2018年第44届软件工程和高级应用欧微会议

CrossSim：利用相互关系来检测类似的开放源码软件项目

Phuong T. Nguyen, Juri Di Rocco, Riccardo Rubei, Davide Di Ruscio
信息工程、计算机科学和数学系 Universita` degli Studi dell'Aquila
Via Vetoio 2, 67100 - L'Aquila, 意大利
{phuong.nguyen, juri.dirocco, riccardo.rubei, davide.diruscio}@univaq.it

***摘要*--*软件开发*是一项知识密集型的活动，需要在不断增加的外部库和资源的压力下掌握多种语言、框架、技术趋势（以及其他方面）。推荐系统在软件工程中越来越重要，因为它们旨在为开发人员提供实时推荐，这可以减少从软件库中发现和理解可重用工件的时间，从而提高生产力和质量。**

在本文中，我们专注于挖掘开源软件库的问题，以确定类似的项目，这些项目可以被评估并最终被开发者重用。为此，我们提出了CROSSSIM作为一种新颖的方法来对开源软件项目和相关工件进行建模并计算它们之间的相似性。对一个包含以下内容的数据集进行了评估
580个GitHub项目显示，CROSSSIM优于现有的技术，该技术已被证明在检测类似的GitHub存储库方面有良好的性能。

***Index Terms*-*mining* software repositories, software similarities, SimRank**

I. 简介

软件开发是一项具有挑战性和知识密集型的活动。它需要在不断增加的外部资源的压力下，掌握多种编程语言、框架、设计模式、技术趋势（以及其他方面）[19]。因此，软件开发人员不断地花时间和精力去了解现有的代码、新的第三方库，或如何正确实现一个新的功能。花在发现有用信息上的时间会对生产力产生巨大的影响[6]。

在过去的几年里，人们在数据挖掘和知识推理技术方面花费了大量的精力，以开发能够为开发者提供自动帮助的方法和工具，帮助他们浏览大型信息空间，并给出可能有助于解决手头特定开发问题的建议。主要的直觉是把推荐系统的概念带到软件开发领域，这些系统通常用于流行的电子商务系统，向用户介绍他们以前不知道的有趣的项目[18]。通过将重点放在以大型资源库的可用性为特征的环境上可重复使用的开源软件（OSS），如GitHub[1]、Bitbucket[2]、SourceForge[3]（仅举几例），因此，构思能够帮助软件工程师识别可重复使用和类似的开源项目的技术和工具是至关重要的，这些项目可以重复使用，而不是实施具有类似功能的内部专有解决方案。

如果两个应用程序实现了相同的抽象所描述的一些功能，那么它们被认为是相似的，即使它们可能包含不同领域的各种功能[14]。了解开源软件项目之间的相似性，可以重用源代码和原型设计，或者选择替代的实现方式[21]，[25]。同时，测量开发者和软件项目之间的相似性是大多数类型的推荐系统的一个关键阶段[17], [20]。未能计算出精确的相似性意味着同时增加了这些系统的整体性能的下降。在以前的工作中，测量软件系统之间的相似性已经被确定为一项艰巨的任务[3], [14]。此外，考虑到开源软件库中工件的混杂性，相似性的计算变得更加复杂，因为许多工件和几个交叉关系普遍存在。目前可用的计算开放源码软件项目相似性的技术可以根据其工作的抽象层分为两个不同的组，即低级和高级。前者考虑源代码、函数调用、API引用等，而后者考虑项目元数据、
例如，文本描述，计算软件系统相似性的readme文件。

在本文中，我们提出了CROSSSIM，一种能够以同质化的方式表示属于不同抽象层的不同项目特征的方法。特别是，我们设计了一个基于图形的模型，可以表示不同的开源软件项目并计算它们的相似性。因此，本文的主要贡献有以下几点：(*i*) 提出了一种新的方法来表示开源软件生态系统，利用其相互关系；(*ii*) 开发了一个可扩展和灵活的框架来计算相似性

本文描述的研究是作为CROSSMINER项目的一部分进行的，欧盟地平线2020研究和创新计划，拨款协议号732223。

[1]GitHub: https://github.com/ [2]Bitbucket: https://bitbucket.org/ 3SourceForge: https://sourceforge.net/

978-1-5386-7383-6/18/$31.00 ©2018 IEE
doi 10.1109/seaa.2018.00069

在开放源码软件项目中，(*iii*)评估所提出的框架与已建立的基线的性能。

本文的其余部分组织如下：第二节概述了最值得注意的检测类似软件应用程序和开源项目的方法。第三节介绍了我们提出的计算开放源码项目之间相似性的方法。第四节描述了对一个真实的GitHub数据集的初步评估。第五节介绍了实验结果。最后，第六节总结了本文，并提出了一些展望工作。

II. 背景介绍

在本节中，我们通过参考过去几年开发的现有技术和工具来介绍检测相似软件项目的问题。根据[3]，根据输入特征集的不同，有两种主要的软件相似性计算类型，即下面讨论的低级和高级。

低水平的相似性。它是通过考虑低级别的数据来计算的，例如，源代码、字节码、函数调用、API参考等。作者在[7]中提出了MUDABlue，一种使用源代码计算软件项目之间相似性的方法。为了计算软件系统之间的相似性，MUDABlue首先从源代码中提取标识符并删除不相关的内容。然后，它创建一个标识符-软件矩阵，每一行对应一个标识符，每一列对应一个软件系统。之后，它删除了太罕见或太流行的标识符。最后，对标识符-软件矩阵进行潜在语义分析（LSA）[10]，使用余弦相似度计算重新得出的矩阵的相似性。CLAN（Closely reLated ApplicatioNs）[14]是一种通过利用对应于包类层次的语义层来自动检测类似Java应用程序的方法。CLAN将源代码文件表示为一个术语文档矩阵（TDM），其中一行包含一个独特的类或包，一列对应一个应用程序。然后应用单值分解来降低矩阵的维度。应用程序之间的相似性被计算为减少的矩阵中的向量之间的余弦相似性。

MUDABlue和CLAN的相似之处在于它们在术语-文档矩阵中表示软件和标识符/API，然后应用LSA来计算相似性。CLAN包括用于计算相似性的API调用，而MUDABlue则将源代码文件中的每个词都整合到术语-文档矩阵中。因此，CLAN的相似性分数比MUDABlue的分数更能反映人类对相似性的感知[14]。

高水平的相似性。它是通过考虑项目元数据来计算的，如主题分布、README文件、文本描述、明星事件（如果有的话，如GitHub中）等。在
[23]作者提出了LibRec，一种图书馆推荐技术，帮助开发者利用现有的图书馆。LibRec采用了关联规则挖掘和协同过滤的方法

该技术用于搜索最相似的项目，并将这些项目使用的库推荐给给定项目。一个项目由一个特征向量来描述，其中每个条目对应于一个库的出现，两个项目之间的相似性被计算为其特征向量的相似性。

在[13]中，标签被用来描述应用程序的特征，然后计算它们之间的相似度。所提出的方法可用于检测用不同语言编写的类似应用程序。与文本描述相比，标签能更好地捕捉应用程序的内在特征，基于这一假设，该方法提取附着在应用程序上的标签并计算其权重[13]。这些信息构成了一个给定的软件系统的特征，并被用来将其与其他系统区分开来。一个应用程序由一个特征向量来描述，每个条目对应一个标签的权重。最终，两个应用程序之间的相似性是用余弦相似性来计算的。

在[25]中，RepoPal被提出来检测类似的GitHub仓库。在这个方法中，如果两个仓库被认为是相似的：(*i*) 它们包含类似的`README.md`文件；
(*ii*) 它们被兴趣相近的用户加星；(*iii*) 它们在短时间内被同一用户加星。因此，GitHub 仓库之间的相似性是通过：`README.md` 文件和每个仓库的星级，以及同一用户的两个后续星级事件之间的时间间隔来计算的。RepoPal与CLAN进行了评估，实验结果[25]显示，RepoPal在两个质量指标方面比CLAN有更好的表现。

总之，通过回顾其他额外的相似性度量[4]、[11]、[12]、[24]（由于篇幅限制，这里无法介绍），我们看到它们通常处理的是低级或高级相似性。我们相信，在计算相似性时结合各种输入信息对开放源码软件库的背景是非常有益的。我们的目标是设计一个代表模型，整合各种工件之间的语义关系，该模型有望提高相似性计算的整体性能。

III. 一种计算OSS项目之间相似性的新方法

在关联数据[1]中，RDF[4] 图是由大量的节点和具有语义关系的导向链接组成的。由于这个特点，这个表示方法为各种计算铺平了道路[5]。通过考虑RDF图的典型应用和检测开源项目的相似性问题的类比，在本节中我们提出了CROSSSIM（计算开源软件相似性的跨项目关系），这是一种利用图来表示开源软件生态系统中不同类型关系的方法。具体来说，我们选择了图模型，因为它允许灵活的数据整合，并有利于众多的相似性指标。

[4]https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

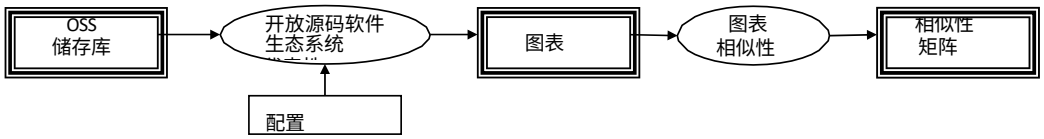


图1.CROSSSIM方法的概述

[2].我们将开发者社区与开放源码软件项目、图书馆以及它们之间的相互作用视为一个*生态系统*。在这个系统中，无论是人类还是非人类因素，都与其他因素相互依赖和影响。在这里，普遍存在着一些联系和相互作用，例如开发者向资源库提交信息，用户向资源库提供信息，或者项目包含源代码文件，仅此而已。

图1描述了CROSSSIM的架构：矩形代表工件，而椭圆代表由开发的CROSSSIM工具自动执行的活动。特别是，该方法从现有的开放源码软件库导入项目数据，并通过开放源码软件生态系统表示模块将它们表示为基于图形的表示。根据所考虑的资源库（以及每个项目的可用信息），要生成的图结构必须被正确配置。例如，在GitHub的情况下，必须指定具体的配置，以使分配给每个项目的星星在目标图中得到表示。这样的配置是 "伪造 "的，只需指定一次，例如，SourceForge不提供GitHub中的星型系统。图形相似性模块实现了相似性算法，该算法应用于基于源图的输入生态系统的表示，生成代表每对输入项目相似性值的矩阵。

在III-A节中给出了对所提出的基于图的开放源码项目的详细描述。关于已实现的相似性算法的细节在第三部分B给出。

A. 开放源码软件生态系统的代表

通过采用基于图的表示法，我们能够将开放源码软件生态系统中各种人工制品之间的关系转化为一种可计算的数学格式。该表示模型通过考虑它们之间直接和间接的相互关系，以及它们作为一个整体的共同出现，以一种统一的方式考虑不同的人工制品。以下关系被用来建立代表开放源码软件生态系统的图形，并最终通过下一节介绍的算法来计算相似度。

- $isUsedBy \subseteq Dependency \times Project$：这种关系决定了一个项目对一个依赖物（例如一个第三方库）的依赖性。该项目需要包括该依赖关系才能运行。根据[14]、[23]的说法
  两个被考虑的项目之间的相似性依赖于它们共同的依赖关系，因为它们的目标是实现类似的功能；
- $开发 \subseteq 开发人员 \times 项目$：我们假设两个项目之间存在一定程度的相似性，如果它们是由相同的开发人员建立的[3]；
- $stars \subseteq User \times Project$：这种关系受到RepoPal[25]中的star事件的启发，表示某个用户在GitHub上标记的项目。然而，我们
  在更广泛的范围内考虑明星事件，即不仅考虑到两个开发商之间的直接联系，而且也考虑到间接联系；
- $开发 \subseteq 用户 \times 项目$: 这种关系用来表示一个特定的用户在其中所贡献的项目。
  源代码开发的条款；
- $implements \subseteq File \times File$：它表示在两个不同文件的源代码之间可能发生的特定关系，例如一个文件中指定的类实现另一个文件中的接口；
- $hasSourceCode \subseteq Project \times File$：它代表一个给定项目中包含的源文件。

图2显示了一个由两个项目`project#1`和`project#2`组成的说明性例子的图。前者包含`HttpSocket.java`，后者包含`FtpSocket.java`和相应的语义谓词`hasSourceCode`。这两个源代码文件都实现了由`implments`标记的接口`#1`。在实践中，一个开放源码软件图要大得多，有许多节点和边，两个项目之间的关系可以被认为是一个子图。

基于图的结构，人们可以利用节点、链接和相互关系，用现有的图相似性算法计算相似性。据我们所知，在图中存在几种计算相似性的指标[2], [16]。在图2中，我们可以使用相关的语义来计算项目1和项目2之间的相似度。

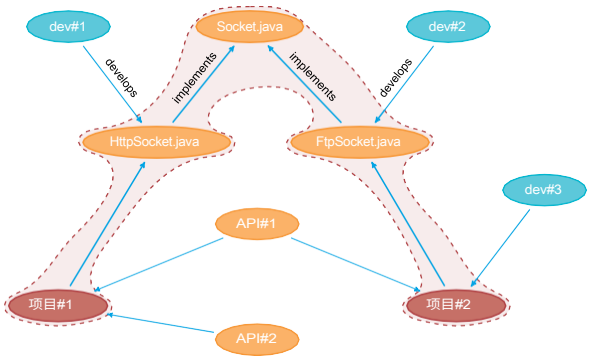


图2.开放源码软件项目之间在实施方面的相似性

路径，例如一跳路径isUsedBy，或者两跳路径hasSourceCode和实现，如图中已经强调的。这个假设是基于这样一个事实：这些项目旨在通过使用共同的库来创造共同的功能[14], [23]。使用该图，也可以计算出开发者dev#1和dev#2之间的相似性，因为他们是通过develop和implements关系间接连接的。

目前可用的CROSSSIM[15]的实现能够管理*isUsedBy*、*develop*和*star*的关系，如第四节所述。

B. 相似性计算

为了评估图中两个节点的相似性，它们的内在特征，如邻居节点、链接以及它们之间的相互作用都被纳入到相似性计算中[5]，[16]。在[8]中，SimRank已经被开发出来，用于计算基于图中节点之间的相互关系的相似性。考虑到两个节点，指向它们的相似节点越多，这两个节点就越相似。在这个意义上，两个节点α，β之间的相似性是通过使用固定点函数来计算的。鉴于$k \geq 0$，我们有$R^{(k)}(\alpha, \beta) = 1$
$\alpha = \beta$，$R^{(k)}(\alpha, \beta) = 0$，$k = 0$，$\alpha \neq \beta$、SimRank的计算方法如下：

$$R^{(k+1)}(\alpha, \beta) = \frac{\Delta}{|I(\alpha)| \cdot |I(\beta)|} \sum_{i=1}^{|I(\alpha)|} \sum_{j=1}^{|I(\beta)|} R^{(k)}(I_i(\alpha), I_j(\beta)) \tag{1}$$

其中Δ是一个阻尼系数（$0 \leq \Delta < 1$）；$I(\alpha)$和$I(\beta)$分别是α和β的传入邻居的集合。$|I(\alpha)| \cdot |I(\beta)|$是用于归一化的系数，因此迫使$R^{(k)}(\alpha, \beta) \in [0, 1]$。

对于CROSSSIM的第一次实施，我们采用了Sim-----。Rank作为计算OSS图节点之间相似性的机制。对于未来的工作，其他的相似性算法也可以灵活地集成到CROSSSIM中，只要它们是为图设计的。

为了研究CROSSSIM的性能，我们使用从GitHub收集的数据集进行了全面的评估。为了进行无偏见的比较，我们选择了其他同类型研究中的现有评估方法[13], [14], [25]。连同其他通常用于评估的指标，即*成功率*、*置信度*和*精确度*，我们决定也使用*排名*来衡量相似性工具对排名结果的敏感性。我们的评估细节将在下一节给出。

IV. 评价

在这一节中，我们讨论了CROSSSIM与RepoPal的对比过程，并应用于评估CROSSSIM的性能。选择RepoPal的理由是，根据*Zhang等人的*研究[25]，RepoPal在*置信度*和*精确度*方面优于CLAN，而CLAN被认为是一个成熟的基线[14]以进行性能比较。我们希望通过所进行的评估来回答以下研究问题：

- **RQ1：** 哪种相似度量产生更好的表现：RepoPal或CROSSSIM?
- **问题2：** 图形结构如何影响CROSSSIM的性能?

为此，已经应用的评估过程如图3所示，由活动和工件组成，详见下文。

我们收集了一个由GitHub Java项目组成的数据集，这些项目可作为相似度计算的输入，并满足以下要求：(*i*) 是GitHub的Java项目；(*ii*) 通过pom.xml或.gradle文件提供其依赖关系的说明[5]；(*iii*) 至少有9个依赖关系；(*iv*) 有README.md文件；

(*v*) 拥有至少20颗星[25]。我们意识到，相似性算法的最终结果要由人类来验证，万一这些项目从本质上来说是不相关的，那么人类评价者给出的认知最终也会是*不一样的*。这对评价相似性是没有价值的。因此，为了方便分析，我们首先观察一些特定类别的项目（例如PDF处理器、JSON解析器、对象关系映射项目和Spring MVC相关工具），而不是以随机方式抓取项目。一旦有一定数量的项目为

在获得每个类别的项目后，我们也开始随机收集，以获得各个类别的项目。

使用GitHub API[6]，我们抓取了一些项目以提供

。直观地说，我们认为RepoPal是一个好的起点

评估的输入。尽管满足单一方法（即RepoPal或CROSSSIM）要求的项目数量很高，但满足两种方法要求的项目数量却低得多。例如，一个项目同时包含pom.xml和README.md，尽管只有5个依赖项，但它并不符合的限制，必须被丢弃。爬行的时间是
对于每个项目来说，至少要发送6次查询才能获得相关数据。GitHub 已经为一个普通账户设定了速率限制：[7]，每小时允许的 API 调用总数为 5.000。而对于搜索操作，速率限制为每分钟30次。由于这些原因，我们最终得到了580个符合评估条件的项目的数据集。我们收集的数据集和CROSSSIM
实验结果已经在网上公布，供公众使用[15]。***RepoPal和CROSSSIM的应用*** RepoPal和CROSSSIM都被应用于收集的数据集。为了解释基于图形的表示方法，图4勾画了两个项目AskNowQA/AutoSPARQL和AKSW/SPARQL2NL之间关系的子图。橙色节点是依赖关系，它们的真实名称在表I中描述。绿松石色的结点

[5] pom.xml和扩展名为.gradle的文件分别与通过Maven（https://maven.apache.org/）和Gradle（https://gradle.org/）管理依赖关系有关。

6GitHub API: https://developer.github.com/v3/

7GitHub速率限制: https://developer.github.com/v3/rate limit/

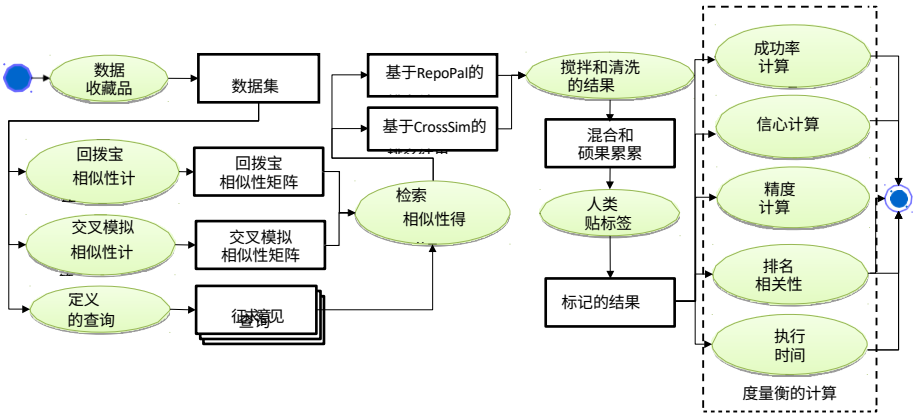


图3.评价过程

是已经在存储库中担任主角的开发者。每个节点在整个图中都使用唯一的编号进行编码。

为了解决RQ2，我们通过考虑各种类型的图，研究了图结构对CROSSSIM性能的影响。在第一种配置中，只有*星形事件*和*依赖关系*被用来构建图此后这被命名为CROSSSIM1。在第二种配置中，我们对CROSSSIM1进行了扩展，也代表了提交者，这种配置被命名为CROSSSIM2。接下来，我们研究了最常见的依赖关系（如表二所示）对计算的影响。为此，从配置CROSSSIM1的图中，删除了所有由这些依赖关系产生的节点和边，这个配置被称为CROSSSIM3。最后，最频繁的依赖关系也从CROSSSIM2中删除，结果为CROSSSIM4。

查询定义 在数据集中的580个项目中，50个有被选中作为查询。由于篇幅所限，以下列表中的本文省略了这50个查询，感兴趣的读者可以参考我们在网上发布的数据集[15]，以了解更多细节。为了达到多样性的目的，我们选择的查询同样涵盖了数据集中的所有项目类别。***检索相似性分数*** 我们的评估与其他一些现有研究[13], [14], [25]一致。特别是，对于50个项目集合中的每个查询

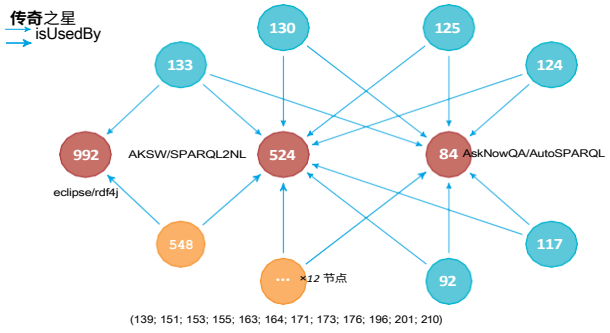


图4.显示AskNowQA/AutoSPARQL、AKSW/SPARQL2NL和eclipse/rdf4j项目代表的一个片段

在上一步骤中定义，使用III-B节中讨论的SimRank算法对数据集中的所有剩余项目进行相似性计算。从检索到的项目中，只选择前5个项目进行后续的评估步骤。对于每个查询，也使用RepoPal计算相似度，以获得前5个最相似的检索项目。

结果的混合和洗牌以及人类标签 为了 有一个公平的评价，对于每个查询，我们将所有相似度量计算产生的前5个结果混合和洗牌在一个文件中，并将它们提交给人类评价者。这有助于消除对某一特定相似度量的任何偏见或成见。特别是，给定一个查询，进行用户研究以评估查询和相应的检索项目之间的相似度。三个研究生参加了用户研究，其中两个是熟练的Java程序员。参与者被要求对每一对项目的相似性进行标注（即，*<query*、

表一
所考虑的数据集中的共享依赖关系

身份证	命名
139	org.apache.jena:jena-arq
151	org.dllearner:component-core
153	net.didion.jwnl:jwnl
155	net.sourceforge.owlapi:owlapi-distribution
163	net.sf.jopt-simple:jopt-simple
164	颚骨：核心
171	com.aliasi:lingpipe
173	org.dllearner:component-ext
176	org.apache.opennlp:opennlp-tools
196	org.apache.solr:solr-solrj
201	org.apache.commons:commons-lang3
210	javax.servlet:servlet-api
548	org.slf4j:log4j-over-slf4j

表二
所考虑的数据集中最频繁的依赖关系

依赖性	频率
珺珺:junit	447
org.slf4j:slf4j-api	217
com.google.guava:guava	171
log4j:log4j	156
共识-io:共识-io	151
org.slf4j:slf4j-log4j12	129

的子图

表三 相似性量表

规模	描述	分数
不相似的	检索到的项目的功能是完全不同的 询项目的人。	1
中性	查询和检索的项目在以下方面具有共同的功能 常见的	2
类似的	这两个项目在以下方面共享大量的任务和功能 常见的	3
高度相似	这两个项目有许多共同的任务和功能，并且 可以认为是相同的	4

检索到的项目>）的应用领域和功能，使用表三[14]中列出的标度。***衡量标准的计算***为了评估算法在用户研究方面的成果，按照一些相关工作[13], [14], [25]中的典型做法，我们考虑了以下衡量标准：

- *成功率*：如果在检索到的前5个项目中，至少有一个项目被标记为相似或高度相似，则认为该查询是成功的。*成功率*是指成功的查询与总的查询次数的比率；
- *信心*：给定一对<*查询，检索的项目*>，评估者的信心是她对项目之间的相似性所赋予的分数；
- *精度*：每个查询的精确性是指前5名列表中被人类标记为相似或高度相似的项目的比例。

除了前面的指标外，我们还引入了一个额外的指标来衡量相似性工具产生的排名。对于一个查询，如果检索到的前5个项目都是相关的，则认为相似性工具是好的。如果存在假阳性项目，即那些被标记为不相似和中性的项目，预计这些项目的排名将低于真阳性项目。如果一个不相关的项目比一个相关的项目有更高的排名，我们认为相似性工具会产生一个不恰当的推荐。下面介绍的*排名*指标是一种评估相似性指标是否产生正确排名建议的手段。

- *排名*：对获得的人类评价进行了分析，以检查由相似性工具计算的排名和人类评价给出的分数之间的相关性。为此，Spearman等级相关系数r_s [22]被用来衡量一个相似性指标在给定的查询中对检索到的项目的排名情况。考虑到两个排名的变量$r_1 = (\rho_1, \rho_2, ..., \rho_n)$和$r_2 = (\sigma_1, \sigma_2, ..., \sigma_n)$，$r_s$被定义为

$$R_s = 1 - \frac{6\sum_{i=1}^{n}(\rho_i - \sigma_i)^2}{n(n^2-1)}$$

由于大平局数，我们还使用了*Kendall's tau*[9]系数、用来衡量两个被考虑的数量之间的顺序关联。r_s和τ的范围都是 –1（完全负相关）到+1（完全正相关）；$r_s = 0$或$\tau = 0$，意味着两个变量不相关。

最后，我们还考虑了与在数据集上应用RepoPal和CROSSSIM以获得相应的相似度矩阵有关的*执行时间*。

V. 实验结果

在这一节中，我们分析了上一节中所讨论的数据，以回答研究问题RQ1和RQ2（见V-A节）。V-B部分还讨论了对我们评价的有效性的威胁。

A. 数据分析

RQ1: 哪种相似性指标能产生更好的性能：RepoPal还是CROSSSIM？ 实验结果表明，RepoPal是计算开放源码软件项目之间相似性的一个好选择。这确实证实了RepoPal的作者在[25]中的说法。与RepoPal相比，CROSSSIM的三种配置获得了更高的性能，其中CROSSSIM3超越了所有配置。

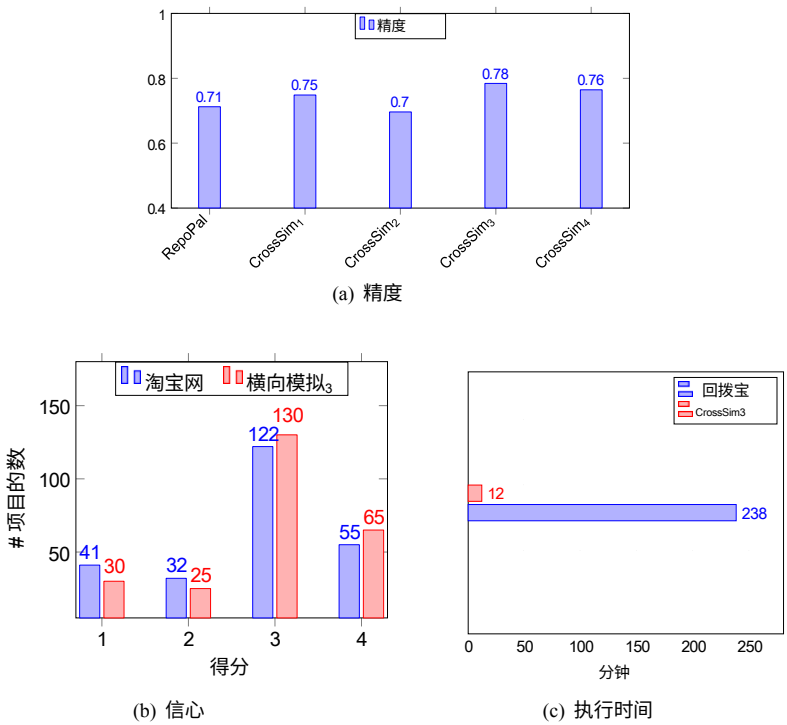
从图5(a)可以看出，CROSSSIM3的性能优于RepoPal在*精度方面的表现*。两者都获得了100%的*成功率*，但是CROSSSIM3的精确度更高。CROSS-SIM3获得了0.78的精度，RepoPal获得了0.71的精度。两个指标的*置信度*如图5(b)所示。此外，通过这个指数，CROSSSIM3产生了更好的结果，因为它有更多的分数是3或4，更少的分数是1或2。

除了传统的质量指数之外，我们还投资了使用Spearman's (r_s)和Kendall's tau (τ)相关指数对这两个指标所产生的排名进行了评估。其目的是看每个指标产生的排名与用户给出的分数之间的相关性有多大，这些分数已经按降序排序了。这样，较低的r_s (τ)意味着较好的排名。r_s和τ是针对所有50个查询和相关的前五个结果计算的。*r*的值$_s$ CROSSSIM3为0.250，*RepoPal*为-0.193。该值CROSSSIM3的τ值为-0.214，*RepoPal*为-0.163。根据这一质量指数，CROSSSIM3的表现略好于*RepoPal*。

与RepoPal的应用有关的执行时间和CROSSSIM3的结果如图5（c）所示。对于实验在使用英特尔酷睿i5-7200U CPU @ 2.50GHz × 4, 8GB RAM, Ubuntu 16.04的笔记本电脑对数据集进行分析时，RepoPal生成相似性矩阵需要≈4小时，而CROSSSIM3的执行，包括生成输入图和生成相似性矩阵的时间，需要≈16分钟。如此重要的时间差异是由于计算`README.md`之间的相似性所需的时间。文件，而*RepoPal正是*依靠这些文件。

CROSSSIM获得的结果证实了我们的假设，即纳入各种特

征，如依赖性，以及与之相关的其他特征。



(a) 精度

(b) 信心

(c) 执行时间

图5.所考虑的衡量标准的结果

在计算相似性的时候，RepoPal考虑的是项目本身的关系，而CROSSSIM考虑的是其他项目之间的交叉关系，这对计算相似性很有帮助。为了计算两个项目之间的相似性，RepoPal考虑的是项目本身的关系，而CROSSSIM通过图形的方式也考虑到了其他项目之间的交叉关系。此外，CROSSSIM更加灵活，因为它可以在不影响内部设计的情况下，在相似性计算中包括其他工件。最后但并非最不重要的是，CROSSSIM的整体性能和它的执行时间之间的比率是非常令人鼓舞的。

RQ2：图结构如何影响CROSSSIM的性能？ 当我们把CROSSSIM1与CROSSSIM2结合起来考虑时，采用提交者的影响可以观察到。CROSSSIM1获得的成功率为100%、精度为0.748。而CROSSSIM2的错误定位数量上升，从而使整体性能大大恶化，其精度为0.696。CROSSSIM2的精度低于*RepoPal*和所有CROSSSIM的同类产品。其性能如果把CROSSSIM3和CROSSSIM4放在一起考虑，就可以进一步看到这种退化。与CROSSSIM3相比，CROSSSIM4的假阳性数量增加了5个项目。我们得出的结论是，将所有的对图中的项目至少提交过一次更新的开发者来说，是适得其反的，因为它增加了精度的下降。在这个意义上，我们做了一个假设，为开发者部署一个加权方案可能有助于抵消性能的下降。我们认为这个问题是我们未来的工作。

我们把CROSSSIM1和CROSSSIM3放在一起考虑，以分析去除最频繁的依赖性的效果。

分数。CROSSSIM3的表现优于CROSSSIM1，因为它获得了0.78的精度，是所有数值中最高的。CROSSSIM1为0.75。移除最频繁的dependency与CROSSSIM2相比，否认也有助于提高CROSSSIM4的性能。总之，这意味着消除原始图中太流行的依赖关系是一个有利可图的修正。一旦我们对III-B节中介绍的SimRank的设计有了更深入的了解，这就可以理解了。在那里，两个项目被认为是相似的如果它们有相同的依赖关系，或者换句话说，它们在图中的对应节点是由一个共同的节点指向的。然而，对于表二中的频繁依赖关系，这一特性可能不再成立。例如，两个项目被junit:junit所指向，因为他们使用JUnit[8] 进行测试。由于测试是许多软件项目的共同功能，它对项目的特征描述没有帮助，因此，需要从图中删除。

综上所述，可以看出，图结构对相似性计算的结果有相当大的影响。在这个意义上，找到一个能滋养相似性计算的图结构是特别重要的。这被认为是一个开放的研究问题。

B. 对有效性的威胁

在本节中，我们将调查可能影响实验有效性的威胁，以及我们如何努力将其降至最低。特别是，我们关注下面讨论的对有效性的内部和外部威胁。

8J单位: http://junit.org/junit5/

内部有效性涉及任何可能影响我们结果的混杂因素。我们试图在评价和评估阶段避免任何偏见：（*i*）通过让三名参与者参与用户研究。特别是，一个用户的标签结果然后由另外两个用户进行双重检查，以确保结果是合理的；（*ii*）通过完全自动评估所定义的指标，没有任何人工干预。事实上，所实施的工具可能是有缺陷的。为了对比和减轻这种威胁，我们进行了几次人工评估和反检查。

外部有效性是指所获得的结果和结论的普遍性。关于我们方法的普遍性，我们只能考虑580个项目的数据集，因为符合RepoPal和CROSSSIM要求的项目数量很少，因此需要长时间的抓取。在数据收集过程中、
我们既抓取了一些特定类别的项目，也抓取了随机项目。随机项目是测试我们算法的通用性的一种手段。如果该算法运行良好，它将不会把新添加的随机项目视为与特定类别的项目相似。在未来的工作中，我们将通过纳入其他相似度指标和更多的GitHub项目来验证我们提出的方法。

VI. 结论

在本文中，我们提出了一种检测类似开源软件项目的方法。我们提出了一种基于图形的开源项目的各种特征和语义关系的表示方法。通过所提出的图形表示法，我们能够将各种人工制品之间的关系，如开发人员、API的使用、源代码、互动，转化为一种数学上可计算的格式。

我们对580个GitHub Java项目的数据集进行了评估，以研究我们方法的性能。获得的结果是有希望的：通过考虑RepoPal作为基线，我们证明了CROSSSIM可以被视为计算开源软件项目之间相似性的良好候选者。在未来的工作中，我们将
研究哪种图结构可以帮助获得更好的相似性结果，以及定义一个阈值，使项目的依赖性被认为是*频繁的*。

参考文献

[1] C.Bizer, T. Heath, and T. Berners-Lee.关联数据--到目前为止的故事。*Int.J. Semantic Web Inf.Syst.*, 5(3):122, 2009.

[2] V.D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. V. Dooren. 图顶点之间的相似性测量：应用于同义词提取和网络搜索。*SIAM Rev.*, 46(4):647-666, Apr. 2004.

[3] N.Chen, S. C. Hoi, S. Li, and X. Xiao.Simapp：一个通过在线内核学习检测类似移动应用的框架。在*第八届ACM网络搜索和数据挖掘国际会议论文集*，WSDM '15，第305-314页，美国纽约，2015。ACM.

[4] J.Crussell, C. Gibler, and H. Chen.Andarwin：对语义相似的安卓应用进行可扩展的检测。J. Crampton, S. Jajodia, and K. Mayes, editors, *Computer Security - ESORICS 2013：第18届欧洲计算机安全研究研讨会，英国埃尼姆，2013年9月9-13日。论文集*，第182-199页，柏林，海德堡，2013。Springer Berlin Heidelberg.

[5] T.Di Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker.链接开放数据以支持基于内容的推荐系统。*第八届国际语义系统会议论文集*，I-SEMANTICS '12，第1-8页，美国纽约，2012。ACM。

[6] E.Duala-Ekoko and M. P. Robillard.提出和回答关于不熟悉的API的问题：一个探索性的研究。在*第34届国际软件工程会议论文集*，ICSE '12，第266-276页，美国新泽西州皮斯卡塔韦，2012。IEEE出版社。

[7] P.K. Garg, S. Kawaguchi, M. Matsushita, and K. Inoue。Mudablue：开放源码库的自动分类系统。*2013年第20届亚太软件工程会议（APSEC）*，第184-193页，2004。

[8] G. Jeh和J. Widom.Simrank：结构-背景相似性的测量。在*第八届ACM SIGKDD知识发现和数据挖掘国际会议论文集*，KDD '02，第538-543页，美国纽约，2002。ACM.

[9] M.G. Kendall.等级相关方法。1948.

[10] T.K. Landauer.*Latent Semantic Analysis*.Wiley Online Library, 2006.

[11] M.Linares-Vasquez, A. Holtzhauer, and D. Poshyvanyk.On automatically detecting similar android apps.*2016年IEEE第24届国际程序理解会议（ICPC）*，00：1-10，2016。

[12] C.Liu, C. Chen, J. Han, and P. S. Yu.Gplag：通过程序依赖图分析检测软件抄袭。在*第12届ACM SIGKDD知识发现和数据挖掘国际会议论文集*，KDD '06，第872-881页，美国纽约，2006。ACM.

[13] D.Lo, L. Jiang, and F. Thung.用协作标签检测类似的应用。*2012年IEEE国际软件维护会议（ICSM）论文集*，ICSM '12，第600-603页，美国华盛顿特区，2012。IEEE计算机协会。

[14] C.McMillan, M. Grechanik, and D. Poshyvanyk.检测类似的软件应用。在*第34届国际软件工程会议论文集*，ICSE '12，第364-374页，美国新泽西州皮斯卡塔韦，2012。IEEE出版社。

[15] P.T. Nguyen, J. Di Rocco, R. Rubei, and D. Di Ruscio.CrossSim工具和评估数据，2018。https://doi.org/10.5281/zenodo.1252866。

[16] P.T. Nguyen, P. Tomeo, T. Di Noia, and E. Di Sciascio.对simrank和个性化pagerank的评估，以建立一个数据网络的推荐系统。In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 1477-1482, New York, NY, USA, 2015.ACM。

[17] T.D. Noia and V. C. Ostuni.推荐系统和链接的开放数据。In *Reasoning Web.网络逻辑规则--2015年第11届国际暑期学校，德国柏林，2015年7月31日至8月4日，教程讲座*，第88-113页，2015年。

[18] F.Ricci, L. Rokach, and B. Shapira.*推荐系统介绍手册》*，第1-35页。Springer US, Boston, MA, 2011.

[19] M.P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, editors.*软件工程中的推荐系统*。Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.doi: 10.1007/978-3-642-45135- 5.

[20] B.Sarwar, G. Karypis, J. Konstan, and J. Riedl.基于项目的协同过滤推荐算法。In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285- 295, New York, NY, USA, 2001.ACM.

[21] J.B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. The adaptive web. Chapter Collaborative Filtering Recommender Systems, pages 291-324.Springer-Verlag, Berlin, Heidelberg, 2007.

[22] C.斯佩尔曼。两个事物之间关联的证明和测量。*美国心理学杂志*，15（1）：72-101，1904。

[23] F.Thung, D. Lo, and J. Lawall.自动化的图书馆推荐。在*2013年第20届逆向工程工作会议（WCRE）*上，第182-191页，2013年10月。

[24] X.Xia, D. Lo, X. Wang, and B. Zhou.软件信息网站中的标签推荐。在*第十届软件资源库挖掘工作会议上*，MSR '13，第287-296页，美国新泽西州皮斯卡塔韦，2013。IEEE出版社。

[25] Y.Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun。检测github上的类似存储库.*2017年IEEE第24届软件分析、进化和再造国际会议（*

SANER），00：13-23，2017。