766fb5	2014-09-23 23:31:02
7ac6c9	2015-04-09 08:02:03	e12863	2015-04-09 13:16:59

此外, 很多研究也表明[31−33]: 不同的开发人员经常修改相同的代码工件、合作修复复杂的 Issue 等. 这说明, 不同的提交可能是相关的. 例如, 为了实现同一功能或修复同一个 bug. 因此, 亟需考虑有“距离”的共变来提高演化耦合分析方法的准确性, 帮助提升上层应用的分析效果.

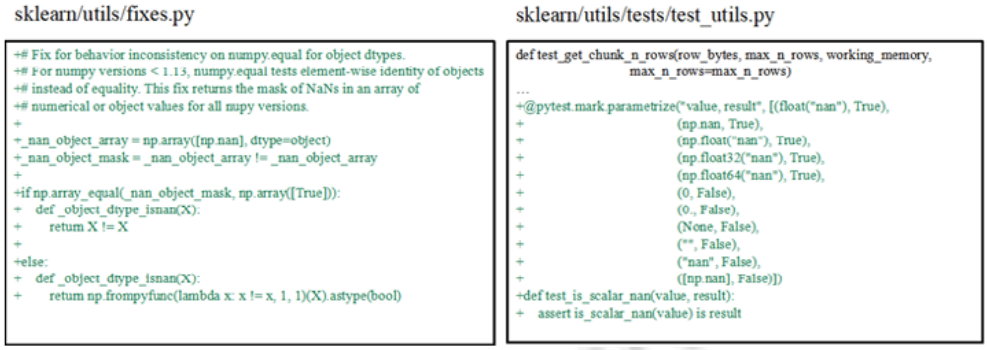


图 2 sklearn/utils/fixes.py 和 sklearn/utils/tests/test_utils.py 变更代码详情

2 关联规则、情节挖掘、潜在语义索引模型相结合的演化耦合分析方法

为了挖掘提交历史中有“距离”的共变关系, 本文提出了关联规则、情节挖掘和潜在语义索引模型相结合的演化耦合分析方法(AR-MIM). AR-MIM 利用关联规则和情节挖掘的各自特点, 从软件修订历史中提取出频繁发生共同变更的文件组. 该方法不仅能捕获同一提交中发生的共变关系, 还能捕获有“距离”的共变关系.

2.1 AR-MIM方法介绍

AR-MIM 方法框架如图 3 所示, 输入是项目修订历史记录, 输出是与目标文件有耦合关系的文件集合以及耦合关系权重. 首先进行文件的变更历史抽取, 从软件项目的修订历史记录中提取每个源文件的变更历史记录序列. 根据文件的变更历史序列, 利用关联规则、情节挖掘算法挖掘文件之间的演化耦合关系. 进一步, 基于潜在语义索引模型计算源文件之间的语义相似性, 对上一步得出的演化耦合结果进行求精. 通过关联规则挖掘、情节挖掘、语义挖掘三者结果的融合, 最终输出与目标文件有耦合关系的文件集合以及耦合权重.

图 3 AR-MIM 方法框架

接下来, 详细介绍这 5 个模块.

(1) 文件的变更历史抽取

首先, 从修订历史记录得到文件 X、Y 的变更历史, 如 X{#1,#3}、Y{#2,#4}的形式, 其中, #1–#4 分别是 commit 的 id. X{#1,#3}表示 X 在 commit#1、commit#3 中发生了变更, Y{#2,#4}表示 Y 在 commit#2、commit#4 发生了变更.

(2) 基于关联规则挖掘演化耦合(association rules-based method, ARM)

目前, 关联规则[34]已被广泛应用于预测共同变更候选项[1,7,29]、特征定位[35]等研究. 本文方法利用经典的关联规则挖掘先找出在同次 commit 提交中可能频繁共变的候选文件组.

假设: 在软件演化历史中, 如果文件 X 和 Y 经常在同一提交中共同变更(简称“共变”), 则两者之间可能存在耦合关系, 用 $X\rightarrow Y$ 表示, 其中, X 为前项, Y 为后项. 当文件 X 在未来某个特定提交中发生了变更, 那么 Y 也有在该提交中变更的倾向. 使用关联规则方法挖掘的结果都是不对称的, 即演化耦合关系 $X\rightarrow Y$ 和 $Y\rightarrow X$ 并不一定同时存在. 因为规则 $X\rightarrow Y$ 和 $Y\rightarrow X$ 的置信度(confidence)可能不同.

过程: 在文件的变更历史上, 利用 Apriori 算法[36]挖掘文件之间的关联规则. 具体来讲, 每一个提交

(commit)中发生变更的文件是项, 多次在相同提交中变更的文件的集合是频繁项集. 使用 Apriori 算法挖掘频繁 2 项集, 并生成规则.

支持度(support)和置信度(confidence)是用于生成关联规则的可调参数. 其中, *support*(*X*)表示 *X* 变更的次数, *support*(*X*→*Y*)表示文件 *X* 和 *Y* 在相同 commit 中变更的次数. *confidence*(*X*→*Y*)用来衡量文件 *X* 发生变更时, 文件 *Y* 也发生变更的概率. 公式如下所示:

$$support(X\to Y)=count(X\cup Y),\ confidence(X\to Y)=support(X\to Y)/support(X).$$

support 和 *confidence* 越高, 说明两个文件共同演化的概率越高.

(3) 基于情节挖掘演化耦合(MINEPI-based method, MIM)

MINEPI[37]是一种频繁情节挖掘(episode mining)方法, 情节挖掘已被广泛应用于预测应用程序的工作负载情况[38]、工业生产中预测故障[39]、挖掘电信网络警报规则、过滤冗余警报、定位网络中的问题、预测严重故障[40]、网络安全时间序列数据预测[41]之中. 本文方法利用 MINEPI 找出频繁发生有“距离”的共变文件组.

假设: 本文认为, 文件之间存在有“距离”的共变. 即: 如果在一个文件 *X* 发生变更后的一段提交历史中, 经常有另一个文件 *Y* 发生变更, 那么文件 *X* 与 *Y* 之间可能存在耦合关系, 用 *X*→*Y* 表示. 使用 MINEPI 得到的结果是不对称的, 因为 *X* 和 *Y* 发生有先后顺序.

过程: 算法 1 展示了使用 MINEPI 判断文件 *X* 和文件 *Y* 是否存在耦合关系 *X*→*Y* 的伪代码.

首先, 本文挖掘文件 *X* 和文件 *Y* 之间有“距离”的共变(算法 1 第 1–8 行). 对于文件 *X* 变更历史中的一个 *commit_X*, 如果文件 *Y* 的变更历史中的一个 *commit_Y* 在其之后发生, 且 *commit_Y* 和 *commit_X* 之间的距离 *width*(距离 *width* 指的是两个 *commit* 之间间隔的提交数)不超过阈值 *max_width*, 则认为发生了一次有“距离”的共变. 阈值 *max_width* 主要为了过滤掉距离太远的共变.

为了过滤掉不经常发生有“距离”共变的文件组(第 9–12 行), 本文判断了文件 *X* 和文件 *Y* 之间的有“距离”的共变次数是否超过最小共变次数阈值 *min_frequency*: 如果超过, 则认为存在耦合关系 *X*→*Y*; 反之, 则认为不存在耦合关系 *X*→*Y*.

算法 1. 使用 MINEPI 判断文件 *X* 和文件 *Y* 是否存在耦合关系 *X*→*Y*.

输入: 文件 *X* 的变更历史 *filechange_X*, 文件 *Y* 的变更历史 *filechange_Y*, 最大距离 *max_width*, 最小共变次数 *min_frequency*;

输出: 文件 *X* 与 *Y* 是否有耦合关系 *X*→*Y*.

```
1  for commit_X in filechange_X:   //获取文件 X 和文件 Y 之间全部有“距离”的共变
2    for commit_Y in filechange_Y:
3      if (time(commit_Y)>time(commit_X)) and (width(commit_Y,commit_X)≤max_width):
         //若 commit_Y 在 commit_X 之后发生, 且两个 commit 间距小于 max_width, 则认为存在有“距离”
           的共变.
4          then
5              distanceCoupling.append(⟨commit_X,commit_Y⟩)
6          end
7      end
8  end
9  if len(distanceCoupling)≥min_frequency:
     //若文件 X 和文件 Y 之间有“距离”的共变次数超过 min_frequency, 则认为存在耦合关系 X→Y.
10   then
11     return X→Y
12   end
```

(4) 基于潜在语义索引模型挖掘耦合关系(LSI-based method, LIM)

潜在语义索引(latent semantic indexing, LSI)是一种简单实用的主题模型. LSI 实现了动态聚类, 可以使用语义相似性生成文档组, 目前已被应用于估计 Web 文档中存在的多媒体对象之间的相似性[42]以及音乐推荐[43]等研究中.

假设: 本文认为: 如果文件之间在语义上有相似度, 那么两者可能存在耦合关系, 使用(X,Y)表示 LSI 得到的耦合关系是对称的.

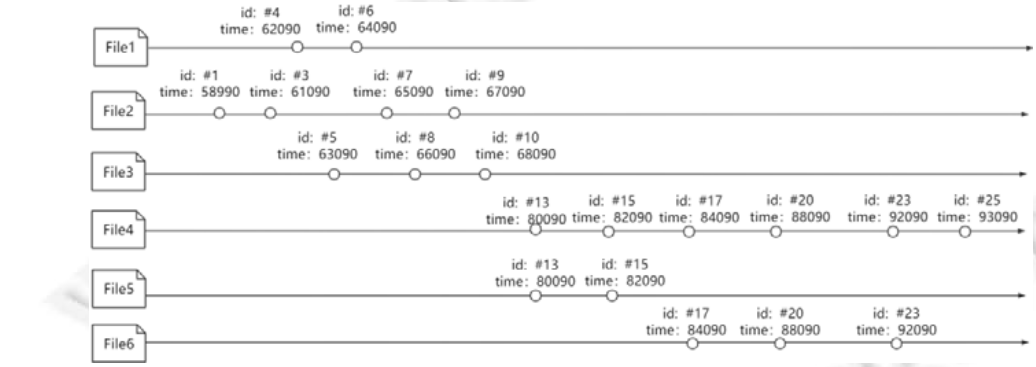
过程: 首先抽取源文件中的标识符, 为每个源文件生成对应的标识符文档, 所有的文档组成语料库. 然后设置主题个数 *num_topic*, 基于 LSI 模型得到话题向量的空间模型, 每一个标识符文档在话题空间中都由一个向量表示, 该向量的每维对应一个话题, 通过计算向量之间的相似度得到两个文件之间的语义相似度, 并通过设置相似度阈值 *min_similarity*, 过滤掉相似度小于阈值的文件对. 最终产生文件之间的语义相似值.

(5) 结果合并

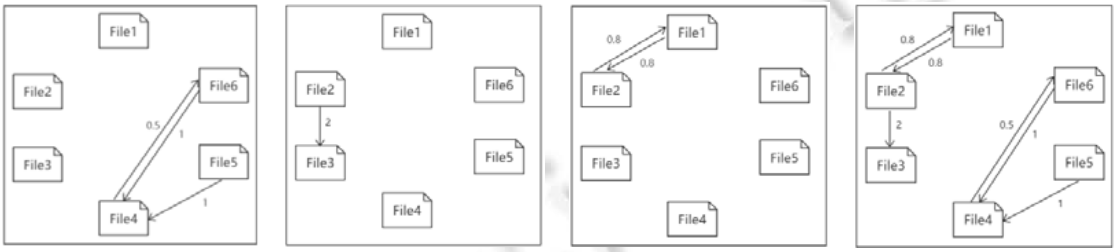
对于给定的目标文件, 选择使用关联规则方法得到的以目标文件为前项的规则; 如果没有, 则使用 MIM 得到的以目标文件为前项的规则. 如果前两种方法都挖掘不到, 则使用 LIM 得到的结果.

2.2 AR-MIM方法过程示例

本节通过一个案例展示 AR-MIM 的详细步骤和输出, 图 4(a)展示了软件演化历史中 6 个文件的变更提交情况, 其中, commit 的 id 用#*n* 表示, 同时, *n* 也代表了修订历史中 commit 的顺序, time 是 commit 发生的时间戳. 一共有 15 个 commit, commit#2、#11、#12、#14、#16、#18、#19、#21、#22、#24 没有展示, 因为这 6 个文件并没有在这些 commit 中发生变更. 以 File1 为例, 它在 commit#4 和 commit#6 中发生了变更.



(a) 文件的变更历史



(b) ARM 的结果 (c) MIM 的结果 (d) LIM 的结果 (e) AR-MIM 的结果

图 4 文件的变更历史及由不同演化耦合分析方法得到的结果

(1) 文件的变更历史抽取

图 4(a)展示了 6 个文件的变更历史: File1{#4,#6}、File2{#1,#3,#7,#9}、File3{#5,#8,#10}、File4{#13,#15,#17,#20,#23,#25}、File5{#13,#15}、File6{#17,#20,#23}.

(2) 基于关联规则挖掘演化耦合

考虑图 4(a)所示的例子, 以 File4 和 File5 为例. File4 在 6 个 commit 中发生了变更, File5 在 2 个 commit

中发生了变更. File4 和 File5 在 2 个相同的 commit 中发生了变更. 因此, 根据 *support* 和 *confidence* 的计算方法可得:

$$support(File4)=6,\ support(File5)=2,\ support(File4,File5)=2,$$
$$confidence(File4\rightarrow File5)=0.33,\ confidence(File5\rightarrow File4)=1.$$

同样地, File4 和 File6 也在相同 commit 中发生了变更, 因此有:

$$support(File4,File6)=3,\ confidence(File4\rightarrow File6)=0.5,\ confidence(File6\rightarrow File4)=1.$$

假设 *support* 和 *confidence* 的阈值分别为 2 和 0.5, 最终得到 3 个关联规则 File5→File4、File6→File4、File4→File6. 图 4(b)展示了基于关联规则方法得到的演化耦合关系.

(3) 基于情节挖掘演化耦合

考虑图 4(a)所示的例子. 假设 *max_width* 的阈值为 1, File1 在 commit#6 中发生了变更, File2 在 commit#7 中发生了变更, 可以得到 File1→File2 有一个 *width* 为 1 的共变(#6,#7); 类似地, File2→File1 有一个 *width* 为 1 的共变(#3,#4), File1→File3 有一个 *width* 为 1 的共变(#4,#5), File3→File1 有一个 *width* 为 1 的共变(#5,#6), File2→File3 有两个 *width* 为 1 的共变(#7,#8)和(#9,#10), File3→File2 有一个 *width* 为 1 的共变(#8,#9). 假设 *min_frequency* 的阈值为 2, 通过过滤掉不频繁的共变, 最终得到规则 File2→File3. 图 4(c)展示了这一结果.

(4) 基于潜在语义索引模型挖掘耦合关系

考虑图 4(a)所示的例子. 由于文件的语义信息占用大量篇幅, 在此不给出文件的语义信息. 设置 *num_topic*=3, *min_similarity*=0.6. 假设只有 File1 和 File2 相似度为 0.8, 大于 0.6. 图 4(d)展示了最终的结果.

(5) 结果合并

考虑图 4(a)所示的例子. 以 File4 为目标文件, 使用关联规则方法挖掘到规则 File4→File6, 因此, File4 和 File6 有演化耦合关系, 并不再考虑利用情节挖掘得到的以 File4 为目标文件的结果. 当 File2 为目标文件时, 使用关联规则方法没有得到它与其他文件的演化耦合关系, 因此, 选择情节挖掘的结果 File2→File3. 当以 File1 为目标文件时, 使用关联规则和情节挖掘都无法获得结果, 因此, 选择潜在语义索引模型得到的结果 File1→File2. 最终, 图 4(e)展示了结合关联规则、情节挖掘和潜在语义索引模型的演化耦合关系.

3 实验设置

本节介绍我们的研究问题、数据收集过程以及实验参数设置.

3.1 研究问题

- RQ1. 准确性分析: 与已有的演化耦合分析方法相比, AR-MIM 的准确性如何? 本文将在共同变更候选项预测的 ground truth 数据集上, 量化评估 ARM、AR-TRM、FOM、MIM、AR-MIM 这 5 种方法的演化耦合分析结果;
- RQ2. 对不同类别依赖的捕获能力分析: 与已有的演化耦合分析方法相比, AR-MIM 捕获代码层显式依赖与隐式依赖的效果如何? 通过研究这个问题, 本文将理解不同方法获取的耦合关系其指示的代码层显式依赖与隐式依赖的差异性;
- RQ3. 领域差异分析: 在不同领域的 Python 项目上, 评估 AR-MIM 方法与已有方法的准确性, 观察在不同领域上的方法性能是否存在差异;
- RQ4. 参数设置分析: 在 Python 项目上, 评估不同的参数设置对 AR-MIM 方法性能的影响.

3.2 数据收集

3.2.1 项目收集过程

本文在 Python 项目上进行了实验, 为了收集公开的 Python 数据集, 本文参考了 Jin 等人[44]和 Jebnoun 等人[45]的工作: 前者从 GitHub 上共收集到 499 个 Python 项目, 后者从 GitHub 上收集到 59 个 AI 领域相关的 Python 项目, 一共 558 个 Python 项目. 接下来, 我们从 558 个项目中筛选出本文研究的分析对象, 过滤掉不符

合条件的项目. 过滤标准包括:

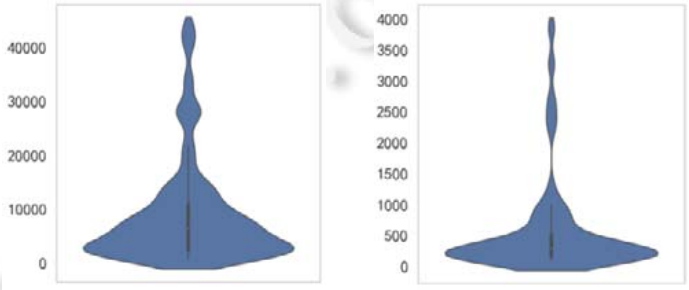
(1) 过滤掉修订历史提交数小于 800 的项目. 原因在于: 软件历史演化耦合分析需要分析项目具有一定规模且管理良好的修订历史记录(git commits), 足够的历史提交规模才能保证数据集的可靠性;

(2) 过滤掉提交历史记录中已修复问题(fixed issue)个数小于 100 的项目. 原因在于: 应用于变更影响预测实验时, 需要构建 ground truth 数据集, 为确保 ground truth 的准确性, 收集项目的修订历史中要包含足够数量的 fixedIssue 的提交.

根据以上步骤, 本文最终收集到 58 个 Python 项目作为研究对象. 表 2 总结了这 58 个 Python 项目的基本信息, 其中, #Commit 代表修订历史中的提交总数, #Issue 代表项目中 fixed Issue 数量. 从表 2 可以观察到: 这些项目的历史提交数量分布在 840–43661 条之间; fixedIssue 数量分布在 101–3864 之间. 图 5 可视化展示了这些信息的分布情况.

表 2 Python 项目规模信息

#Statistic	LOC	File	#Commit	#Issue
Min	6 133	30	840	101
Medium	6 9085	460	6 417	294
Max	521 092	3 238	43 661	3 864
Sum	6 258 092	39 464	530 270	33 169



(a) #commitNum (b) #IssueNum

图 5 Python 项目的提交历史和问题分布

参考 Jin 等人[44]标注的 Python 项目的领域, 本文统计这 58 个项目的领域, 结果见表 3. 例如: 表 3 中的第 1 行显示, 39.7% (58 个项目中有 23 个)属于 Scientific and Numeric 领域, 其中, 17 个是 AI 项目, 6 个是非 AI 项目. 本文收集的项目涵盖了各类不同的领域, 前三大应用领域包括 Scientific and Numeric (39.7%)、Software Development (27.6%)以及 Web and Internet Development (12.1%). 58 个项目中有 25 个 AI 项目和 33 个非 AI 项目.

表 3 Python 项目的领域分布

Domain	AI 项目数	非 AI 项目数	总和	占比(%)
Scientific and Numeric	17	6	23	39.7
Software Development	3	13	16	27.6
Web and Internet Development	3	4	7	12.1
Database Access	0	3	3	5.2
Image Processing and Computer	1	1	2	3.4
Game and 3D Graphics	1	0	1	1.7
Desktop GUIs	0	1	1	1.7
Network Programming	0	1	1	1.7
Education	0	1	1	1.7
Others	0	3	3	5.2

3.2.2 训练集和 ground truth 收集

对于每个项目, 本文从版本控制系统 Git 收集从项目开始到当前分析版本的所有修订历史记录. 参考之前的工作[46], 对于每个项目, 本文按比例将其修订历史分为两组, 分别为前 75%和后 25%: 前 75%修订历史用于生成训练集, 挖掘实体之间的演化耦合关系; 后 25%用于生成 ground truth 集合(GS), 验证训练集上挖掘

的演化耦合预测共同变更候选项的准确度. 参考 Kagdi 等人[29]和 Rolfsnes 等人[3]的工作, 在历史记录中只考虑修改源文件的提交, 且忽略修改源文件数量大于 30 的单次提交, 以排除组合提交、分支合并等因素的影响. 例如: GPflow 项目中, commit 927eda 中修改了 36 个.py 文件, 根据 commit message 描述, 该提交完成了两项工作, “Add types to the configuration, and remove unused imports”. 该提交属于组合提交, 需要忽略.

接下来详细介绍 GS 收集方法. 本文从后 25%修订历史中筛选出与修复 Issue 相关的提交的集合生成 GS. 具体而言, 本文首先使用 GitHub API 从 GitHub 上获取项目的 fixed Issue. 本文选择了 tag 为 closed 的 Issue, 并忽略了 tag 为 wontfix、duplicate、invalid、help wanted 的 Issue, 因为这些 Issue 并没有对代码进行有效的更改. 然后使用项目的后 25%的修订历史与 fixed Issue 匹配, 找出 fixed Issue 相关的提交. 具体的匹配方法是: 使用 git log 从 commit message 中找到相关的 issue id, 将找到的 issue id 与 GitHub 上报告的 fixed Issue 匹配. 将为修复 fixed Issue 而产生修改的文件两两生成文件对, 作为 GS, GS={⟨file1,file2⟩}. 其中, file1 和 file2 修复了同一 fixed Issue. 根据上述过程, 本文从 58 个项目中总共收集到 242 074 条 commit 作为训练集, 330 660 个 ground truth (https://github.com/zxylz/AR-MIM.git).

本文以 gensim 项目为例, 展示了 GS 的收集过程与结果. 表 4 展示了在 gensim 项目后 25%的修订历史中的一部分提交, 包括 commit Id、commit 修复的 Issue 以及 commit 中发生变更的文件. 如在 commit 2589c5ab 中, 修改了文件‘gensim/corpora/mmcorpus.py’‘gensim/utils.py’和‘setup.py’, 复了 Issue #3077, 解决了“various documentation warnings”问题.

表 4 gensim 项目部分提交信息

commit	fixed Issue	变更的文件
2589c5ab	#3077 (fix various documentation warnings)	‘gensim/corpora/mmcorpus.py’, ‘gensim/utils.py’, ‘setup.py’
cc441b7b	#2096 (fix documentation for various modules)	‘gensim/corpora/dictionary.py’, ‘gensim/corpora/hashdictionary.py’, ‘gensim/corpora/mmcorpus.py’, ‘gensim/corpora/wikicorpus.py’, ‘gensim/interfaces.py’, ‘gensim/matutils.py’, ‘gensim/models/phrases.py’, ‘gensim/models/tfidfmodel.py’, ‘gensim/similarities/docsim.py’, ‘gensim/sklearn_api/phrases.py’, ‘gensim/utils.py’
5355c065	#1910 (Refactor documentation for gensim.similarities.docsim)	‘gensim/corpora/mmcorpus.py’, ‘gensim/corpora/textcorpus.py’, ‘gensim/matutils.py’, ‘gensim/similarities/docsim.py’
c3f08c1a	#1825 (add cython version of MmReader)	‘gensim/__init__.py’, ‘gensim/corpora/mmcorpus.py’, ‘gensim/matutils.py’, ‘gensim/test/test_corpora.py’, ‘setup.py’

在 gensim 项目中, 本文得到 4 个 fixed Issue (见表 4). 将每个 fixed Issue 中修复的文件两两组合, 可以得到 74 个文件对. 当预测目标为 gensim/corpora/mmcorpus.py 时, 从 74 个文件对中选择包含目标文件 gensim/corpora/mmcorpus.py 的文件对来生成共同变更候选项的 GS, GS 为

{⟨gensim/corpora/mmcorpus.py,gensim/utils.py⟩,⟨gensim/corpora/mmcorpus.py,setup.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/corpora/dictionary.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/corpora/hashdictionary.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/corpora/wikicorpus.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/interfaces.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/matutils.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/models/phrases.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/models/tfidfmodel.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/sklearn_api/phrases.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/corpora/textcorpus.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/similarities/docsim.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/__init__.py⟩,⟨gensim/corpora/mmcorpus.py,gensim/test/test_corpora.py⟩}.

3.3 已有的演化耦合分析方法

(1) 基于关联规则的演化耦合分析方法 ARM

具体分析过程与第 2.1 节中的步骤(2)使用关联规则方法挖掘演化耦合相同.

(2) 关联规则与传递性关联规则相结合的演化耦合分析方法(association rules and transitive association rules-based method, AR-TRM)

Islam 等人[2]提出了传递性关联规则方法(transitive association rules, TRM), 并将其用于预测共同变更候选项. 并且, 他们经过研究证实, 关联规则与传递性关联规则方法的组合(AR-TRM)在预测共同变更候选项方面比单独的关联规则或单独的传递性关联规则方法表现得更好. 因此, 本文直接采用 Islam 等人[2]推荐的方法, 将关联规则和传递性关联规则方法的结果取并集. 其中, 基于关联规则的演化耦合分析方法已在第 2 节中给出介绍, 下面介绍基于传递性关联规则的演化耦合分析方法.

假设: 如果两个实体 *X* 和 *Z* 过去没有发生过共变, 但是 *X* 和 *Z* 分别经常与 *Y* 发生共变, 存在耦合关系, 那么 *X* 与 *Z* 之间也可能有耦合关系, 用 *X*→*Z* 表示. 使用传递性关联规则方法挖掘的结果也是不对称的, 因为传递性关联规则也使用了置信度来度量耦合强度.

过程: 使用算法 Apriori 挖掘 *X* 和 *Z* 与 *Y* 之间的关联规则, 得到规则 *X*→*Y* 和 *Y*→*Z*. 由 *X*→*Y*、*Y*→*Z* 可以得到 *X* 和 *Z* 之间的传递性关联规则 *X*→*Z*. *X*→*Z* 的置信度为

$$confidence(X\rightarrow Z)=confidence(X\rightarrow Y)*confidence(Y\rightarrow Z).$$

confidence 越高, 表明 *X* 与 *Z* 之间耦合关系越强. 通常设置 *confidence* 阈值过滤掉共同演化概率低的实体.

规则之间可以一直传递, 直到不满足 *confidence* 阈值. 例如: 如果挖掘出关联规则 *Z*→*W*, 可以继续由上述得出的传递性关联规则 *X*→*Z* 与关联规则 *Z*→*W* 得到 *X*→*W*.

示例: 考虑图 4(a)所示的例子. 在第 1.2 节中, 通过关联规则方法得到了 3 条规则 File5→File4、File6→File4、File4→File6. 根据文件的变更历史, File5 和 File6 并没有在相同 commit 中发生变更, 但是由规则 File5→File4 和 File4→File6 可得到传递性规则 File5→File6, 且有:

$$confidence(\text{File5}\rightarrow\text{File6})=confidence(\text{File5}\rightarrow\text{File4})*confidence(\text{File4}\rightarrow\text{File6})=0.5.$$

设置 *confidence* 阈值为 0.5, 最终得到一条传递性规则 File5→File6.

最终将关联规则和传递性关联规则方法的结果取并集, 图 6(a)展示了 AR-TRM 挖掘的演化耦合结果.

(3) 基于模糊重叠的演化耦合分析方法(fuzzy overlap-based method, FOM)

Kruizinga[19]提出了模糊重叠(fuzzy overlap)算法, 用于研究软件架构坏味道与共变的关系. Jin 等人[44]也使用模糊重叠算法挖掘共变, 探索显式和隐式依赖捕获共变的能力.

假设: 如果实体 *X* 变更前后的一段时间内, 实体 *Y* 经常发生变更, 那么实体 *X* 与 *Y* 之间可能存在耦合关系. 模糊重叠算法挖掘到的共变是一种“模糊”的共变, 得到的结果用(*X*,*Y*)表示. 与情节挖掘不同, 模糊重叠算法没有严格要求 *X* 和 *Y* 的变更发生的顺序. 此外, 与其他演化耦合分析方法不同, 模糊重叠算法得到的耦合关系是对称的, 如果得到(*X*,*Y*)有耦合关系, 那么(*Y*,*X*)同样有相同的耦合关系.

过程: 与 MIM 方法相似, 首先从修订历史记录得到实体 *X* 和 *Y* 的变更历史. 对于实体 *X* 变更历史中的一个 commit, 如果实体 *Y* 的变更历史中的某个 commit 在该提交前后发生, 且两个 commit 之间的距离和时间差不超过阈值 *CommitDistance* 和 *TimeDistance*, 则认为发生了一次“模糊”的共变, 其中, *CommitDistance* 是两个 commit 之间的提交数之差, *TimeDistance* 是两个 commit 发生的时间差. 最终找到实体 *X* 和实体 *Y* 之间所有的“模糊”的共变.

通过设置 *CommitDistance* 和 *TimeDistance*, 可以过滤掉延迟较长的共同变更. 如果文件只有一次“模糊”的共同变更, 则很容易归因于偶然. 因此, 还需要设置阈值 *MatchThreshold*, 过滤掉不经常共同变更的实体对.

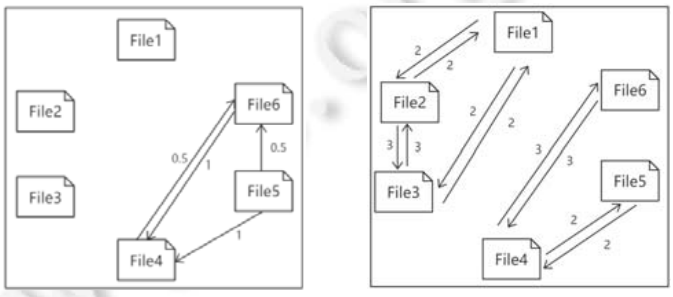
示例: 考虑图 4(a)所示的例子, 假设 *CommitDistance*=1, *TimeDistance*=1000, 可以得到耦合关系(File1, File2), 该文件对发生了两次“模糊”的共变(#4,#3)和(#6,#7), 即 File1 在#4 中发生变更前, File2 在#3 中发生过变更. File1 在#6 中发生变更后, File2 在#7 中发生过变更. 同样地, (File1,File3)发生了两次“模糊”的共变(#4,#5)和(#6,#5). (File2,File3)发生了 3 次模糊的共变(#7,#8)、(#9,#8)、(#9,#10). (File4,File5)发生了两次“模糊”的共变(#13,#13)、(#15,#15). (File4,File6)发生了 3 次模糊共变(#17,#17)、(#20,#20)、(#23,#23). 设置 *CommitDistance* 阈值为 1, *TimeDistance* 阈值为 1 000, *MatchThreshold* 阈值为 2, 过滤掉不经常共变的文件对, 最终得到存在耦合关系的文件对为(File1,File2)、(File1,File3)、(File2,File3)、(File4,File5)、(File4,File6). 图 6(b)展示了这一结果.

(4) 小结

上述 3 种方法和第 1 节中本文提出的方法均挖掘了不同类型的共变. ARM 挖掘同一提交中发生的共变, 认为过去经常发生共变的两个实体有耦合关系. 相比于 ARM, AR-TRM 还挖掘了“传递性”共变, 认为两个实体过去即使没有发生共变, 但是如果他们都与另一个实体有耦合关系, 那么他们也有耦合关系. AR-MIM 是 ARM、MIM 和 LIM 方法的结合, 除了挖掘到了同一提交中发生的共变, 还挖掘了有“距离”的共变, 认为: 如果一个实体变更后的一段提交内, 经常有另一个实体发生变更, 那么他们之间有耦合关系, 并且还包括了通过语义信息得到的耦合关系. FOM 挖掘“模糊”的共变, 认为: 如果一个实体变更前后的一段时间内, 经常有另一个实体发生变更, 那么他们之间有耦合关系.

针对图 4(a)展示的文件变更历史, 图 4(b) –图 4(e)和图 6(a)、图 6(b)展示了使用 5 种方法挖掘的不同的结果. 为了更清楚地展示 MIM 挖掘的结果, 将 MIM 方法挖掘的结果单独展示. 可以看出: ARM、TRM、MIM 挖掘到不同的耦合关系; FOM 挖掘了大量的耦合关系, 除了包含 ARM 和 MIM 的结果外, 还挖掘了很多其他的耦合关系.

因此, 本文将选择 ARM、AR-TRM、MIM、FOM 共 4 种方法作为基线方法, 所提出的 AR-MIM 方法将与这 4 种方法进行对比.



(a) AR-TRM 的结果 (b) FOM 的结果

图 6 不同演化耦合分析方法得到的结果

3.4 方法参数设置

对于提出的 5 个演化耦合分析方法, 本文为其设置参数如下.

- 对于 FOM 方法, 遵循原文[19]中推荐的最佳设置. *CommitDistance* 是通过查看修订历史记录中一天平均有多少提交来设置的, 该平均值忽略了未提交的天数. *TimeDistance* 通过查看修订历史记录中连续提交之间的时间间隔的第 3 个四分位数来设置的. 对于 *MatchThreshold*, 首先将 *MatchThreshold* 设置为 1 运行模糊重叠算法, 计算实体共变次数分布, 选择共变次数分布的第 95 个百分位作为 *Match Threshold* 阈值;
- ARM 有 *support* 和 *confidence* 两个参数. 参考 Mondal 等人[4]的工作, 将 *support* 设置为 2, 放弃了最低支持(*support*=1)规则, 因为根据现有的研究[47−49], 这些规则是最没有希望的规则, 通常放弃这些规则. *confidence* 阈值参考了 Mo 等人[50]的工作, 设置为 0.33, 以过滤掉弱耦合;
- TRM 只有一个 *confidence* 阈值, 参考原文[2], 设置为 0.7. 在该设置下, 预测共同变更候选项准确度最高. 因此, AR-TRM 中, ARM 阈值设置 *support* 为 2, *confidence* 为 0.33, TRM 的 *confidence* 为 0.7;
- 对于 MIM, 在本文实验中, 参考 FOM 方法中 *MatchThreshold* 的设置方法, 在将 *min_frequency* 设置为模糊重叠共变次数分布的第 80 个百分位、*max_width* 设置为 5 时效果较好;
- 对于 LIM, 参考相关的研究[1,29], 设置 *num_topic* 为 10, *min_similarity* 为 0.95.

最终, AR-MIM 方法中, ARM 阈值 *support* 为 2, *confidence* 为 0.33, MIM 和 LIM 按照上述方法设置.

3.5 评价指标

本文实验使用精确率(*P*)、召回率(*R*)、*F*1 分数(*F*1)来测量不同方法预测共同变更候选项的能力.

P: 预测正确共变在所有预测的共变中的占比;

R: 预测正确共变在 ground truth 中的占比;

$F1$: P 和 R 的调和平均数.

使用以下公式计算预测共同变更候选项的 P、R、$F1$:

$$P=\frac{|RS\cap GS|\times 100}{RS},\ R=\frac{|RS\cap GS|\times 100}{GS},\ F1=\frac{2\times P\times R}{P+R},$$

其中, GS 表示 ground truth 集合. 由于 ARM、AR-TRM、MIM、AR-MIM 挖掘的是有耦合关系的规则, 是不对称的, FOM 方法挖掘的是有耦合关系的文件对, 是对称的, 因此, 当使用 ARM、AR-TRM、MIM、AR-MIM 方法时, RS 表示演化耦合挖掘的以目标文件为前项的规则集合. 当使用 FOM 方法时, RS 表示演化耦合挖掘的包含目标文件的文件对集合.

4 实验结果

本节将说明 AR-MIM 与已有 4 种方法的实验对比过程和结果.

4.1 准确性分析(RQ1)

4.1.1 评估设置

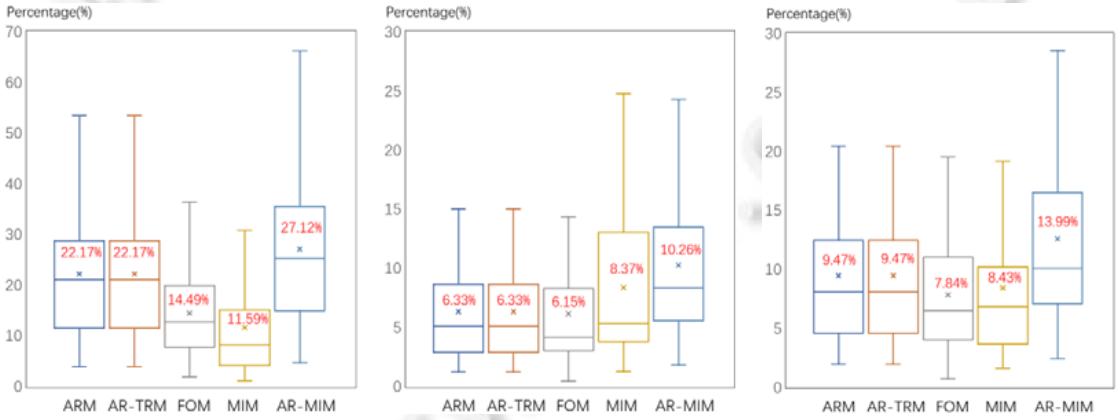
本实验使用第 3.5 节中介绍的精确率(P)、召回率(R)、$F1$ 分数($F1$), 以测量不同方法预测共同变更候选项的能力. P、R、$F1$ 值越大, 表明方法预测共同变更候选项的性能越好.

4.1.2 实验结果

图 7 显示了在 58 个 Python 项目上使用 5 种方法得到的 P、R 和 $F1$. 其中, AR-MIM 方法的 P、R 和 $F1$ 值比 ARM、FOM、AR-TRM 方法都要高. 本文进一步使用威尔科克森符号秩检验(Wilcoxon signed rank test)[51]来比较 AR-MIM 方法的 P、R、$F1$ 值是否显著大于其他方法的结果, 比较结果如下.

- P: AR-MIM 方法的 P 大于其他方法的 P, $P\text{-}value<0.01$;
- R: AR-MIM 方法的 R 大于其他方法的 R, $P\text{-}value<0.01$;
- $F1$: AR-MIM 方法的 $F1$ 大于其他方法的 $F1$, $P\text{-}value<0.01$.

以上 3 个结果表示方法的比较有统计学意义.



(a) P (b) R (c) $F1$

图 7 5 种方法预测共同变更候选项的结果

表 5 进一步显示了 AR-MIM 和几种方法的 P、R、$F1$ 均值之间的差值. Δ均值表示 AR-MIM 与几种方法差值的平均值. AR-MIM 方法的 P 值相比于 ARM 和 AR-TRM 提升了 4.95%, 相比于 FOM 提升了 12.63%, 相比于 MIM 提升了 15.53%. AR-MIM 方法的 P 值相比于这 4 种方法平均提升了 9.52%. R 值的提升范围是 1.89%–

4.11%, 平均提升了 3.47%. *F*1 的提升范围是 4.52%–6.15%, 平均提升了 5.19%. 通过威尔科克森符号秩检验, 得出上述比较的 *P-value* 小于 0.05, 表明上述观察具有统计显著性. 与 *P*、*R*、*F*1 均值的比较, 可以验证得出 AR-MIM 方法预测共同变更候选项的效果最好.

表 5 AR-MIM 与其他方法的比较结果(%)

	Δ((AR-MIM)−ARM)	Δ((AR-MIM)−(AR-TRM))	Δ((AR-MIM)−FOM)	Δ((AR-MIM)−MIM)	Δ均值
P	4.95	4.95	12.63	15.53	9.52
R	3.93	3.93	4.11	1.89	3.47
*F*1	4.52	4.52	6.15	5.56	5.19

为了进一步说明 AR-MIM 方法比 ARM 方法有提升的原因, 本文分析了 AR-MIM 和 ARM 预测结果的差异, 观察发现: AR-MIM 挖掘出的规则集合包含了 ARM 挖掘到的 19 117 条规则, 并且 AR-MIM 相比于 ARM 多挖掘到了很多条规则. 这表明, MIM 和 LIM 弥补了 ARM 不能挖掘到的耦合关系.

此外, 从图 7 观察到, AR-TRM 方法得到的 *P*、*R*、*F*1 均值和 ARM 方法得到的结果相差不大. 本文进一步对 TRM 方法的结果进行分析, 找到 AR-TRM 方法结果与 ARM 方法结果相差不大的原因. 表 6 展示了 58 个项目使用 TRM 方法挖掘到的规则数量, 其中, 58.62%的项目使用 TRM 方法挖掘不到规则, 39.66%的项目挖掘的规则少于 10 条, 只有一个项目挖掘到了 242 条规则. 将这些规则与 ground truth 对比, 发现只有 pyro 项目中有一条规则预测正确. 这说明, 对大多数 Python 项目而言, TRM 的方法是不适用的. 因此, AR-TRM 方法的 3 个指标与 ARM 方法几乎相同.

从图 7 可以观察到: 与 FOM 相比, ARM 方法对应的 *P*、*R*、*F*1 都更高. MIM 方法的 *P* 是最低的, 均值为 11.59%, 但是 *R* 较高, 均值达到了 8.37%.

表 6 TRM 方法挖掘的规则情况

挖掘出的规则数	项目数(比例)	挖掘出的规则数	项目数(比例)	挖掘出的规则数	项目数(比例)
0	34 (58.62%)	1	11 (18.97%)	2	4 (6.90%)
3	2 (3.45%)	4	4 (6.90%)	5	1 (1.72%)
7	1 (1.72%)	242	1 (1.72%)	–	–

4.1.3 小 结

与 ARM、FOM、AR-TRM、MIM 相比, 本文提出的 AR-MIM 方法在 *P*、*R*、*F*1 都表现得更好, 平均分别提升了 9.52%、3.47%、5.19%. 通过分析 ARM 和 AR-MIM 预测结果的差异, 发现 AR-MIM 比 ARM 多挖掘到了更多的规则. 这表明, MIM 和 LIM 可以弥补 ARM 不能挖掘到的耦合关系. 实验结果指出: 在获取同一个提交的共变关系基础上, 探索有“距离”共变关系和语义信息, 有助于提升演化耦合分析效果.

4.2 对不同类别依赖的捕获能力分析(RQ2)

显式依赖是源代码中显式显示的语法依赖关系. 隐式依赖是由于动态语言的特性导致分析时缺少类型信息而没有显式显示的语法依赖关系, 其依赖实体可能被解析为多个类型, 并且依赖关系只能在运行时唯一确定. Jin 等人[44]的工作在大量 Python 项目集上分析了显式和隐式依赖对软件不同分析任务的影响. 参考他们的工作, 本文实验将评估所提出的 AR-MIM 方法捕获显式和隐式依赖的能力, 并与基线方法进行对比.

4.2.1 评估设置

(1) 获取显式和隐式依赖

使用 ENRE[52]获取每个项目第 75%处提交版本的显式和隐式依赖, 分别记为 ED (explicit dependency)和 PD (possible dependency). 为了保证隐式依赖的准确性, 参考 Jin 等人[44]的工作, 本文同样使用可静态确定的隐式依赖, 即 *P*1 的结果.

(2) 评价指标

本文使用 *P*(*C*)进一步评估 RQ1 中 ARM、AR-TRM、FOM、MIM、AR-MIM 这 5 种方法得到的演化耦合(*C*)捕获显式和隐式依赖的能力, 具体计算方式如下:

$$P(C)=\frac{|RS\cap DS|\times 100}{DS},$$

其中, RS 表示 5 种方法得到的演化耦合; DS 表示显式或隐式依赖的结果; $P(C)$表示 5 种方法得到的演化耦合同时也在显式或隐式依赖中的百分比, $P(C)$值越大, 表示捕获显示或隐式依赖的能力越强.

4.2.2 实验结果

图 8 显示了捕获的 58 个 Python 项目的隐式依赖和显式依赖的信息, 显式依赖比隐式依赖更多. 显式依赖数量的均值为 1 435 个, 隐式依赖数量的均值为 520 个.

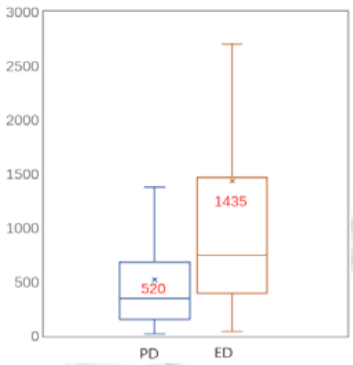
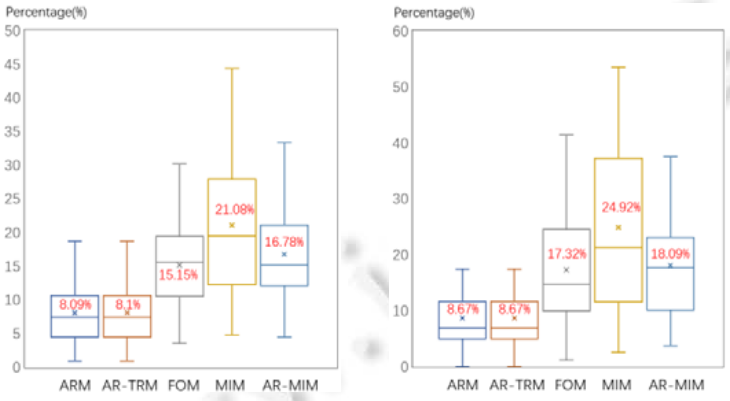


图 8 隐式依赖(PD)和显式依赖(ED)的数量

图 9(a)、图 9(b)分别展示了 5 种方法得到的演化耦合捕获显式和隐式依赖的 $P(C)$, 可以看出: MIM 捕获显式和隐式依赖的能力最强, 均值分别达到了 21.08%和 24.92%; ARM 捕获显式和隐式依赖的能力最差, 均值是 8.09%和 8.67%; AR-MIM 方法比 MIM 方法捕获显式和隐式依赖的比例更低一些, 均值是 16.78%和 18.09%. 其中一个原因是: 在 AR-MIM 方法的结果合并过程中(详见第 2.1 节的步骤(5)), 本文对 MIM 方法的结果作了选择.

此外, 对比图 9(a)和图 9(b)可以观察到: 相比于显式依赖, 5 种方法捕获隐式依赖的能力更强, 均值分别高 0.58%、0.57%、2.17%、3.84%、1.31%. 说明相比于显式依赖, 这几种方法得到的演化耦合捕获隐式依赖的能力更强.



(a) 显式依赖的 $P(C)$指标的结果 (b)隐式依赖的 $P(C)$指标的结果

图 9 不同方法捕获显式和隐式依赖的 $P(C)$指标的结果

4.2.3 小 结

结果表明: AR-MIM 挖掘的演化耦合捕获显式和隐式依赖的能力仅次于 MIM 方法, 捕获到的显式、隐式依赖的百分比均值分别是 16.78%和 18.09%; 此外, ARM 挖掘的演化耦合捕获显式和隐式依赖的能力最弱, 捕

获到的显式、隐式依赖的百分比均值分别是 8.09%和 8.67%; 5 种方法挖掘的共变捕获隐式依赖的能力相比于显式依赖都更强, 均值提高了 0.57%–3.84%.

4.3 领域差异分析(RQ3)

第 3.2.1 节中对参与实验的 58 个 Python 项目进行了领域分析, 发现项目集根据领域分布可以划分为两个子集: 一个数据集包括 25 个 AI 项目, 另一个数据集包括 33 个非 AI 项目. 因此, 本节在 RQ1 的基础上进一步对比在 AI 项目和非 AI 项目中不同演化耦合分析方法的准确性, 观察其是否存在差异. 若存在差异, 则进一步分析准确性存在差异的原因.

4.3.1 评估设置

(1) 对于 25 个 AI 和 33 个非 AI 项目, 比较 ARM、AR-TRM、FOM、MIM、AR-MIM 预测共同变更候选项的 P、R 和 $F1$;

(2) 评估 AI 和非 AI 项目在 ground truth 中新增文件的影响. 由于在后 25%修订历史记录中会出现新增的文件, 这些文件未在训练集中出现过, 因此无法预测这些文件的共同变更候选项, 这会导致 P、R 和 $F1$ 的降低. 因此, 本文统计了 AI 项目和非 AI 项目的 ground truth 中新增文件的比例, 并重新计算了排除新增文件后的 P、R 和 $F1$ 值;

(3) 对每个项目的修订历史记录进行了分析, 统计每次提交中变更的.py 文件的个数, 找出 AI 项目比非 AI 项目 P、R、$F1$ Score 低的可能原因.

4.3.2 实验结果

表 7 展示了对于 AI 项目和非 AI 项目(¬AI), AR-MIM 和几种方法的 P、R、$F1$ 均值之间的差值. Δ均值表示 AR-MIM 与几种方法差值的平均值. 对于 AI 项目, ARM 比其他 4 种方法的 P、R、$F1$ 平均提高了 8.25%、2.49%、4.1%; 对于非 AI 项目, ARM 比其他 4 种方法的 P、R、$F1$ 平均提高了 9.56%、4.12%、5.82%. 这表明: 对于 AI 和非 AI 项目, 本文提出的 AR-MIM 方法在预测共同变更候选项上都有更好的性能.

表 7 AR-MIM 与其他方法在 AI 和非 AI(¬AI)项目上的比较结果(%)

		Δ((AR-MIM)−ARM)	Δ((AR-MIM) −(AR-TRM))	Δ((AR-MIM)−FOM)	Δ((AR-MIM)−MIM)	Δ均值
AI	P	4.16	4.16	11.3	13.36	8.25
	R	2.31	2.31	3.33	2.01	2.49
	$F1$	3.13	3.13	5.31	4.83	4.1
¬AI	P	4.63	4.64	12.72	16.26	9.56
	R	5.07	5.07	4.61	1.71	4.12
	$F1$	5.38	5.38	6.59	5.91	5.82

表 8 展示了未排除新增文件时, 5 种演化耦合分析方法在 AI 和非 AI 项目上预测共同变更候选项的 P、R、$F1$ 的均值. Δ(¬AI−AI)表示非 AI(¬AI)项目和 AI 项目 P、R、$F1$ 的差值. 表中Δ(¬AI−AI)均为正值, 说明 AI 项目的 P、R、$F1$ 均比非 AI 项目的要低.

表 8 5 种方法在 AI 和非 AI(¬AI)项目上预测共同变更候选项的结果

		ARM (%)	AR-TRM (%)	FOM (%)	MIM (%)	AR-MIM (%)
P	AI	18.52	18.52	11.38	9.32	22.68
	¬AI	24.94	24.93	16.85	13.31	29.57
	Δ(¬AI−AI)	6.42	6.41	5.47	3.99	6.89
R	AI	4.98	4.98	3.96	5.28	7.29
	¬AI	7.35	7.35	7.81	10.71	12.42
	Δ(¬AI−AI)	2.37	2.37	3.85	5.43	5.13
$F1$	AI	7.68	7.68	5.5	5.98	10.81
	¬AI	10.82	10.82	9.61	10.29	16.2
	Δ(¬AI−AI)	3.14	3.14	4.11	4.31	5.39

由于 ground truth 新增的文件会导致 P, R 和 $F1$ 降低, 因此, 本文怀疑 AI 项目的 P, R, $F1$ 比非 AI 项目低的可能原因之一是, AI 项目 ground truth 中新增的文件比例比非 AI 项目中更高. 图 10 统计了每个项目的 ground truth 中新增的文件比例. AI 项目 ground truth 中新增的文件比例均值为 51.01%, 非 AI 项目为 37.57%. AI

项目 ground truth 中新增的文件比例更高, 说明相比于非 AI 项目, AI 项目更偏向开发而不是维护. 为了进一步验证这是可能的原因之一, 本文统计了排除新增文件后, 5 种方法的预测共同变更候选项结果.

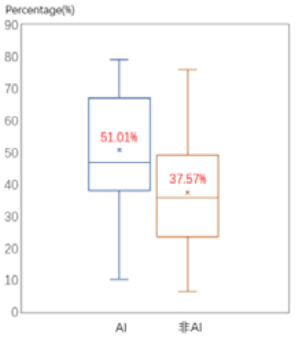


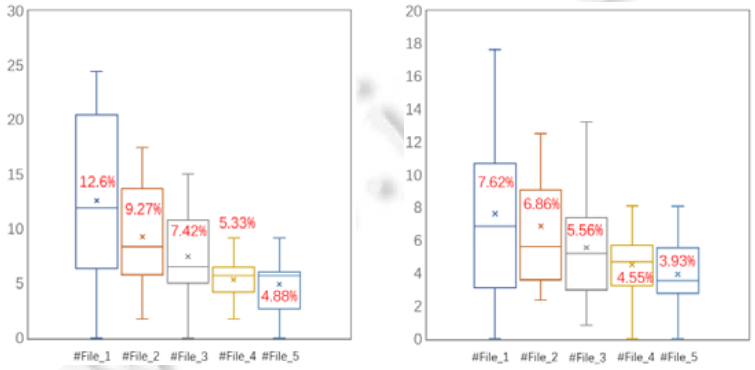
图 10 AI 和非 AI 项目 ground truth 中新增文件比例

表 9 展示了排除新增文件影响之后的结果, Δ(¬AI−AI)仍都大于 0. 将排除新增文件后的Δ(¬AI−AI)与未排除新增文件的Δ(¬AI−AI)比较发现: 对于 *P*, 5 种方法的Δ(¬AI−AI)降低了 0.69%−4.84%; 对于 *R*, ARM, AR-TRM, AR-MIM 的Δ(¬AI−AI)降低了 1.44%−1.96%; 对于 *F*1, 5 种方法的Δ(¬AI−AI)降低了 0.07%−3%. 即: 排除新增文件后, AI 项目和非 AI 项目预测结果之间的差距减小. 因此, 未排除新增文件是导致演化耦合方法在 AI 项目上表现不好的原因之一.

表 9 5 种方法在 AI 和非 AI(¬AI)项目预测共同变更候选项结果(排除新增文件后)

		ARM (%)	AR-TRM (%)	FOM (%)	MIM (%)	AR-MIM (%)
P	AI	36.49	36.49	21.86	17.34	44.67
	¬AI	39.9	39.9	26.64	20.59	46.72
	Δ(¬AI−AI)	3.41	3.41	4.78	3.25	2.05
R	AI	12.37	12.37	10.45	13.98	18.12
	¬AI	13.3	13.3	14.33	19.67	21.29
	Δ(¬AI−AI)	0.93	0.93	3.88	5.69	3.17
*F*1	AI	17.93	17.93	12.93	13.33	24.97
	¬AI	19.07	19.07	16.87	17.57	27.36
	Δ(¬AI−AI)	1.14	1.14	3.94	4.24	2.39

排除了新增文件后, AI 项目的 *P*、*R*、*F*1 仍低于非 AI 项目. 为了找到可能的原因, 本文进一步对 AI 和非 AI 项目的修订历史记录进行分析. 图 11 展示了修订历史中, 修改 1 到 5 个.py 文件的单次提交在所有提交次数中的占比. 可以观察到, AI 项目均高于非 AI 项目. 其中, AI 项目平均有 12.6%的提交只修改了一个文件, 非 AI 项目只有 7.62%. 这导致 AI 项目文件之间的耦合关系并不多, 这也是 AI 项目的 *P*、*R*、*F*1 Score 较低的原因之一.



(a) AI 项目 (b) 非 AI 项目

图 11 AI 和非 AI 项目单个提交修改.py 文件数量(#File_1−#File_5 分别代表修改 1−5 个.py 文件)的分布

4.3.3 小 结

在 AI 项目集上, 本文提出的 AR-MIM 方法比已有的 ARM、AR-TRM、FOM、MIM 方法的准确率、召回率、*F*1 分数分别提升了 8.25%、2.49%、4.1%. 此外, 所有这 5 种方法在 AI 项目上预测共同变更候选项的准确性均比在非 AI 项目上的性能偏低, 准确率、召回率、*F*1 分数分别降低了 3.99%–6.89%、2.37%–5.43%、3.14%–5.39%. 经过初步分析, 本文发现, AI 项目上预测共同变更候选项的准确性不高可能有两个原因.

- 一方面, AI 项目 ground truth 中 commit 新增文件的比例高达 51.01%, 而非 AI 项目的相应比例只有 37.57%. 说明相比于非 AI 项目, AI 项目的历史记录主要是新功能开发, 而不是已有文件的修改和维护, 因此导致效果产生偏差;
- 另一方面, AI 项目的单次提交中涉及的文件数量较少, 仅修改一个文件的提交占比高达 12.6%; 而非 AI 项目相应的比例为 7.62%. 导致缺乏足够共变训练集以获取文件之间的变更规则, 因此在 AI 项目上的效果不好.

因此, 对不同领域的数据集进行演化耦合分析, 相同方法可能表现出不同的性能, 建议未来结合更多的语义信息, 评估不同的主题模型, 以进一步提升演化耦合分析方法的效果.

4.4 参数设置分析(RQ4)

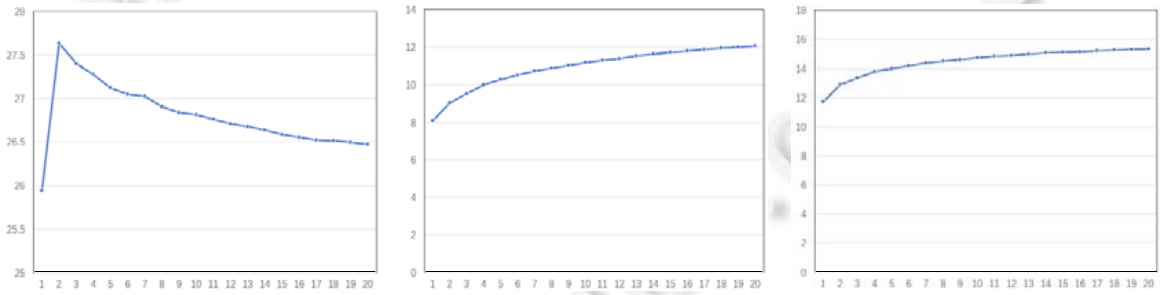
第 3.4 节介绍了 AR-MIM 方法的参数设置, 其中, 提交之间的距离 *width* 设置为定值 5. 为了进一步探索 *width* 的大小对 AR-MIM 方法性能的影响, 通过调整 *width* 的大小, 观察 AR-MIM 方法预测共同变更候选项的能力.

4.4.1 评估设置

对于 AR-MIM 方法, 将 *width* 分别设置成整数 1–20, 其他参数设置保持不变, 分别计算在 58 个 Python 项目上预测共同变更候选项的精确率(*P*)、召回率(*R*)、*F*1 分数(*F*1)的均值.

4.4.2 实验结果

图 12(a) –图 12(c)分别展示了随着 *width* 的逐渐增大, 58 个 Python 项目的 *P*、*R*、*F*1 均值的变化趋势. 对于 *P*, 当 *width* 为 2 时有最大值 27.64%; 当 *width* 大于 2 时, *P* 逐渐降低. 对于 *R* 和 *F*1, 在[1,20]的范围内, 随着 *width* 的增大而逐渐升高; 当 *width* 等于 20 时, *R*、*F*1 最高, 分别为 12.05%和 15.35%; 但是随着 *width* 的逐渐增大, *F*1 的提升速度逐渐变缓, 当 *width* 由 6 增加到 20 时, *F*1 的提升不足 2%.



(a) 精确率(*P*) (b) 召回率(*R*) (c) *F*1 分数(*F*1)

图 12 *width* 在[1,20]范围内, AR-MIM 的性能变化

4.4.3 小 结

对于 AR-MIM 方法, 设置 *width* 在[1,20]范围内, *P* 在 *width* 为 2 时有最大值; 随着 *width* 的增大, *R*、*F*1 逐渐升高并趋于稳定. 这表明, 通过调整 *width*, AR-MIM 方法的综合效果(*F*1 值)有一定提升, 并且在固定点之后趋于稳定.

5 相关工作

5.1 演化耦合分析方法

Zimmermann 等人[28,53−56]利用关联规则挖掘算法分析软件的演化历史, 以发现软件实体之间的演化耦合关系. 之后, 部分研究工作[4,46,57,58]改进了基于关联规则挖掘的演化耦合分析方法, 提高了分析结果的准确率. 例如: Bantelay 等人[46]结合交互历史和提交记录, 提出了 4 种组合模型; Mondal 等人[4]额外利用变更实体的标识符的相似性, 量化演化耦合的程度; Pugh 等人[59]动态地选择部分提交记录进行演化耦合分析. 一些研究还调查了新的度量指标, 如重要性[57]、模式年龄和模式距离[58], 用于量化演化耦合的程度. Mondal 等人[57]发现: 单个提交操作中涉及很多实体, 则该提交可能会包含不相关的更改. 因此, 使用重要性指标来评估实体的耦合程度. Alali 等人[58]发现, 使用数据挖掘技术挖掘演化耦合通常有很高的假阳性. 他们使用模式距离和模式年龄以进一步过滤噪音耦合.

上述基于关联规则的历史耦合分析方法只考虑了同一提交中的文件组作为可能的共同变更集合, 但是也有一些研究认为, 有些共变文件可能会在不同的历史提交中相继出现. Islam 等人[2]在提交的历史分析中引入了传递关联规则挖掘算法, 认为, 过去没有共同变化的实体也可能存在演化耦合的概率. 他们通过实验发现: 结合关联规则和传递性规则挖掘的演化耦合在预测共同变更候选项时, 具有更高的准确率和召回率. Ronald 等人[19]提出了一种名为 Fuzzy Overlap 的方法, 以挖掘“模糊”的共变并分析这些模糊的演化耦合与架构坏味道之间的关系.

与已有方法不同, 本文提出了一种结合关联规则、情节挖掘和潜在语义索引模型的方法, 即 AR-MIM, 可以挖掘出现有方法难以准确获取的有“距离”的历史耦合关系.

5.2 基于演化耦合的应用研究

已有研究将演化耦合分析方法[2−4]应用于预测共同变更候选项. Rolfsnes 等人[3]提出了 TARMAQ, 解决了看不见的查询问题, 即待预测的目标实体没有在之前变更历史中出现过或目标实体的组合没有在之前一起出现过. Mondal 等人[5]对演化耦合分析得到的共同变更候选项进行排序, 提出了一种基于历史的排名方法 HistoryRank. Agrawal 等人[6]通过改变历史提交的权重, 研究了变更历史年龄对共同变更预测结果的影响. 此外, 演化耦合也通常与代码克隆检测[7]、概念耦合[29]、结构耦合[60]、语义耦合[1]相结合, 预测共同变更候选项.

基于历史的演化耦合分析也被广泛应用于软件缺陷或故障的预测、定位和修复. D’Ambros 等人[12]的工作指出, 使用演化耦合信息可以提高缺陷预测的性能. Graves 等人[13]调研了模块更改历史的特征与软件故障的相关性. Kirbas 等人[15]研究了演化耦合对缺陷倾向性的影响. Sohn 等人[16]分析了测试代码和产品代码之间的演化耦合并应用于故障定位. Sohn 等人[17]利用提交历史的年龄等信息用于缺陷定位. Li 等人[61]表示, 代码变更是常见的软件缺陷预测度量. Mo 等人[62]分析了演化耦合文件的缺陷倾向性, 并检查了缺陷倾向性的可能原因. Sadiq 等人[63]在考虑最近和所有提交的情况下, 找出缺陷修复引发的变更和演化耦合关系之间的实际关系. Cui 等人[64]使用依赖级别的变更来分析缺陷修复, 并实现了自动检测缺陷修复中的依赖级别变更的原型工具.

基于历史的演化耦合分析还可以应用于架构问题的检测和架构恢复. 已有工作研究了演化耦合对架构缺陷检测的影响[20]以及与架构坏味道的演变之间的相关性[21]. Kruizinga[19]发现, 历史共同变更关系先于架构坏味道的引入. Khomh 等人探索了代码坏味道与软件变更倾向之间的关系[65]以及反模式与软件变更倾向之间的关系[22]. Kouroshfar 等人[18]使用演化历史构建了 3 种类型的架构衰退预测模型: 架构缺陷、架构气味和模块化质量. Saydemir 等人[66]将演化耦合作为架构恢复的补充信息源, 使架构恢复的准确性提高了 40%.

基于历史的演化耦合分析还被应用于识别软件系统的设计弱点和模块化问题, 从而加以重构. Prajapati 等人[67]使用演化耦合等信息重构面向对象的软件系统. Rathee 等人[68]利用变更历史等信息提高模块化标准的有效性. Prajapati 等人[69]在研究工作中使用了演化耦合等信息模块化软件系统.

此外, 基于修订历史的演化耦合分析还被广泛应用于横切关注点[23−25]、特征定位[70]等研究. 可见, 演化

耦合分析是一个基础性的工作, 其结果的有效性将严重影响下游应用分析的可靠性与性能. 因而, 本文提出的演化耦合方法旨在提高历史耦合分析的准确率.

5.3 Python软件上的相关研究

Python 被广泛应用于机器学习和深度学习模型的开发. 很多研究工作针对人工智能软件进行分析. 已有工作对 Python 项目的代码坏味道[45]、性能缺陷[71]、特定重构推荐[72]等进行了分析. Jebnoun 等人[45]研究了深度学习应用程序的代码质量, 调查了代码坏味道在深度学习应用程序中的分布. Cao 等人[71]分析了深度学习系统中的性能缺陷. Côté 等人[73]确定了与机器学习软件系统质量差相关的不良实践. Tambon 等人[74]研究了深度学习框架中的行为错误及其对客户端代码的影响. Tang 等人[72]研究了与机器学习系统相关的特定重构, 制定了机器学习系统中重构的层次分类法. 然而, 目前仍缺乏在 Python 项目中的演化耦合分析研究. 本文对 Python 项目(包括 AI 领域的项目)进行了演化耦合分析, 初步探索了两者演化耦合分析效果存在差异的原因.

由于 Python 语言的动态类型特性, 实体之间存在显式依赖和隐式依赖. Jin 等人探索了由于在 Python 语言中缺乏类型信息而可能导致的隐式依赖关系对体系结构级可维护性的影响[75], 通过使用类型提示实践改进了隐式依赖提取, 并且他们还比较了不同类型依赖捕获共变的能力, 增加了对架构反模式影响的调查[44]. 然而, 已有工作缺乏对利用历史中的共变信息来捕获实体之间依赖能力的评估. 本文利用多种演化耦合分析方法获取实体的历史共变信息, 并利用共变信息捕获显式和隐式依赖, 进一步对比了不同分析方法捕获不同类型依赖的能力.

6 结果讨论

- 方法阈值和预测目标讨论

本文方法与已有方法均涉及阈值设置, 在本文实验中, 对不同方法的参数设置采用了已有文献工作推荐的最佳设置. 我们认为, 这样的设置是公平且客观的. 然而, 不同的阈值可能会影响方法的性能, 未来我们将探讨本文方法对参数设置的敏感性. 其次, 现有研究挖掘了由两个以上文件组成的关联规则, 本文挖掘了由两个文件组成的二元规则, 且只是对单独的一个目标文件预测其共同变更候选项. 但是, 本文的方法可以拓展为一组目标文件预测共同变更候选项. 当建议一组目标方法的共同变更候选项时, 可以为每个目标方法生成规则, 将所有规则取并集作为一组目标方法的共同变更候选项. 这也是本文未来工作的一部分.

- 开发实践应用讨论

为了进一步验证本文方法的应用价值, 我们尝试着对 Github 开源项目中没有彻底解决的 Issue(即 open Issue)提出可能的 Issue 修复建议. 即: 对于 Github 中这些 Issue, 使用 AR-MIM 给出潜在的需要修改的文件集合. 这一工作选择了 Github 的一个开源项目 ENRE-PY 中 open Issue 的信息, 这个 open Issue 之前被修复过, 但是没有修复成功. 表 10 列出了对这个 Issue 的描述、AR-MIM 推荐的可能需要修改的文件列表以及经过开发者确认的真正需要修改的文件. 例如, 表 10 第 1 行显示: 对于 Issue#4 (“The analysis of project “sympy” has no results without stoping”), 开发者之前修改了文件 enre/analysis/env.py, 但却没有修复成功. 为了进一步修复 Issue#4, AR-MIM 推荐了接下来可能需要修改的 9 个文件. 其中, 文件 enre/analysis/analyze_manager.py 和 enre/analysis/analyze_expr.py 确实在开发者确认的真正需要修改的文件中.

表 10 ENRE-PY 项目中的 open Issue 的信息

Issue	已修改文件	AR-MIM 推荐的待修改文件	经过开发者确认的真正需要修改的文件
#4 (“The analysis of project “sympy” has no results without stoping”)	enre/analysis/env.py	enre/analysis/analyze_manager.py, enre/analysis/analyze_expr.py, enre/ent/EntKind.py,enre/ent/entity.py, enre/passes/entity_pass.py, enre/analysis/assign_target.py, enre/analysis/analyze_stmt.py, enre/analysis/value_info.py,enre/ref/Ref.py	enre/analysis/analyze_manager.py, enre/analysis/analyze_expr.py

- 在不同编程语言上进行方法泛化能力分析

本文的研究针对有限数量的 Python 项目. 虽然这些收集的 Python 项目具有不同的规模、领域和修订历史长度, 来自最近的工作[44,45], 但不能断言本文的结论可以适用于所有项目, 尤其是用不同的编程语言编写的项目. 因此, 又在开源的 Java 语言和 C 语言项目上对方法的性能进行评估. 目前, 在一个 C 语言和一个 Java 语言项目上评估了不同方法的准确度, 结果见表 11. 对于 Java 项目 spring-integration, AR-MIM 方法相比于其他方法没有明显提升; 但对于 C 语言项目 haproxy, AR-MIM 方法明显好于其他方法.

表 11 Java 语言和 C 语言项目上演化耦合分析结果

项目名	语言		ARM (%)	AR-TRM (%)	FOM (%)	MIM (%)	AR-MIM (%)
spring-integration	Java	*P*	33.40	33.40	45.45	11.79	34.30
		R	62.73	62.73	44.03	73.90	64.81
		*F*1	43.59	43.59	44.73	20.34	44.86
haproxy	C	*P*	29.43	29.43	15.48	12.88	32.51
		R	21.62	21.62	21.19	24.79	38.19
		*F*1	24.93	24.93	17.89	16.95	35.12

- 应用场景或领域分析

在软件供应链应用中, 由于第三方组件依赖及代码复用可能导致上游的漏洞被引入到下游的组件及应用软件中, 已有研究工作[8,9]将软件历史分析应用到软件供应链领域. 本文挖掘了单个项目中的演化耦合关系, 未来将对不同项目之间的演化耦合进行分析, 利用软件和第三方组件的耦合关系, 帮助检测可能引入的不安全的代码文件. 现有工作[76]也表明: 针对新涌现的领域, 如人工智能, 当前对其第三方组件及应用软件的风险识别需求紧迫, 人工智能平台和模型复用第三方组件存在安全风险. RQ3 也初步发现了不同领域的演化耦合分析性能的差异, 未来将结合更多的语义信息评估不同的主题模型, 以进一步提升方法的性能.

7 总 结

本文提出了一种有“距离”的演化耦合分析方法 AR-MIM, 旨在从软件修订历史中挖掘出频繁发生共同变更的文件组. AR-MIM 方法融合了关联规则挖掘、情节挖掘、语义挖掘得到的特征, 从软件维护记录中分析出文件之间的演化耦合关系. 在 58 个 Python 项目数据集上, 将提出的方法与已有的 4 种代表性的基线方法进行比较, 结果发现: 在预测共同变更候选项的应用上, AR-MIM 的准确率、召回率、*F*1 值相比于已有 4 种方法, 平均提高了 9.52%、3.47%、5.19%. 此外, 本文分析了不同演化耦合分析方法捕获代码中显式依赖和隐式依赖的能力差异, 指出本文提出的 AR-MIM 方法结合了情节挖掘和潜在语义的优势. 最后, 实验结果表明: 相同方法在不同领域项目上的性能表现存在差异, 并分析出可能的原因以及潜在的改进方向.

References:

[1] Kagdi H, Gethers M, Poshyvanyk D. Integrating conceptual and logical couplings for change impact analysis in software. Empirical Software Engineering, 2013, 18(5): 933−969. [doi: 10.1007/s10664-012-9233-9]

[2] Islam MA, Islam MM, Mondal M, Roy B, Roy CK, Schneider KA. Detecting evolutionary coupling using transitive association rules. In: Schneider KA, ed. Proc. of the 18th Int’l Working Conf. on Source Code Analysis and Manipulation (SCAM). Madrid: IEEE, 2018. 113−122. [doi: 10.1109/SCAM.2018.00020]

[3] Rolfsnes T, Di Alesio S, Behjati R, Moonen L, Binkley DW. Generalizing the analysis of evolutionary coupling for software change impact analysis. In: Binkley DW, ed. Proc. of the 23rd Int’l Conf. on Software Analysis, Evolution, and Reengineering (SANER), Vol.1. Osaka: IEEE, 2016. 201−212. [doi: 10.1109/SANER.2016.101]

[4] Mondal M, Roy B, Roy CK, Schneider KA. ID-correspondence: A measure for detecting evolutionary coupling. Empirical Software Engineering, 2021, 26(1): 1−34. [doi: 10.1007/s10664-020-09921-9]

[5] Mondal M, Roy B, Roy CK, Schneider KA. HistoRank: History-based ranking of co-change candidates. In: Proc. of the 27th IEEE Int’l Conf. on Software Analysis, Evolution and Reengineering (SANER). London: IEEE, 2020. 240−250. [doi: 10.1109/SANER48275.2020.9054869]

[6] Agrawal A, Singh RK. Predicting co-change probability in software applications using historical metadata. IET Software, 2020, 14(7): 739−747

[7] Mondal M, Roy B, Roy CK, Schneider KA. Associating code clones with association rules for change impact analysis. In: Proc. of the 27th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). London: IEEE, 2020. 93−103. [doi: 10.1109/SANER48275.2020.9054846]

[8] Wan LY. Automated vulnerability detection system based on commit messages [MS. Thesis]. Nanyang Technological University, 2019. https://dr.ntu.edu.sg/handle/10220/48651

[9] Gonzalez D, Zimmermann T, Godefroid P, Schäfer M. Anomalicious: Automated detection of anomalous and potentially malicious commits on github. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Madrid: IEEE, 2021. 258−267. [doi: 10.1109/ICSE-SEIP52600.2021.00035]

[10] Steff M, Russo B. Co-evolution of logical couplings and commits for defect estimation. In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories (MSR). Zurich: IEEE, 2012. 213−216. [doi: 10.1109/MSR.2012.6224283]

[11] Tantithamthavorn C, Ihara A, Matsumoto KI. Using co-change histories to improve bug localization performance. In: Proc. of the 14th ACIS Int'l Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. Honolulu: IEEE, 2013. 543−548. [doi: 10.1109/SNPD.2013.92]

[12] D'Ambros M, Lanza M, Robbes R. On the relationship between change coupling and software defects. In: Proc. of the 16th Working Conf. on Reverse Engineering. Lille: IEEE, 2009. 135−144. [doi: 10.1109/WCRE.2009.19]

[13] Graves TL, Karr AF, Marron JS, Harvey S. Predicting fault incidence using software change history. IEEE Trans. on Software Engineering, 2000, 26(7): 653−661. [doi: 10.1109/32.859533]

[14] Knab P, Pinzger M, Bernstein A. Predicting defect densities in source code files with decision tree learners. In: Proc. of the 2006 Int'l Workshop on Mining Software Repositories. New York: Association for Computing Machinery, 2006. 119−125. [doi: 10.1145/1137983.1138012]

[15] Kirbas S, Sen A, Caglayan B, Bener A, Mahmutogullari R. The effect of evolutionary coupling on software defects: an industrial case study on a legacy system. In: Proc. of the 8th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. New York: Association for Computing Machinery, 2014. 1−7. [doi: 10.1145/2652524.2652577]

[16] Sohn J, Papadakis M. CEMENT: On the use of evolutionary coupling between tests and code units. A case study on fault localization. In: Proc. of the 33rd IEEE Int'l Symp. on Software Reliability Engineering. 2022. 133−144.

[17] Sohn J, Yoo S. Empirical evaluation of fault localisation using code and change metrics. IEEE Trans. on Software Engineering, 2019, 47(8): 1605−1625. [doi: 10.1109/TSE.2019.2930977]

[18] Kouroshfar E. Studying the effect of co-change dispersion on software quality. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). San Francisco: IEEE, 2013. 1450−1452. [doi: 10.1109/ICSE.2013.6606741]

[19] Sas D, Avgeriou P, Kruizinga R, Scheedler R. Exploring the relation between co-changes and architectural smells. SN Computer Science, 2021, 2(1): 1−15. [doi: 10.1007/s42979-020-00407-5]

[20] Zakurdaeva AV. Using machine learning to detect architectural integrity violations associated with bugs [Ph.D. Thesis]. Ottawa: Carleton University, 2021.

[21] Sas D, Avgeriou P, Pigazzini I, Arcelli Fontana F. On the relation between architectural smells and source code changes. Journal of Software: Evolution and Process, 2022, 34(1): e2398.

[22] Khomh F, Penta MD, Guéhéneuc YG, Antoniol G. An exploratory study of the impact of antipatterns on class change-and fault-proneness. Empirical Software Engineering, 2012, 17(3): 243−275. [doi: 10.1007/s10664-011-9171-y]

[23] Breu S, Zimmermann T. Mining aspects from version history. In: Proc. of the 21st IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2006). Tokyo: IEEE, 2006. 221−230. [doi: 10.1109/ASE.2006.50]

[24] Eaddy M, Zimmermann T, Sherwood KD, Garg V, Murphy GC, Nagappan N, Aho AV. Do crosscutting concerns cause defects. IEEE Trans. on Software Engineering, 2008, 34(4): 497−515. [doi: 10.1109/tse.2008.36]

[25] Bram A, Ahmed EH. Identifying crosscutting concerns using historical code changes. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering (ICSE 2010). New York: IEEE, 2010. 305−314.

[26] Shen B, Zhang W, Kästner C, Zhao H, Wei Z, Liang G, Jin Z. SmartCommit: A graph-based interactive assistant for activity-oriented commits. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. New York: Association for Computing Machinery, 2021. 379–390. [doi: 10.5281/zenodo.5111654]

[27] Yamashita S, Hayashi S, Saeki M. Changebeadsthreader: An interactive environment for tailoring automatically untangled changes. In: Proc. of the 27th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). London: IEEE, 2020. 657–661. [doi: 10.1109/SANER48275.2020.9054861]

[28] Zimmermann T, Zeller A, Weissgerber P, Diehl S. Mining version histories to guide software changes. IEEE Trans. on Software Engineering, 2005, 31(6): 429–445. [doi: 10.1109/tse.2005.72]

[29] Kagdi H, Gethers M, Poshyvanyk D, Collard ML. Blending conceptual and evolutionary couplings to support change impact analysis in source code. In: Proc. of the 17th Working Conf. on Reverse Engineering. Beverly: IEEE, 2010. 119–128.

[30] Kaczorowski M. Secure at every step: What is software supply chain security and why does it matter. https://github.blog/2020-09-02-secure-your-software-supply chain-and-protect-against-supply-chain-threats-github-blog/

[31] Kumar A, Gupta A. Evolution of developer social network and its impact on bug fixing process. In: Proc. of the 6th India Software Engineering Conf. New York: Association for Computing Machinery, 2013. 63–72.

[32] Kumar A, Gupta A. Evolution of developer social network and its impact on bug fixing process. In: Proc. of the 6th India Software Engineering Conf. New York: Association for Computing Machinery, 2013. 63–72.

[33] Kumar A, Desai Y, Gandhi M, Agarwal S. Studying multifaceted collaboration of OSS developers and its impact on their bug fixing performance. In: Proc. of the 7th Int'l Workshop on Quantitative Approaches to Software Quality. IEEE, 2019. [doi: 10.1145/19856793.12455832]

[34] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. In: Proc. of the 1993 ACM SIGMOD Int'l Conf. on Management of Data. New York: IEEE, 1993. 207–216.

[35] Canfora G, Cerulo L, Di Penta M. On the use of line co-change for identifying crosscutting concern code. In: Proc. of the 22nd IEEE Int'l Conf. on Software Maintenance. Philadelphia: IEEE, 2006. 213–222. [doi: 10.1109/ICSM.2006.43]

[36] Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proc. of the 20th Int'l Conf. on Very Large Data Bases, Vol.1215. New York: VLDB, 1994. 487–499.

[37] Mannila H, Toivonen H, Inkeri Verkamo A. Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery, 1997, 1(3): 259–289.

[38] Amiri M, Mohammad-Khanli L, Mirandola R. An online learning model based on episode mining for workload prediction in cloud. Future Generation Computer Systems, 2018, 87: 83–101. [doi: 10.1016/j.future.2018.04.044]

[39] Sellami C, Miranda C, Samet A, Bach MA. On mining frequent chronicles for machine failure prediction. Journal of Intelligent Manufacturing, 2020, 31(4): 1019–1035. [doi: 10.1007/s10845-019-01492-x]

[40] Hatonen K, Klemettinen M, Mannila H, Ronkainen P, Toivonen H. Knowledge discovery from telecommunication network alarm databases. In: Proc. of the 12th Int'l Conf. on Data Engineering. New Orleans: IEEE, 1996. 115–122.

[41] Cheng W. Research on key technologies of time series data mining for large-scale network security situation analysis [Ph.D. Thesis]. Changsha: National University of Defense Technology, 20l0 (in Chinese with English abstract).

[42] Srikanth D, Sakthivel S. Vantage point latent semantic indexing for multimedia Web document search. Cluster Computing, 2019, 22(5): 10587–10594. [doi: 10.1007/s10586-017-1135-6]

[43] Yürekli A, Kaleli C, Bilge A. Alleviating the cold-start playlist continuation in music recommendation using latent semantic indexing. Int'l Journal of Multimedia Information Retrieval, 2021, 10(3): 185–198.

[44] Jin W, Zhong D, Cai Y, Kazman R. Evaluating the impact of possible dependencies on architecture-level maintainability. IEEE Trans. on Software Engineering, 2022, 49(3): 1064–1085.

[45] Jebnoun H, Ben Braiek H, Rahman MM, Khomh F. The scent of deep learning code: An empirical study. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. New York: Association for Computing Machinery, 2020. 420–430. [doi: 10.1145/3379597.3387479]

[46] Bantelay F, Zanjani MB, Kagdi H. Comparing and combining evolutionary couplings from interactions and commits. In: Proc. of the 20th Working Conf. on Reverse Engineering (WCRE). Koblenz: IEEE, 2013. 311–320.

[47] Ying ATT, Murphy GC, Ng R, Chu-Carrol MC. Predicting source code changes by mining change history. IEEE Trans. on Software Engineering, 2004, 30(9): 574−586. [doi: 10.1109/tse.2004.52]

[48] Bavota G, Dit B, Oliveto R, *et al*. An empirical study on the developers' perception of software coupling. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). IEEE, 2013. 692−701. [doi: 10.1109/icse.2013.6606615]

[49] Kotsiantis S, Kanellopoulos D. Association rules mining: A recent overview. GESTS Int'l Trans. on Computer Science and Engineering, 2006, 32(1): 71−82.

[50] Mo R, Zhan M. History coupling space: A new model to represent evolutionary relations. In: Proc. of the 26th Asia-Pacific Software Engineering Conf. (APSEC). Putrajaya: IEEE, 2019. 126−133. [doi: 10.1109/APSEC48747.2019.00026]

[51] Kaur A, Kumar R. Comparative analysis of parametric and non-parametric tests. Journal of Computer and Mathematical Sciences, 2015, 6(6): 336−342.

[52] Jin W, Cai Y, Kazman R, Zheng Q, Cui D, Liu T. ENRE: A tool framework for extensible eNtity relation extraction. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. (ICSE-Companion). Montreal: IEEE, 2019. 67−70. [doi: 10.1109/ICSE-Companion.2019.00040]

[53] D'Ambros M, Lanza M. Reverse engineering with logical coupling. In: Proc. of the 13th Working Conf. on Reverse Engineering. Benevento: IEEE, 2006. 189−198. [doi: 10.1109/WCRE.2006.51]

[54] Hanakawa N. Visualization for software evolution based on logical coupling and module coupling. In: Proc. of the 14th Asia-Pacific Software Engineering Conf. (APSEC 2007). Nagoya: IEEE, 2007. 214−221. [doi: 10.1109/ASPEC.2007.36]

[55] Ahsan SN, Wotawa F. Fault prediction capability of program file's logical-coupling metrics. In: Proc. of the Joint Conf. of the 21st Int'l Workshop on Software Measurement and the 6th Int'l Conf. on Software Process and Product Measurement. Nara: IEEE, 2011. 257−262. [doi: 10.1109/IWSM-MENSURA.2011.38]

[56] Ali N, Jaafar F, Hassan AE. Leveraging historical co-change information for requirements traceability. In: Proc. of the 20th Working Conf. on Reverse Engineering (WCRE). Koblenz: IEEE, 2013. 361−370.

[57] Mondal M, Roy CK, Schneider KA. Improving the detection accuracy of evolutionary coupling. In: Proc. of the 21st Int'l Conf. on Program Comprehension (ICPC). San Francisco: IEEE, 2013. 223−226.

[58] Alali A, Bartman B, Newman CD, Maletic JI. A preliminary investigation of using age and distance measures in the detection of evolutionary couplings. In: Proc. of the 10th Working Conf. on Mining Software Repositories (MSR). San Francisco: IEEE, 2013. 169−172. [doi: 10.1109/MSR.2013.6624024]

[59] Pugh S, Binkley D, Moonen L. The case for adaptive change recommendation. In: Proc. of the 18th Int'l Working Conf. on Source Code Analysis and Manipulation (SCAM). Madrid: IEEE, 2018. 129−138.

[60] Shen B, Zhang W, Yu A, Wei Z, Liang G, Zhao H, Jin Z. Cross-language code coupling detection: A preliminary study on Android applications. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution (ICSME). Luxembourg: IEEE, 2021. 378−388. [doi: 10.1109/ICSME52107.2021.00040]

[61] Li Z, Jing XY, Zhu X. Progress on approaches to software defect prediction. IET Software, 2018, 12(3): 161−175.

[62] Mo R, Yin Z. Exploring software bug-proneness based on evolutionary clique modeling and analysis. Information and Software Technology, 2020, 128: 106380. [doi: 10.1016/j.infsof.2020.106380]

[63] Sadiq AZ, Mostafa MJI, Sakib K. On the evolutionary relationship between change coupling and fix-inducing changes. In: Proc. of the ENASE. Madrid: IEEE, 2019. 494−501.

[64] Cui D, Fan L, Chen S, Cai Y, Zheng Q, Liu Y, Liu T. Towards characterizing bug fixes through dependency-level changes in Apache Java open source projects. Science China Information Sciences, 2022, 65(7): 1−19. [doi: 10.1007/s11432-020-3317-2]

[65] Khomh F, Di Penta M, Gueheneuc YG. An exploratory study of the impact of code smells on software change-proneness. In: Proc. of the 16th Working Conf. on Reverse Engineering. Lille: IEEE, 2009. 75−84. [doi: 10.1109/WCRE.2009.28]

[66] Saydemir A, Simitcioglu ME, Sozer H. On the use of evolutionary coupling for software architecture recovery. In: Proc. of the 15th Turkish National Software Engineering Symp. (UYMS). IEEE, 2021. 1−6.

[67] Prajapati A, Parashar A, Chhabra JK. Restructuring object-oriented software systems using various aspects of class information. Arabian Journal for Science and Engineering, 2020, 45(12): 10433−10457. [doi: 10.1007/s13369-020-04785-z]

[68] Rathee A, Chhabra JK. Clustering for software remodularization by using structural, conceptual and evolutionary. Journal of Universal Computer Science, 2018, 24(12): 1731−1757.

[69] Prajapati A, Parashar A, Rathee A. Multi-dimensional information-driven many-objective software remodularization approach. Frontiers of Computer Science, 2023, 17(3): 1−18. [doi: 10.1007/s11704-022-1449-2]

[70] Lukas S. Visualizing feature coupling evolution by utilizing source code co-change and issue tracking data [Ph.D. Thesis]. Wien: Unterschrift Verfasser, 2021.

[71] Cao J, Chen B, Sun C, Hu L, Wu S, Peng X. Understanding performance problems in deep learning systems. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. New York: Association for Computing Machinery, 2022. 357−369.

[72] Tang Y, Khatchadourian R, Bagherzadeh M, Singh R, Stewart A, Raja A. An empirical study of refactorings and technical debt in machine learning systems. In: Proc. of the 43rd ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 238−250. [doi: 10.1109/ICSE43902.2021.00033]

[73] Côté PO, Nikanjam A, Bouchoucha R, Khomh F. Quality issues in machine learning software systems. arXiv:2208.08982, 2022.

[74] Tambon F, Nikanjam A, An L, Khomh F, Antoniol G. Silent bugs in deep learning frameworks: An empirical study of Keras and TensorFlow. arXiv:2112.13314, 2021.

[75] Jin W, Cai Y, Kazman R, Zhang G, Zheng Q, Liu T. Exploring the architectural impact of possible dependencies in Python software. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2020. 758−770. [doi: 10.1109/IC12.2021.00033]

[76] He XX, Zhang YQ, Liu QX. Survey of software supply chain security. Journal of Cyber Security, 2020, 5(1): 57−73 (in Chinese with English abstract).

附中文参考文献:

[41] 程文聪. 面向大规模网络安全态势分析的时序数据挖掘关键技术研究 [博士学位论文]. 长沙: 国防科技大学, 2010.

[76] 何熙巽, 张玉清, 刘奇旭. 软件供应链安全综述. 信息安全学报, 2020, 5(1): 57−73.

张鑫雨(1999－), 女, 硕士生, 主要研究领域为软件历史分析.

晋武侠(1989－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为软件分析, 微服务, 软件架构与质量.

刘靖雯(1996－), 女, 硕士生, CCF 学生会员, 主要研究领域为软件架构坏味道, 软件重构.

范铭(1991－), 男, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为移动软件安全, 隐私合规, AI 安全.

刘烃(1981－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件工程, 智能电网, AI 安全.