软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10. 13328/j. cnki. jos. 000000]

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

基于图嵌入的软件项目 API 检索方法*

凌春阳[1,2], 邹艳珍[1,2,3], 林泽琦[1,2], 谢冰[1,2], 赵俊峰[1,2,3]

[1](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[2](北京大学 信息科学技术学院,北京 100871)

[3](北京大学(天津滨海)新一代信息技术研究院,天津 300450)

通讯作者: 谢冰, E-mail: xiebing@pku.edu.cn

摘 要: 检索和复用软件项目 API(Application Program Interface, 应用程序接口)是软件工程领域的一项重要任务. 随着软件项目的规模越来越大、越来越复杂, 当前 API 检索一方面需要提高自然语言查询的准确性, 另一方面需要定位和展示目标 APIs 及其相关代码之间的关联, 以更好地辅助用户理解 APIs 的实现逻辑和使用关系. 为此, 本文提出了一种基于图嵌入的软件项目 API 检索方法. 该方法能够基于软件项目源代码, 自动构建其代码结构图, 并通过图嵌入对源代码进行信息表示. 在此基础上, 用户可以输入自然语言问题、检索并返回相关的 APIs 及其关联信息构成的连通代码子图, 从而提高了 API 检索和复用的效率. 在以开源项目 Apache Lucene 和 POI 为例的检索实验中, 本文方法检索结果的 F1 值相比现有基于最短路径的方法提高了 10%, 平均响应时间缩短了约 60 倍.

关键词: API 检索;代码检索;代码图;图嵌入

中图法分类号: TP311

中文引用格式: 凌春阳, 邹艳珍,林泽琦,谢冰,赵俊峰. 基于图嵌入的软件项目 API 检索方法. 软件学报. http://www. jos. org. cn/1000-9825/ 0000. htm

英文引用格式: Ling CY, Zou YZ, Lin ZQ, Xie B, Zhao JF. Searching Software Project APIs with Graph Embedding. Ruan Jian Xue Bao/Journal of Software, 2018 (in Chinese). http://www. jos. org. cn/1000-9825/0000. htm

Searching Software Project APIs with Graph Embedding

LING Chun-Yang[1,2], ZOU Yan-Zhen[1,2,3], LIN Ze-Qi[1,2], XIE Bing[1,2], ZHAO Jun-Feng[1,2,3]

[1](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

[2](School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[3](Beida (Binhai) Information Research, Tianjin 300450, China)

Abstract: Software API (Application Program Interface) search is an important research issue in software engineering. As software projects are becoming more and more complex, existing search tools mainly face the following two challenges. First, more accurate search results are required in natural language question based search process. Second, the relationships between APIs are required to illustrate so that we are able to understand these APIs' underlying logic and usage scenarios more quickly. In this paper, we propose a novel approach to searching a software project's APIs based on graph embedding. It aims to improve the accuracy of natural language based code graph search. We build a software project's code graph automatically from its source code and represent them through graph embedding. For a natural language question, a code-connected subgraph, composed by relevant APIs and their associated relationships, are returned as the best answer. In

* 基金项目: 国家重点研发计划(2016YFB1000801), 国家杰出青年科学基金(61525201);

This work was supported by the Foundation item: National Key Research and Development Program (2016YFB1000801), National Science Fund for Distinguished Young Scholars (61525201)

收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 00-00-00; jos 在线出版时间: 0000-00-00

CNKI 在线出版时间: 0000-00-00

experiments, we select *Apache Lucene* and *POI* projects as examples to perform some API search tasks. The experimental results show that our approach improves F1-score by 10% than existing shortest path based approach, while reduces average response time about 60 times.
Key words: API search; code search; code graph; graph embedding

软件项目 API 检索是软件工程领域中的一个重要研究问题. 在软件开发过程中, 开发者最常用的复用方式就是通过检索、调用软件项目提供的 APIs (Application Programming Interfaces, 应用程序接口)来实现所需的功能. 对于面向对象的软件项目而言，APIs 主要是软件项目中可供用户使用的类和方法。然而, 随着软件项目变得越来越大、越来越复杂, API 检索也变得越来越困难[1]. 譬如，开源项目 Lucene 包含了超过 5,377 个类，以及 34,042 个方法. 而研究人员在研究搜索引擎查询日志时发现, 开发者往往花费了大量的时间与精力来查找 APIs[2]. 由此也促生了在很多技术性问答网站(如 StackOverflow)上, 可以发现大量关于如何使用 APIs 的提问.

受自然语言处理技术的影响, 目前许多软件源代码检索工具, 譬如 Krugle[4], Ohloh[5]和 Sourcerer[6]等, 都是将源代码视为纯文本进行处理. 其基本思路是利用软件源代码中的自然语言特征, 基于查询问题与代码的文本相似度进行匹配, 返回相似度较高的检索结果. 典型的相似度匹配模型包括布尔模型(Boolean Model)[7], 向量空间模型(Vector Space Model)[7], BM25 模型[7], 等等. 然而研究表明, 这些工具的准确率不能让人满意. 在 F. Lv 等人[8]2015 年进行的一项研究中表明, Ohloh 返回的排名前十的结果中只有 25. 7%-38. 4%是有用的. 研究者们指出, 导致现有软件源代码检索工具效果不理想的一个主要原因是其缺乏对自然语言查询问题语义的理解. 对此, 研究者们从深化问题理解的角度提出了许多新的方法, 如扩展语义相近的词(Semantic Similar Words)[9], 引入查询重构策略(Query Reformulation)[10], 或利用其他在线文档(如 MSDN)来优化问题理解和检索过程[8]等. 虽然这些技术很大程度上提高了 API 定位的准确度, 但其需要积累和利用大量的辅助信息, 在上述信息缺乏的情况下, 往往不能达到预期效果. 本文则希望仅从软件项目的源代码出发, 帮助开发者提高 API 检索的效率, 因为源代码是复用过程中能够直接获得的资源, 其他信息有时则难以获得.

此外, 现有这些检索工具返回的结果往往只有相关的代码片段信息, 缺乏对关联 APIs 的说明与展示, 不便于用户理解目标 APIs 调用背后的逻辑关联和整体概念, 在一定程度上影响了软件复用的效率. 为此, 研究者们指出不应单纯地将软件代码视为纯文本. 譬如在 C.McMillan 和 W.K.Chan 等人[11, 12]的工作中将源代码以有向图的形式组织起来, 图中的结点表示代码元素, 边表示代码元素之间的关联关系. 图结构是对源代码知识的一种自然且有效的表示, 被广泛应用于软件文档检索、特征定位等问题[13]. 借助代码的结构图, 对源代码的检索就转化为图上的搜索过程, 这样得到的结果既能包含需要被定位的代码元素, 又能充分体现代码之间的关联关系. 举例来说, 假设在使用开源软件项目 Apache Lucene[1]的过程中, 开发者当前使用 StringField 类对一个字符串的字段进行处理. 当需求发生变化, 还要对含有浮点数的字段进行处理时, 这个开发者就需要使用 FloatPoint 类并考虑如何重构代码. 而我们可以从下图 1 所示源代码图中发现, 这两个类都继承自 Field 类, 实现了 IndexableField 接口，同时具有 tokenStream 方法. 因此对开发者来说, 一种更好的策略是使用该抽象的父 Field 作为变量类型, 并在处理不同查询时赋值为不同的子类, 从而能够较好的满足其上述需求.

目前在基于图结构的 API 检索方面, 主要采用了最短路径[12] 、PageRank[11] 等浅层语义分析技术. 考虑到在图上进行搜索是一个计算量很大的问题, 且要求延迟越短越好, 算法的效率至关重要. 如果使用在线查询最短路径的方式, 则由于搜索过程中经常需要计算图上两个结点之间的距离, 将会导致较长的延迟时间. W. K. Chan 等人的工作[12]也提到了这个问题, 并利用了一些近似算法进行优化.

针对这些问题, 本文提出一种基于图嵌入的软件项目 API 检索方法. 图嵌入是一种表示学习技术, 能够将图上的结点映射为低维空间中的实值向量, 通过向量之间的关系来表达图上的结构信息. 对于本文的问题而言, 图嵌入技术一方面能够有效表达软件代码图中的深层结构信息, 与最短路径等方法相比, 更充分地考虑了结点的上下文和关联关系;另一方面, 图嵌入过程可以离线进行, 在用户在线查询阶段可以立即参与计算,

1 http://lucene. apache. org/

时间复杂度仅是常数级别, 大大提高了算法效率.

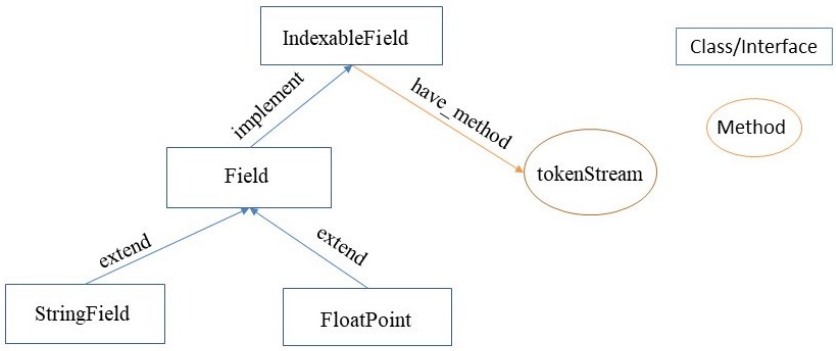


Fig 1 Code graph of Lucene project (part)

图 1 开源项目 Lucene 的代码图示例(部分)

对比现有工作, 本文主要贡献包括:

(1) 提出了一种将软件源代码进行图嵌入表示的方法. 该方法能够能够基于软件项目源代码, 自动构建其代码图结构, 并通过图嵌入对源代码进行信息表示;

(2) 提出了一种基于图嵌入的软件项目 API 检索方法. 该方法能够基于项目源代码的图嵌入技术表示, 将自然语言查询问题语义匹配到代码子图, 提高了检索的准确性同时展示了 API 的关联关系;

(3) 实现了一个基于图嵌入的软件项目 API 检索工具原型, 并通过两个具体软件项目中关于源代码检索的实际问题为例, 验证了本文方法的有效性. 与 W. K. Chan 等人提出的方法相比, 本文方法在召回率、F1 值上有了显著提升, 并有效减低了检索响应时间.

本文章节安排如下:本文第 1 节介绍图嵌入技术;第 2 节详细描述本文提出的基于图嵌入的软件项目源代码检索方法;第 3 节通过实验验证本文方法的有效性;第 4 节介绍相关工作; 第 5 节对本文工作进行总结并展望未来研究工作.

1 图嵌入

图嵌入(Graph Embedding)的主要目的是将一张图上的结点映射到低维空间的向量, 通过这些向量来体现原本图上的结构信息. 这里的结构信息可以是一阶、二阶甚至更高阶的结构, 一阶结构信息即保持原本相邻的结点在嵌入空间中的距离仍然很近, 而更高阶的结构信息则可以保持原本具有相似上下文的结点在嵌入空间中的距离仍然很近. 目前图嵌入技术在知识表示领域颇受关注, 被广泛应用于可视化、顶点分类、关联预测、推荐等任务. 现有图嵌入技术主要可以分为以下几类[14] :

基于因子分解的图嵌入方法. 该类方法将图上结点之间的关系表示成矩阵, 对该矩阵进行因子分解从而得到嵌入的向量. 矩阵的类型包括邻接矩阵, 拉普拉斯矩阵(Laplacian matrix), 卡兹相似矩阵(Katz similarity matrix)等, 根据不同的矩阵类型有不同的分解方法. 其中, M. Belkin 等人[15]提出的 Laplacian Eigenmaps 就是对拉普拉斯矩阵做特征值分解得到的;A. Ahmed 等人[16]提出的 GF 算法则是基于对邻接矩阵做因子分解. 这种方法的时间复杂度是结点数的平方量级, 可扩展性较差, 对于规模庞大的代码图而言并不适用.

基于随机游走的图嵌入方法. 该类方法算法能够近似表示图的许多属性, 比如结点的中心度和相似度. B. Perozzi 等人提出的 DeepWalk [17]就是该方法的典型代表, 该算法利用了神经语言模型 SkipGram 的思想, 根据周围的邻居结点来预测当前结点的嵌入向量. 该类方法通常从某一结点为中心出发进行多次随机游走, 最大化观测到的前 k 个结点以及后 k 个结点的概率, 能够保持高阶的相似性. 当只需要观测图的部分结构或者图太大而无法全部测量时, 该类方法是一个不错的选择. 但是该类方法通常只能捕捉到一条路径上的局部结构

信息, 并且难以找到最佳的采样方法, 调优困难.

基于深度学习的图嵌入方法. 该类方法有的利用深度自编码器(Deep Autoencoder)来实现降维, 因为自编码器拥有学习数据中非线性结构的能力.典型的, D. Wang 等人[18]提出的 SDNE 包括无监督和有监督两部分, 前一部分由自编码器学习能够重新构邻居的结点嵌入, 后一部分则基于 Laplacian Eigenmaps 对那些本来相邻而映射到嵌入空间后却相距很远的结点进行惩罚. 还有一些算法利用卷积神经网络来得到嵌入向量, 如 T. N. Kipf 等人[19]提出的一种半监督学习的方法, 这些算法的计算开销更小, 更适用于相对稀疏的图. 这一类方法虽然能更好地捕捉非线性的结构, 然而复杂度还是较高, 需要大量的计算开销.

本文采用的是 J. Tang 等人[20]提出的 LINE(Large-scale Information Network Embedding)方法, 该方法的可扩展性很好, 能够快速地对大规模网络进行嵌入, 并且能够同时保留局部和全局的结构信息. 该方法严格意义上不属于上述的三种类型, 而是在模型中显式地定义了一阶和二阶的近似函数, 以及任意两个结点之间的联合概率分布, 通过最小化嵌入矩阵与邻接矩阵所代表的两个概率分布之间的 KL(Kullback-Leibler)散度得到图嵌入的向量. 定义好目标函数之后, LINE 并没有采用传统的随机梯度下降算法进行优化, 而是提出了一种新的边采样的方法(Edge-sampling), 根据边的权值成比例的概率进行采样, 这样能够防止随机梯度下降中由于边的权值过大导致的梯度爆炸问题.为了进一步形象地阐明一阶近似和二阶近似的意义, 我们以图 2 为例进行说明. 一阶近似能够将直接有边相连的结点映射到更近的距离, 譬如图 2 中的结点 6 和结点 7, 它们之间有一条权重很大的边连接. 而二阶近似则能够将具有相同邻居的结点映射到更近的距离, 譬如图 2 中的结点 5 和结点 6, 它们都具有很多相同的邻居结点. 因此一阶近似能够保留更多局部的结构信息, 而二阶近似能够保留更多全局的结构信息, 同时考虑两种近似的结果就是将结点 5、6、7 都映射到嵌入空间中更近的距离.

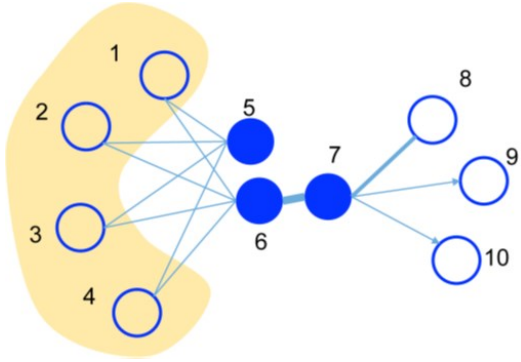


Fig 2 Graph embedding maps nodes with similar structure to closer distance in the embedding space[20]

图 2 图嵌入将结构相似的结点映射到嵌入空间中更近的距离[20]

2 本文工作

本文提出了一种基于图嵌入的软件项目 API 检索方法. 其基本思想是利用图嵌入技术有效表达软件代码图中的深层结构信息, 在检索过程中充分地考虑了结点的上下文和关联关系. 如图 3 所示, 该方法主要包括四个部分:

1）代码图的构建:获取项目的源代码并做相应解析, 自动构建其代码图;

2）代码图的嵌入:使用图嵌入技术将代码图中的结点表示为向量, 以备后续阶段使用;

3）问题与代码图结点的匹配:对用户输入的自然语言问题进行预处理, 然后匹配到代码图中结点作为候选结点, 并对候选结点进行度量;

4）代码子图的生成与推荐:从第二阶段得到的候选结点中挑选合适的结点构成子图, 并扩展为连通的子图, 将排名最佳的结果返回给用户.

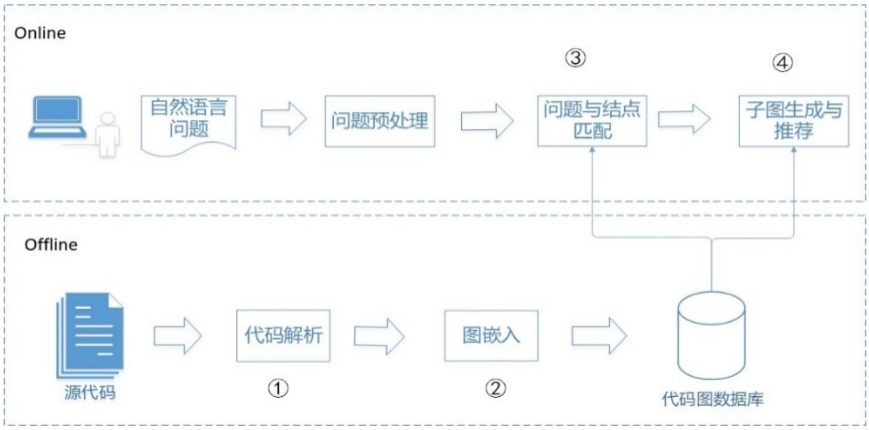


Fig. 3 Framework of searching software source code based on graph embedding

图 3 基于图嵌入的软件项目源代码检索方法框架

2.1 代码图的构建

在一个软件项目（对于面向对象的函数库而言）的源代码中，主要有两种组成成分：一种是类或接口，另一种是方法. 而类与方法之间的关联关系的类型大体上所有面向对象语言都共有，当然部分关联关系与具体编程语言有关. 本文以 Java 语言作为实验对象，考虑了 Java 语言中的如下六种关联关系，分别为继承(Inh)、实现(Imp)、成员(Mem)、参数(Par)、返回值(Ret)和调用(Call)，用集合$Rel = \{Inh, Imp, Mem, Par, Ret, Call\}$表示. 在此基础上，针对一个开源的软件项目，我们可以利用现有工具可以完成对其源码进行解析，构建代码图并存储于图数据库中.

代码图(Code Graph). 代码图$G = (V_G, E_G)$是一个有向无权图，顶点集 V_G是所有的类和方法的结点，边集E_G中任一条边$e(u, v), \exists R \in Rel$, 使得$uRv$. 其中$uRv$表示 u, v 之间存在 R 类型的关联关系.

Table 1 Relationship types in code graph

表 1 代码图中的关系类型

关系名称	关联的结点类型	描述
继承(Inh)	类/接口 A-类/接口 B	表示类/接口 A 是类/接口 B 的父类/接口
实现(Imp)	类 A -接口 B	表示类 A 实现了接口 B
成员(Mem)	方法 A-类/接口 B	表示方法 A 是类/接口 B 的成员
参数(Par)	方法 A-类/接口 B	表示类/接口 B 是方法 A 的参数
返回值(Ret)	方法 A-类/接口 B	表示类/接口 B 是方法 A 的返回值
调用(Call)	方法 A-方法 B	表示方法 A 调用了方法 B

现在主流的 Java 源码解析工具有很多，例如 CheckStyle[21], JAVAssist[22], Yasca[23], JDT[24]等. 这些工具都是现成可用，可以解析 Java 项目并提取出代码元素，以及它们之间的关系. 本文使用的是 Eclipse 的 JDT，作为一个轻量级的代码解析工具，它可以迅速将 Java 代码构建成一个 DOM 结构的抽象语法树(Abstract Syntax Tree, AST). 代码中的每个元素都对应 AST 上的一个结点，通过遍历 AST 就可以得到所有代码元素和它们之间的关系，用来构建软件代码结构图.

本文使用图数据库 Neo4j[1]来存储代码图. Neo4j 是一个用 Java 实现，完全兼容 ACID 的图数据库，其本身数据组织的结构和我们需求的代码结构图非常相似，也是通过结点与结点之间的关系构成整张图. 除此之外，

1 https://neo4j.com/

Neo4j 还提供了诸如索引(Index), 遍历(Traversal)等机制以及一种类 SQL 的查询语言 Cypher, 以方便在图上进行更高级的操作.

2.2 代码图的嵌入

本文使用图嵌入的原因是在第三阶段, 即代码子图的生成与推荐时, 需要计算图上任意两个顶点之间的距离. 图嵌入技术可以使得这一步骤变得十分方便与快速. 如果采用在线查询两个顶点的最短路径的方式则将非常耗时, 而图嵌入过程却是离线的, 在构建完代码图后即可完成.

本文使用了 J. Tang 等人[20]提出的名为 LINE 的方法. 该方法的可扩展性很好, 时间复杂度较低, 能够快速地对大规模网络进行嵌入, 并且同时保留局部和全局的结构信息, 能够适用于有向图、无向图、带权图等多种类型. LINE 的源代码可以在 Github 上找到, 本文用 Java 语言重新实现了该算法.

结点之间的距离. 代码图嵌入以后, 代码图中的任一顶点 $v \in V_G$, 都映射到一个向量 $r_v \in R^d$, 其中 d 表示该向量的维度;任意两个顶点 u, v 之间的距离定义为相应的向量之间的欧氏距离, 即 $dist(u,v) = \| r_u - r_v \|_2$.

2.3 问题与结点的匹配

从这一阶段开始, 便进入在线查询的过程. 当用户输入一个自然语言的问题后, 我们先对问题进行预处理, 然后匹配到代码图中一些候选结点.

2.3.1 问题与结点的预处理

本文采用简单的词袋模型(bag of words), 将问题视为一些无序的词组成的集合, 忽略其句式和位置等信息. 由于自然语言文本还存在单词之外的一些符号, 如标点符号等, 本文将去除非数字和非字母的其他符号, 并以此作为分隔符进行切词.切词以后,进行停用词处理. 自然语言包含许多无实际意义的功能词, 譬如冠词和介词等. 这些词存在十分普遍而包含信息的却极少, 人们一般称之为停用词(stopwords). 本文去除的停用词包括常见的英文停用词, Java 保留字, 以及项目领域特有的功能词(譬如 lucene), 等等.

对于代码图中结点(类或方法)的预处理, 是将该类或方法的名字通过切词得到一个词袋, 用来描述该结点的语义信息. 本文使用驼峰切词(camel split)法进行处理, 比如 QueryParser 这个类名, 切词后的结果就是{query, parser}.

问题 (Query). 问题 Q 是由一些去除停用词之后的单词的集合组成, 将该单词集合记为 $BOW_Q = \{q_1, \cdots, q_n\}$.

结点单词集合. 代码图 G 中的任一结点 v, 都有一个与之关联的单词集合, 记为 $BOW_v = \{s_1, \cdots, s_m\}$.

2.3.2 产生候选结点

对于问题词袋 BOW_Q 中的每个词, 我们都为之在图上找到一系列的候选结点, 这样做是为了充分保留原问题中的语义信息.

候选结点集合. 问题 Q 所匹配到的**候选结点集合**是一个集族, 记为 $Candidate_Q = \{C_1, \cdots, C_n\}$, 其中 $C_i (1 \le i \le n)$ 是单词 q_i 匹配到的代码图中结点的集合, 其大小记为 $|C_i|$.

接下来需要明确对于每个单词 q, 如何匹配到它所对应的候选结点. 本文使用文本匹配的方法, 主要考虑了如下五种匹配情形:

（1）**全名匹配**. 单词 q 本身就是就是一个类或方法的全名, 譬如 q = QueryParser.

（2）**部分匹配.** 单词 q 被结点 v 所对应的词袋所包含, 即 $q \in BOW_v$, 譬如 q = query.

（3）**词根化匹配**. 单词 q 与结点 v 所对应的 BOW_v 中的词经词根化后相同. 词根化(stemming)能够将语义上相同而由于语法等原因导致词形有所不同的词抽取出同样地词根, 本文采用 Snowball[1]作为词根化工具. 譬如 q = parse, 由于 parse 和 parser 词根化后都是 pars, 因此 q 也会匹配到 QueryParser 这个结点.

---

1 http://snowball. tartarus. org/

（4）**缩略规则匹配.** 单词 q 经过手工构造的一些规则转换后，被结点 v 所对应的BOW_v包含. 这是为了解决自然语言的词汇与代码元素中的命名不一致的问题，代码元素的命名为了方便，很可能只采取了自然语言单词的缩写，譬如 *Document* 在代码中的命名可能是 *Doc*, *Number* 在代码中的命名可能是 *Num*, 等等.

（5）**同义词匹配.** 单词 q 与结点 v 所对应的BOW_v中的词，经过相似度计算被判定为同义词. 同近义词是自然语言处理领域的一个重要研究问题，在自然语言与代码元素匹配的过程中也存在这个问题，譬如问题中的词是 *delete*，但代码中某个方法的名字可能是 *remove*，这种情况也应该被考虑. 为了解决这个问题，本文使用了在 Wikipedia 上预训练的 glove[1]词向量. 通过计算两个词向量的余弦相似度来表示语义的相近性，当大于特定阈值时就认为这两个词是同义词.

$$Similarity(u,v) = \frac{\boldsymbol{w}_u \cdot \boldsymbol{w}_v}{\| \boldsymbol{w_u} \| \| \boldsymbol{w}_v \|} \tag{1}$$

2.3.3 候选结点的度量

上一步骤为了保证对原问题中的词汇的覆盖程度，产生了大量的候选结点. 但这些结点不应该都是同等地位的，有的候选结点与问题语义更贴近，有的却徒增了许多噪音，因此需要对结点与问题的文本相似程度进行度量.

本文考虑两种评价指标，一是该结点的BOW_v与问题的BOW_Q中相关的词越多越好;二是该结点引入的不相关的词越少越好，即当结点的BOW_v与问题的BOW_Q相关的词数量相同时，BOW_v的大小越小的结点质量越好. 可类比于信息检索中的准确率与召回率，具体定义如下:

$$score_{relevant}(v) = \frac{|BOW_v \cap BOW_Q|}{|BOW_Q|} \tag{2}$$

$$score_{irrelevant}(v) = \frac{|BOW_v \cap BOW_Q|}{|BOW_v|} \tag{3}$$

为了考虑同义词的影响，在计算两个词袋的交集$BOW_v \cap BOW_Q$时，需要进行特殊处理. 如果BOW_v中的某个单词不在BOW_Q中出现过，那么利用词向量计算该单词与BOW_Q中所有单词的余弦相似度，取其中的最大值作为该单词对交集的贡献. 因此$|BOW_v \cap BOW_Q|$的值不一定是整数.

结点权重. 以$score_{relevant}$和$score_{relevant}$的 *F1* 值作为该结点与问题的相关度的度量，称为该结点的**权重**，记为$w(v)$,其计算公式如下所示:

$$w(v) = \frac{2 * score_{relevant}(v) * score_{irrelevant}(v)}{score_{relevant}(v) + score_{irrelevant}(v)} \tag{4}$$

2.4 子图的生成与推荐

得到候选结点集合之后，需要从中挑选合适的结点来构成能够反映问题语义的子图. 从大量的候选结点中构成子图的搜索空间非常巨大，如何高效地生成子图并度量其优质程度是问题的重中之重. 最后，将生成的子图扩展成为连通的子图返回给用户. 因此本阶段可分为两个部分:子图的生成与度量，子图的扩展与推荐.

2.4.1 子图的生成与度量

本文从两个方面来评价子图的质量:一是子图中的结点与问题之间的累积文本相似程度尽可能高，每个结点的文本相似程度即结点的权重$w(v)$;二是子图中结点之间的距离越近越好，这将起到消除歧义的作用. 譬如子图中的一个结点是 *Document* 这个类，现在有两个同名的方法 *add*，它们的文本相似程度是相同的，那么距离 *Document* 类结点更近的那个方法才可能是相关的，而距离远的结点可能是一个属于其他类的方法，与本问题无关.

首先，我们定义如何从候选结点的集合中选择子图的顶点集.

顶点集. $Candidate_Q = \{C_1, \cdots, C_n\}$构成的子图 G'的顶点集 $V' = \{C_{i,j} \mid x_{i,j} = 1\}$，其中 $x_{i,j}$是二元变量，

1 https://nlp. stanford. edu/projects/glove/

表示C_i中的第 j 个结点是否被选中，满足约束条件 $\sum_{j=1}^{|C_i|} x_{i,j} \geq 1, x_{i,j} = x_{i',j'} \ iff \ C_{i,j} = C_{i',j'}$.

这里的第一个约束条件是为了使问题中的每个词所对应的候选结点集合中至少有一个结点被选中，以此保证对原问题语义的覆盖度. 另外，同一个结点可能在不同的集合中重复出现，因此只要该结点在某个集合中被选中，则其他集合中也必须选中它，这就由第二个约束条件保证.

接下来，我们定义一个子图G'的度量函数.

$$score(G') = \sum_{u,v \in V', u \neq v} \frac{dist(u,v)}{w(u) * w(v)} \tag{5}$$

其中的距离$dist(u,v)$即 2.2 节中所述的使用图嵌入向量计算的欧氏距离. 子图的生成过程即求解这个优化问题 $min\, score(G')$，这可以看成一个二次优化问题. 为了方便求解，本文提出一种基于柱状搜索(*beam search*)的方法. 柱状搜索可以看作对贪心法的一种优化，每次将保留前 k 个最佳的候选结果. 具体如下表的算法 1 所示，对于输入的候选结点集合，按权值排序后取最高的 k 个结点作为起始结点，其中参数 k 就代表着柱状的大小(*beam size*). 第 7-10 行是先根据公式(5)中的度量函数计算加入当前结点后增加的代价，然后生成一个新的子图加入候选集合中，并计算累积的代价. 第 13, 14 行仍是只保留列表中排序靠前的 k 个结果. 最后我们返回排名最高的一个子图.

算法 1 柱状搜索子图

输入:候选结点集合$\boldsymbol{Candidate_Q = \{C_1, \cdots, C_n\}}$
输出:排名最高的一个子图

```
1.  C ← ∪_{i=1}^{n} C_i
2.  sort C by weight, 按权值从高到低排序
3.  Beam ← Top(k, C), 取前 k 个权值最高的
4.  for i ← 2 to n do
5.    foreach V' ∈ Beam do
6.      foreach c ∈ C_i do
7.          delta ← Σ_{v∈V'} dist(c,v) / (w(c)*w(v))
8.          newV' ← V' ∪ {c}
9.          cost ← cost + delta, 增加该结点后子图的总的代价
10.         Add(newV', newBeam), 将结果加入到一个新的 beam 中
11.     end
12.   end
13.   sort NewBeam by cost, 按代价从低到高排序
14.   Beam ← Top(k, newBeam), 取前 k 个代价最小的
15. end
16. return Top(1, Beam)
```

2.4.2 子图的扩展与推荐

利用图嵌入的向量来计算距离，带来了计算的便利和速度的提升，但由于在选择结点的过程中我们并没有考虑实际存在的边，因此到目前为止，我们只得到了子图的顶点集. 为了给用户提供更形象地理解，我们应该将这些结点是如何连接起来的一同展示给用户，即从顶点集扩展成一个连通的子图.

本文利用将这个问题定义成给顶点集 V' 构造一棵最小生成树(Minimum Spanning Tree, MST). 将 V' 中任意两个顶点之间的最小跳数定义为它们之间的边的权重，这样做就意味着用尽可能少的边将所有顶点连接起来. 具体算法如下表的算法 2 所示，其中第 5 行的FindShortestPath函数即寻找结点集合 X 和 Y 之间最短路径，而每两个结点之间的最短路径可以通过 Cypher 语句在代码图中进行查询. 第 8, 9 行是将这条路径上所经过的结点和边都加入到扩展后的子图中. 直到所有结点都被包含进来，该算法便得到了一个连通的子图.

算法 2 扩展为连通子图

输入:顶点集 V'
输出:扩展后的子图 G^e
1. $V^e \leftarrow V', E^e \leftarrow \emptyset$
2. $X \leftarrow$ randomly select a node from V'
3. $Y \leftarrow V' - X$
4. while $Y \neq \emptyset$ do
5. $v, path \leftarrow$ FindShortestPath(X, Y)
6. $X \leftarrow X \cup \{v\}$
7. $Y \leftarrow Y - \{v\}$
8. $V^e \leftarrow V^e \cup path.V$
9. $E^e \leftarrow E^e \cup path.E$
10. end
11. $G^e \leftarrow (V^e, E^e)$
12. return G^e

3 实验与实例

基于上述方法, 本文设计并实现了基于图嵌入的软件 API 检索工具原型. 下面我们通过一些实验来验证本文所提出方法和工具的有效性. 这些实验主要用于回答下列研究问题:

研究问题 1:本文所提出的方法是否能够有效地定位用户所需要的 APIs?

本文方法将源代码组织成代码结构图的形式而非仅视为纯文本来处理, 其中利用了图嵌入技术, 定义了子图的度量函数并使用基于柱状搜索的算法进行代码子图的检索. 我们希望本文所提出的方法, 在回答开发者提出的实际自然语言问题时, 能够有效提升软件项目 API 检索的效果. 为了验证这一点, 我们选择了两个著名的开源项目以及与它们相关的自然语言问题进行了验证, 并将本文方法与传统基于文本匹配的 API 检索方法进行了对比.

研究问题 2:本文中所提出的图嵌入方法与现有其他检索方法相比效果如何?

在本文所提出的方法中, 我们使用了图嵌入方法来挖掘代码图的结构信息, 在度量结点之间的距离时使用图嵌入向量之间的距离, 并由此定义了子图的度量函数. 我们关心图嵌入方法是否能够有效地表达代码结构图中蕴含的语义信息, 从而改进代码检索的效果. 因此, 我们设计了相应的实验, 将本文所使用的图嵌入方法与现有的基于最短路径的方法进行了对比, 以验证图嵌入方法的有效性.

3.1 实验设计

我们选取了两个著名的开源软件项目——Apache Lucene 和 Apache POI 作为实验对象, 自动构造了这两个项目的代码图, 并分别对其进行了基于自然语言的代码检索实验. 实验设计的细节列举如下:

3.1.1 代码图构造

对于每个软件项目, 我们基于它的源代码中构建一个代码结构图. Lucene 和 POI 项目的源代码都可以通过其官网的连接或 Github 直接下载. 一个软件项目往往会有很多个不同的版本, 我们选取了这两个项目其中被广泛使用版本进行代码结构图的提取. 与代码结构图相关的统计信息在表 2 中进行了展示, 包括:源代码版本、类或接口的结点数量、方法的结点数量、关联关系的数量以及构造时间. 我们在一台 3. 40GHz 双核处理器、8GB 内存的服务器上, 解析生成两个项目的代码结构图的时间开销分别为 39 分钟和 31 分钟.

在 W. K. Chan 等人的工作中, 他们实验所使用的源代码数据是 JSE, 它是一个非常通用的底层函数库. 本文所关注的问题是在软件项目复用时如何有效进行 API 检索, 我们实验中使用的 Lucene 和 POI 是领域特定的函数库/软件项目. 基于功能实现的需要, 其 APIs 之间的关联往往非常紧密, 使得这类领域特定的软件项目在实际中的进行代码检索和复用的难度更大. 从表 2 中可以看出, Lucene 项目代码图中, 结点(类和方法等)数量是 39,000 左右, 但它们之间的关联关系数量达到了 255,000. 同时, 在 POI 项目的代码图中结点数和关联关系的数量关系也与此类似.

Table 2 Code graphs for experiments
表 2 实验所用项目的代码图信息

项目名称	源代码版本	类/接口结点数	方法结点数	总结点数	关联关系数	构造时间(分钟)
Lucene	6. 3. 0	5, 377	34, 042	39,419	255, 880	39
POI	3. 14	3, 678	32, 001	35,679	211, 692	31

3.1.2 API 检索问题

进行 API 检索的自然语言提问多种多样. 为了保证问题的真实有效性, 本文使用了两种客观的方式来获得实验所需的问题.

对于 POI 项目, 我们通过其官方网站上提供的用户指南[1], 抽取了 20 个问题作为第一组 Query. 每一个问题在用户指南中都有对应的示例代码, 因此我们将这些示例代码中涉及的 API 作为该问题的 ground truth 进行标注.

对于 Lucene 项目, 由于其官方网站上没有相关教程和示例代码, 我们使用 StackOverflow 上关于 Lucene 项目的问答帖子作为来源, 采取随机抽样的方法, 直到挑选出 20 个与源代码检索相关的问题作为第二组 Query. 具体抽取方法是:首先从 StackOverflow 上 2008-2016 年的数据中找出带有 Lucene 标签的问题, 并按照问题的投票为正, 问题有被接受的答案, 答案中含有代码片段的条件进行筛选, 得到了 923 问题. 再从这些候选问题中进行无放回的随机抽取, 人工阅读是否属于源代码检索相关的问题, 直到挑选出 20 个相关问题. 其次, 由于这些问题在 StackOverflow 上分为标题和内容两部分, 且标题基本上都可以作为其内容概括, 本文便以问题的标题作为代码检索工具的查询输入, 即 Query. 针对这 20 个 Query, 我们还需要确定其对应的目标 API. 为此, 我们组织了 3 名熟悉 Lucene 项目的研究生, 分别阅读了这些 Query 在 StackOverflow 上的问题和答案, 结合帖子的答案和源代码的相关知识, 在代码图上进行标注. 具体的标注过程是:开发人员以该问题答案中出现的代码片段为基准, 并结合源代码的相关文档, 利用代码图数据库的查询结果, 最终确定该 Query 对应的 API 集合.

本文实验所使用的问题是开发者在实际开发过程中遇到的、真实存在的、用自然语言表述的问题. 其中部分问题及其标注如表 3 所示, 问题之中可能含有代码元素的名称, 譬如 lengthNorm, 也可能没有. 因此我们认为对这些问题的解答可以很好地反映各种方法的实际效果.

3.1.3 比较对象

为了考察本文工作的实际效果, 我们首先分析比较了本文方法在检索过程中选择第一个结果和选择前三个结果的情况.其次, 我们将这两种情形与其它两种典型的代码检索方法进行了对比. 实验方法包括:

(1) **本文方法（Top 1）**. 这一检索方法只采用了本文基于图嵌入的代码检索方法的第一个结果.
(2) **本文方法（Top 3）**. 这一检索方法采用了本文基于图嵌入的代码检索方法的前三个结果.
(3) **基于文本匹配的方法**. 这一检索方法只考虑问题与结点的文本相似度进行匹配, 而不考虑结点之间的距离因素, 即 2. 4. 1 节中子图的度量函数公式(5)中只含权重部分而没有距离部分.
(4) **基于最短路径的方法**. 这一检索方法来源于 W. K. Chan 等人的工作, 其采用最短路径来计算两个结点之间的距离, 而非本文方法中使用的图嵌入向量之间的距离.

[1] https://poi.apache.org/spreadsheet/quick-guide.html

Table 3 Examples of Query and annotated APIs

表 3 问题与标注 APIs 示例

编号	问题	标注的 APIs	所属项目
1	How to set document boost attribute?	Document, Attribute, BoostAttribute, AttributeImpl, BoostAttributeImp. setBoost	Lucene
2	How to merge several taxonomy indexes?	TaxonomyMergeUtils, IndexWriter, TaxonomyMergeUtils. merge	
3	How is lucene query boost affected by lengthNorm similarity?	Similarity, TFIDFSimilarity, BoostQuery, TFIDFSimilarity. lengthNorm	
4	Sorting on a numeric field in Lucene.	Field, SortField, SortedNumericSortField	
5	How to create a query using wildcards?	Query, MultiTermQuery, AutomatonQuery, WildcardQuery	
6	How to escape some characters utilizing QueryParser?	QueryParser, QueryParserBase, QueryParserBase. escape	
7	Get string representation of Terms using HighFreqTerms.	HighFreqTerms, Terms, Terms. iterator, HighFreqTerms. getHighFreqTerms, BytesRef. toString	
8	How to get the score of top document in lucene?	ScoreDoc, TopDocs, Document, TopDocs. getMaxScore	
9	How does regexp query work on lucene?	RegexpQuery, automaton. RegExp, AutomatonProvider	
10	How to implement a fuzzy search query in lucene?	FuzzyQuery, Query, MultiTermQuery	
31	How to create workbook and sheet in POI?	Workbook, Workbook. createSheet	POI
32	How to create and set cell style in a workbook?	Workbook, Workbook. createCellStyle, CellStyle, Cell. setCellStyle, CellStyle. setDataFormat	
33	How to set bottom border of a cell?	Cell, Cell. setCellStyle, CellStyle, CellStyle. setBottomBorder	
34	Get document summary information in a workbook.	Workbook, HSSFWorkbook, Workbook. getDocumentSummaryInformation	
35	How to set header center in a sheet?	Sheet, XSFSheet, Header, Sheet. getHeader, Header. setCenter	
36	Set page number right on the footer.	Footer, HSSFFooter, HeaderFooter, HeaderFooter. page, HeaderFooter. numPages, Footer. setRight	
37	Set italic font in a workbook.	Workbook, Workbook. setFont, Font, Font. setItalic	
38	How to set hyperlink in a cell?	Cell, Cell. setHyperlink, Hyperlink	
39	Get all pictures data from a workbook.	Workbook, Workbook. getAllPictures, PictureData, PictureData. getData	
40	How to set repeating row in a sheet?	Sheet, Row, Sheet. setRepeatingRows	

3.1.4 评价指标

为了评价检索结果的好坏, 本文使用了准确率(Precision), 召回率(Recall)和 F1-值(F1-score)作为评价指标. 准确率和召回率的计算与真阳性(True positives)等概念有关, 其定义见下表 4 所示. 在本文的实验中, 准确率和召回率的定义如公式(6~8)所示. 其中 V_H标注结果的结点集合, V_G本文工具返回结果的结点集合. 这里的 True positives 即两个集合的交集, F1 值即准确率和召回率的调和平均数.

Table 4 Definitions for search results

表 4 检索结果的度量指标

	正确的	不正确的
预测到的	true positives(tp)	false positive(fp)
未预测到的	false negatives(fp)	true negatives(tn)

$$P(G) = \frac{tp}{tp+fp} = \frac{|V_G \cap V_H|}{|V_G|} \tag{6}$$

$$R(G) = \frac{tp}{tp+fn} = \frac{|V_G \cap V_H|}{|V_H|} \tag{7}$$

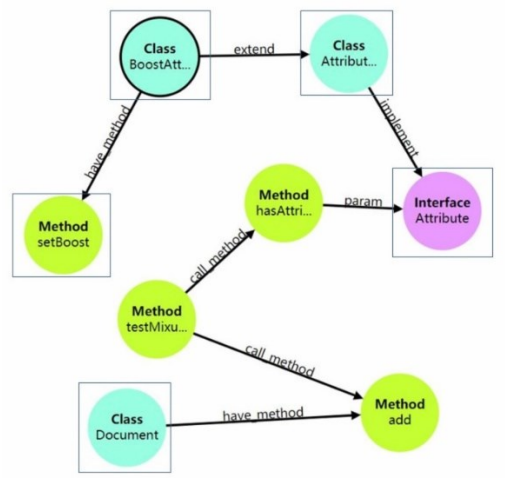
$$F = \frac{2 \cdot P \cdot \mathrm{R}}{P+\mathrm{R}} \tag{8}$$

3.2 检索实例

针对表 3 中的实验问题 1“how to set document boost attribute in Lucene？”，本文在 Lucene 软件项目的代码图上进行了检索实验. 首先，这个问题被预处理后的单词集合为{set, document, boost, attribute}，对于集合中每一个单词，根据 2.3.2 节中提到的匹配方法，分别得到了{1623, 1101, 63, 122}个对应的候选结点，可以看出候选结点的规模相当庞大. 接着，我们针对每个候选结点计算与问题相关程度作为该结点的权重. 譬如，*setBoost* 结点的权重为 0.75，*Attribute* 结点的权重为 0.4，在这些候选结点的基础上，按照 2.4 节中的算法构造问题检索对应的子图. 这里分为两个部分：

1) 子图的生成：基于离线完成的图嵌入向量的结果可以计算两个结点之间的距离，譬如 *Document* 结点和 *setBoost* 结点之间距离为 13.47，根据本文定义的目标函数通过柱状搜索算法可以得到子图中应该包含的候选结点为{*Document, Attribute, setBoost*}.

2) 子图的扩展：采用基于最小生成树的方法将子图中的结点扩展为连通的代码子图，得到的结果如图 4(a)所示，最终的连通子图共包含了 8 个结点.



(a)

```java
/** Implementation class for {@link BoostAttribute}.
 * @lucene.internal
 */
public final class BoostAttributeImpl extends AttributeImpl implements
  private float boost = 1.0f;

  @Override
  public void setBoost(float boost) {
    this.boost = boost;
  }

  @Override
  public float getBoost() {
    return boost;
  }

  @Override
  public void clear() {
    boost = 1.0f;
  }

  @Override
  public void copyTo(AttributeImpl target) {
    ((BoostAttribute) target).setBoost(boost);
  }

  @Override
  public void reflectWith(AttributeReflector reflector) {
    reflector.reflect(BoostAttribute.class, "boost", boost);
  }
}
```

(b)

Fig 4 Search result for query ”how to set document boost attribute”

图 4 检索示例”how to set document boost attribute”

根据问题的检索结果, 我们可以计算对应的准确率、召回率和 F1 值. 在下图中, 用蓝色方框标出的 5 个结点是该问题被人工标注的 API, 而其余 3 个结点则不属于被标注的. 所以该检索结果的准确率为 5/8. 而由于蓝色方框的 5 个结点刚好覆盖了全部标注的 API, 因此该结果的召回率为 1, 最终得到的 F1 值为 10/13.

本文检索结果是一个子图, 不仅展示了用户需要的目标 APIs, 还展示了其与相关代码元素的关联. 在此基础上, 用户可以通过点击图中的 API 结点显示其对应的代码片段. 如图 4(b)所示, 就是在上述检索结果中点击结点 *BoostAttributeImpl* 后, 能够看到其对应的代码体。由此, 该检索结果能为用户更好地理解和复用 APIs 提供有效支持.

3.3 实验结果

针对上述得到的两组各 20 个问题, 本文对比了上述四种方法检索结果的平均准确率, 召回率和 F1 值等评价指标, 实验结果如表 5 所示. 实验环境是在 Windows 10 系统, 2. 30GHz 的双核处理器, 8GB 内存, Java 版本是 1. 8. 实验中图嵌入向量的维度设置为 200, 柱状搜索中的柱大小(beam size)设置为 8.

3.3.1 方法效果对比

从表 5 中可以看出, 在 Lucene 和 POI 这两个项目中, 本文方法的 Top3 都同时获得了最高的平均准确率、召回率和 F1 值. 而本文方法 Top 1 与 Top 3 的结果相差很小, 说明对于大部分问题本文方法的 Top 1 就是最佳的结果.

基于文本匹配的方法在两个项目中都获得了最低的平均准确率、召回率和 F1 值, 表明单纯利用问题与结点的文本相似度而忽视代码结构图中的语义信息, 难以取得很好的效果. 而本文引入图嵌入技术挖掘代码图结构的方法之后, 检索结果得到了显著的提升.

基于最短路径的方法在检索过程中也利用到了代码结构图的信息, 只是在计算结点之间的距离时使用的最短路径, 而非本文方法中使用的图嵌入向量之间的距离. 该方法的效果与纯文本匹配方法相比有了较大提升, 但仍然显著低于本文方法的结果. 这表明了本文图嵌入方法的有效性, 能够比最短路径方法挖掘到更全面的图结构信息.

此外, 我们还对比了不同方法的平均响应时间和最长响应时间. 从表 5 中可以看出, 纯文本匹配方法的效率最高, 平均响应时间和最长响应时间都非常短, 因为其计算十分简单, 然而其检索效果也是最不令人满意的. 而最短路径方法的平均响应时间则非常长, 超过了 1 分钟, 这对于实际的用户交互而言难以接受. 这是因为该方法每次计算距离都需要在线查询两个结点之间的最短路径, 使用广度优先搜索的算法最坏情况下也需要 O(N)的复杂度, 这里的 N 表示所有结点的数量. 图嵌入过程由于离线已经完成, 在线计算两个向量的距离时只需要常数时间, 所以本文方法在改进检索效果的同时也大大提升了运行的效率.

Table 5 Experiment results analysis

表 5 实验结果分析

方法名称	Lucene 项目检索			POI 项目检索			效率分析	
	准确率	召回率	F1 值	准确率	召回率	F1 值	平均响应时间	最长响应时间
本文方法 Top 1	0. 53	0. 85	0. 63	0. 70	0. 81	0. 74	1. 39s	3. 42s
本文方法 Top 3	0. 55	0. 86	0. 65	0. 72	0. 83	0. 76	1. 39s	3. 42s
基于文本匹配的方法	0. 41	0. 57	0. 43	0. 43	0. 62	0. 49	0. 13s	0. 80s
最短路径方法	0. 47	0. 62	0. 52	0. 65	0. 66	0. 64	86. 68s	257. 18s

W. K. Chan 等人提出的方法[12]是基于最短路径方法, 他们使用的子图度量函数与本文有所差别, 他们在论文中报告的实验结果为准确率 0. 53, 召回率 0. 56, F1 值 0. 54, 与本文实验中最短路径方法的结果十分接近. 我们使用的数据集并不相同, 他们实验中所使用的问题是从 KodeJava 网站上挑选的关于使用 JSE 编程的 20 个示例, 并且将问题抽取为多个词组的形式作为输入. 而本文实验中使用的 Lucene 和 POI 则是领域特定的函

数库，输入也直接是自然语言的问题.

3.3.2 准确率与召回率分析

从实验结果来看，本文方法检索的召回率比较高，但准确率比较低. 也就是说，代码子图中除了与 Query 相关的结点之外，还包含了较多无关的结点. 特别是对于 Lucene 项目而言，准确率和召回率的差别比较大，因此我们对 Lucene 项目的 20 个问题进行了进一步的分析.

首先，我们对每个问题的准确率进行了具体分析，结果如图 5 所示. 其中，斜线底色的柱形表示检索结果中所有的结点(API)数量，纯色柱形表示检索结果中正确的结点数量. 可以看到，在我们的检索结果中，通常更多的 API 被返回和推荐出来，大部分子图的规模(结点数)都在 5-8 之间，只有两个问题的子图大小达到了 10. 经分析发现，导致这个问题的主要原因是基于最小生成树的子图扩展算法. 由于很难断定哪种类型边更加有益，我们将所有边的权值都设为相同，从而产生了大量相同长度的路径,因而在扩展路径上很可能引入没有被标注的结点. 虽然我们不能确定这些更多的 API 是必要的，但从召回率较高的事实出发，也就是在保证能够返回用户所需 API 的情况下，我们相信提供更多的信息对用户理解和使用这些 API 仍然是有益的. 而对子图扩展算法的优化可以作为本文的进一步工作.

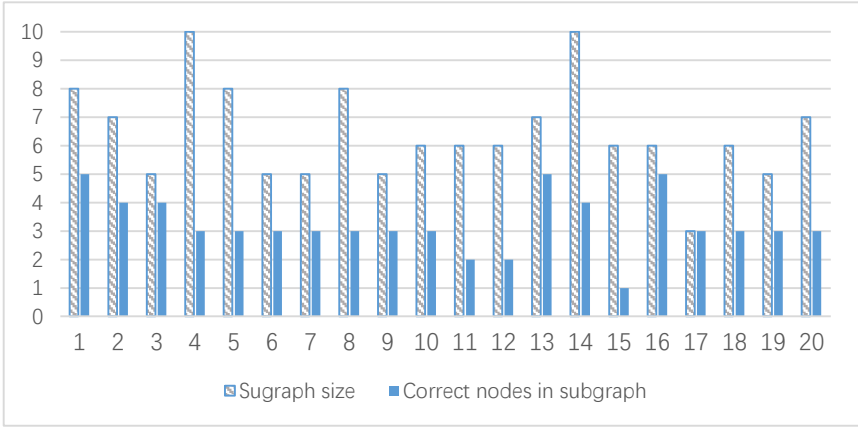


Fig 5 Precision of each question

图 5 每个问题的准确率

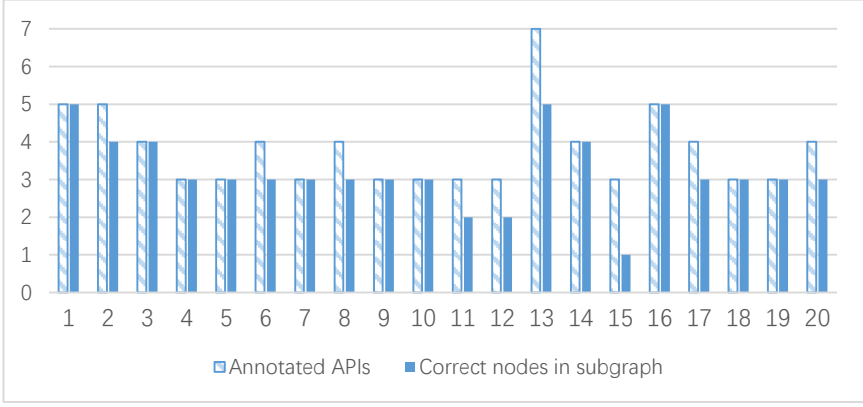


Fig 6 Recall of each question

图 6 每个问题的召回率

我们对每个问题的检索召回率也进行了具体分析，结果如图 6 所示. 其中，斜线底色的柱形表示人工标注

时的 API 结点数量, 纯色柱形表示检索结果中正确的结点数量. 从图中可以看出, 两种柱形高度十分接近, 表明整体的召回率很高. 对于其中的 10 个实验示例而言, 召回率都达到了 100%, 而召回率比较低的只有编号为 13 和 15 的两个问题, 柱形高度之差为 2. 我们对这些召回率偏低的问题进行了分析, 譬如编号为 15 的问题 “How to query document have any term in a set?”, 这里影响其实验效果的因素是自然语言的歧义性, 问题中的 set 是指集合的意思, 而源代码中则存在大量将 set 作为动词使用的方法名.由于本文在解析问题时只采用了简单的词袋模型, 目前尚无法很好地解决这类的歧义问题.本文工具最终匹配到的是 setTerm 方法, 因为该方法含有两个与问题相同的词, 结点的权重就很高. 在未来工作中,我们将考虑利用代码结构图和图嵌入技术来尽可能消除自然语言歧义, 进一步提升代码检索的效果.

3.4 有效性讨论

实验评估的合理性. 本文工作目标是提高软件项目 API 检索与复用的效率, 检索结果是一个代码子图. 我们关注该子图中的 API 及其关联结点是否能够解决开发者的问题. 本文参考 W.K. Chan 等人的实验指标, 使用准确率、召回率和 F 值作为主要评价, 而非信息检索中的 MRR 等排名相关指标. 而在进行问题答案的标注过程中, 我们从该问题对应的示例代码中提取 APIs 并通过交叉验证等方式保证标注结果的正确性和客观性. 由于实验的问题都对应有被接受的代码片段, 因此如果检索的子图中包含了其中的 APIs, 那么我们认为它的确能够帮助开发者解决问题. 在未来工作中, 我们也将考虑进行用户研究(User Study), 以进一步定量探讨给开发者带来的提升效果.

影响实验结果的内部因素. 关于检索实例. 本文针对 Lucene 和 POI 项目, 分别通过两组代码检索实例进行了实验, 共 40 个问题. 尽管问题数量有限, 但我们相信这些问题具有很好的代表性, 不会影响实验结论. 首先, 我们选取的实验问题源于实际, 一组问题来源于项目官网的用户指南, 另一组问题来源于 StackOverflow 的帖子. 这些问题都是开发过程中真实存在的, 并且都有相应的代码片段, 能够保证问题的较高质量. 而对于帖子的筛选和随机抽取也能够保证其代表性. 其次, 问题数量对实验效果没有明显影响. 在实验 3.3.2 中可以看到, 单个检索实例的评价指标结果与所有实例的结果差异不大. 在相关工作, 检索实例的数量基本与本文相似. 譬如 CodeHow[8] 使用了 34 个问题, W.K. Chan[12] 使用了 40 个问题.

影响实验结果的外部因素. 关于实验对比方法. 由于本文不仅需要定位目标 APIs, 还需要展示其与相关代码元素的关联。为此，检索结果是一个子图. 返回代码片段或 API 列表的相关工作不适合与我们直接进行比较. W.K. Chan 等人的工作是目前与本文场景最接近的工作, 其方法的本质是基于最短路径的. 因此本文实验中以最短路径方法作为比较, 能够验证本文方法的有效性.

4 相关工作

本文的相关工作主要包括对源代码 API 检索和图嵌入技术两部分, 图嵌入技术已在第一节进行了介绍. 本节主要介绍源代码检索与 API 推荐的相关技术, 重点关注的场景是针对源代码的基于自然语言问题检索.

早期的源代码检索工作主要是将源代码视为纯文本,考虑查询问题与代码之间的文本相似度进行匹配, 利用了 TF-IDF、BM25 等信息检索技术. Krugle[4] 、Ohloh[5] 和 Sourcerer[6] 就是其中的典型代表. Sourcerer 将解析后的源代码存储在本地的关系型数据库中并利用 Lucene 进行索引. 这些方法的查询方式通常是基于关键词, 返回结果是包含查询中关键词或正则表达式的代码片段. 然而由于其缺乏对问题语义的理解, 这些工具的实际效果往往不够令人满意.

针对上述问题, 近来有许多研究工作结合了自然语言处理技术, 从深化问题语义理解的角度提出了一系列的查询精炼（Query Refinement）技术应用于代码检索, 包括查询扩展（Query Expansion）、查询重构(Query Reformulation)等等. 典型的, COCAB[25] 从 StackOverflow 帖子含有的代码片段中提取结构化信息来增强用户的查询, 以此解决自然语言与代码元素的词汇不匹配问题. SNIFF[26] 先为代码片段中的 APIs 寻找到相应文档作为其注解, 继而在这些被注解过的代码上进行自然语言的查询. CodeHow[8] 则是通过搜索在线文档识别出一些与问题相关的潜在 APIs, 然后使用扩展布尔模型将这些 APIs 的信息结合到代码检索的过程中去, 从而提

高准确度. 还有一些方法通过加入语义上相似的词来扩展用户的自然语言查询, 这些相似的词可以来源于对网页[27] 或者软件项目[11, 27]的挖掘. DERECS[30] 就预先构造了一个包含大量代码-描述对的语料库, 利用它对缺少注释的代码进行描述增强. 然而有研究表明, 如果在扩展查询时引入了一些不合适的词, 将会导致更为糟糕的结果[29] . 另一些方法则扩展了用户的查询方式, Gu[31] 等人的工作中, 用户除了可以使用关键词进行查询, 还可以加入代码的语法和语义约束条件, 从而实现更精确地检索. Wang 等人[32] 提出了一种动态的检索方法, 能够结合用户的反馈信息来优化查询. Haiduc 等人[33] 提出了一个名为 Refoqus 的工具, 能够预测用户查询的质量并自动推荐查询重构的策略. BIKER[34] 从 StackOverflow 上抽取了相似的问题和 API 并利用词嵌入(Word Embedding)来计算文本描述的相似度. 基于机器学习的代码检索和 API 推荐工作也是目前的一个研究热点. 譬如, ROSF[35] 先通过信息检索的方法生成候选集, 再利用有监督学习为候选结果进行重排序. DEEPAPI[36] 使用 RNN Encoder-Decoder 模型, 将自然语言问题翻译到 API 调用序列. Function Assistant[37] 则利用语义解析的技术, 从代码-文本对中提取特征学习翻译模型. APIRec[38] 从细粒度的代码修改库中训练统计学习的模型, 基于上下文上进行 API 推荐. RecRank[39] 则引入路径相关的特征对 APIRec 的结果进行重排序, 从而提升了 Top 1 推荐的准确率. 虽然这些技术很大程度上提高了 API 定位的准确度, 但其需要积累和利用大量的辅助信息, 在上述信息缺乏的情况下, 往往不能达到预期效果. 在本文工作中, 我们尤其关注了在没有辅助文档信息和大量查询历史的前提下,如何借助代码自身蕴含的语义信息来辅助提升代码检索的效果.

由于现有代码检索工具往往返回问题相关的代码片段, 缺乏对关联 API 的说明与展示, 不便于用户理解目标 APIs 调用背后的逻辑关联, 如调用链、类图等, 来理解整体的概念[40, 41]. 为此, 一些工作开始关注如何在定位与问题相关 APIs 的同时, 整合、利用和展示代码元素之间的关联信息. 典型地, C. McMillan 和 W. K. Chan 等人的工作就将代码组织成有向图的结构, 将代码检索问题转化为图上的搜索问题. C. McMillan 等人提出的工具 Portfolio[11], 利用了 PageRank 和 SAN（Spreading Activation Network）算法, 返回图上与问题相关的前 k 个结点作为答案. 而 W. K. Chan 等人的工作[12] 则更进一步地保证了返回结果是一个连通的子图, 从而能够清晰地展现出各个结点之间的连接关系. RACS[42] 针对 JavaScript 框架, 构建方法之间的调用关系图, 并将自然语言问题也抽取为图形式, 寻找与问题的图结构相似的代码片段. 许多其他工作也将同样地思想用于文档检索、特征定位等诸多场景[43, 44]. 受这些工作的启发, 本文也将源代码组织为有向图的结构, 利用了图嵌入技术来挖掘图结构的深层信息, 能够更充分地考虑代码图的上下文信息, 同时由于图嵌入可以离线完成, 提高了在线查询算法的效率.

5 总结与展望

为了帮助用户更好地检索和复用软件项目 APIs, 本文提出了一种基于图嵌入的软件项目 API 检索方法. 图结构能够很好地反映软件项目源代码中 APIs 之间的关联, 帮助用户理解软件项目; 图嵌入技术能够更充分地挖掘图结构的深层信息, 并且可以离线完成, 有效地提高检索过程中生成子图的效率. 基于上述方法, 我们设计并实现了相应的软件项目 API 检索工具, 并以开源项目 Apache Lucene 和 POI 为例, 通过相关实验验证了本文方法的有效性.

在未来的的工作中, 我们将在图嵌入技术的基础上, 进一步改进问题与结点的匹配方法, 利用词性、句法等自然语言处理技术更好地挖掘问题的语义; 同时, 在匹配过程中增加对边的类型的分析, 在扩展子图时考虑路径上其余结点与问题的相关度, 以减少不相关结点的引入, 从而提升检索的准确率. 此外, 我们将进一步探讨在实际过程中用户与搜索结果之间的交互会面临的问题，增强搜索结果的可读性.

References:

[1] C. Scaffidi. Why are apis difficult to learn and use? *ACM Crossroads*, 12(4):4, 2006.

[2] R. Hoffmann, J. Fogarty, and D. S. Weld. Assieme: Finding and leveraging implicit references in a web search interface for programmers. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*, pp. 13–22, 2007.

[3] J. Stylos and B. A. Myers. Mica: A web-search tool for finding API components and examples. In *Proceedings of the Visual Languages and Human-Centric Computing (VLHCC '06)*, pp. 195–202, 2006.

[4] Krugle code search. [Online]. Available: http://www. krugle. com/

[5] Ohloh code search. [Online]. Available: https://code. ohloh. net/

[6] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, vol. 18, pp. 300–336, 2009.

[7] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval: The Concepts and Technology behind Search. Addison-Wesley, 2011.

[8] Fei Lv, Hongyu Zhang, Jian-guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. Codehow: Effective code search based on api understanding and extended boolean model. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE '15)*, pp. 260-270, 2015.

[9] E. Hill, L. L. Pollock, and K. Vijay-Shanker. Improving source code search with natural language phrasal representations of method signatures. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE ' 11)*, pp. 524–527, 2011.

[10] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE ' 13)*, pp. 842–851, 2013.

[11] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu. Portfolio: finding relevant functions and their usage. In *ICSE*, pp. 111–120, 2011.

[12] W. K. Chan, H. Cheng, David Lo. Searching connected API subgraph via text phrases. In *SIGSOFT FSE*, 2012.

[13] Lin ZQ, Zou YZ, Zhao JF, Cao YK, Xie B. Software Text Semantic Search Approach Based on Code Structure Knowledge. Ruan Jian Xue Bao/Journal of Software, 2017 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

[14] P. Goyal, E. Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. arXiv:1705. 02801 [cs. SI], 2017.

[15] M. Belkin, P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In NIPS, Vol. 14, pp. 585–591, 2001.

[16] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, A. J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web(WWW' 13)*, pp. 37–48, 2013.

[17] B. Perozzi, R. Al-Rfou, S. Skiena. Deepwalk: Online learning of social representations. *In Proceedings 20th international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

[18] D. Wang, P. Cui, W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, ACM, pp. 1225–1234, 2016.

[19] T. N. Kipf, M. Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609. 02907. Published as a conference paper at ICLR 2017.

[20] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei. Line: Largescale information network embedding. In *Proceedings 24th International Conference on World Wide Web(WWW' 15)*, pp. 1067–1077, 2015.

[21] http://checkstyle. sourceforge. net/

[22] http://www. csg. ci. i. u-tokyo. ac. jp/~chiba/javassist/

[23] http://www. scovetta. com/yasca. html/

[24] http://www. eclipse. org/jdt/

[25] R. Sirres, T. F. Bissyand´e, Dongsun Kim, David Lo, J. Klein, Kisub Kim, Yves Le Traon. Augmenting and structuring user queries to support efficient free-form code search. *Empirical Software Engineering*, Jan. 2018.

[26] S. Chatterjee, S. Juvekar, and K. Sen. Sniff: A search engine for Java using free-form queries. In *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering (FASE ' 09)*, pp. 385–400, 2009.

[27] Y. Tian, D. Lo, and J. L. Lawall. Automated construction of a software specific word similarity database. In Proc. of CSMR-WCRE, pp. 44–53, 2014.

[28] J. Yang and L. Tan, "Inferring semantically related words from software context," in 9th IEEE Working Conference of Mining Software Repositories (MSR'12), pp. 161–170, 2012.

[29] G. Sridhara, E. Hill, L. L. Pollock, and K. Vijay-Shanker. Identifying word relations in software: A comparative study of semantic similarity tools. In *Proceedings of the 16th IEEE International Conference on Program Comprehension (ICPC '08)*, pp. 123–132, 2008.

[30] Li X, Wang QX, Jin Z. Description reinforcement based code search. Ruan Jian Xue Bao/Journal of Software, 2017,28(6):1405−1417 (in Chinese). http://www.jos.org.cn/1000-9825/5226.htm

[31] Gu YS, Zeng GS. Accurate search method for source code by combining syntactic and semantic query. Journal of Computer Applications, 2017, 37(10):2958-2963.

[32] S. Wang, D. Lo, and L. Jiang. Active code search: Incorporating user feedback to improve code search relevance. In *Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering (ASE '14)*, 2014.

[33] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE' 13)*, pp. 842 – 851, 2013.

[34] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, Xinyu Wang. API Method Recommendation without Worrying about the Task-API Knowledge Gap. In *Proceedings of the 2018 33rd International Conference on Automated Software Engineering (ASE' 18).pp. 292-303, 2018.*

[35] Jiang H, Nie L, Sun Z et al. ROSF: Leveraging information retrieval and supervised learning for recommending code snippets. IEEE Transactions on Services Computing, 2016.

[36] Gu XD, Zhang HY, Zhang DM, Kim SH. Deep API learning. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering. ACM Press, pp. 631−642, 2016.

[37] Kyle Richardson and Jonas Kuhn. Function Assistant: A Tool for NL Querying of APIs. In Proceedings of EMNLP, 2017

[38] A.T. Nguyen, M. Hilton, M. Codoban, H.A. Nguyen, L. Mast, E. Rademacher, API Code Recommendation using Statistical Learning from Fine-Grained Changes. In *Proceedings of the 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*. ACM, 511–522, 2016.

[39] Xiaoyu Liu, LiGuo Huang, Vincent Ng. Effective API Recommendation without Historical Software Repositories. In *Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering* (ASE '18), 2018

[40] J. Sillito, G. C. Murphy, and K. De Volder. Asking and answering questions during a programming change task. *IEEE Trans. Softw. Eng.*, 34(4):434–451, 2008.

[41] J. Sillito, G. C. Murphy, and K. D. Volder. Questions programmers ask during software evolution tasks. In *SIGSOFT FSE*, pages 23–34, 2006.

[42] Li X, Wang ZR, Wang QX, Yan SM, Xie T, Mei H. Relationship-Aware code search for JavaScript frameworks. In: Proc. of the 24th ACM SIGSOFT Symp. on the Foundations of Software Engineering. pp. 690−701, 2016.

[43] Fu K, Wu YJ, Peng X, Zhao WY. A feature location method based on call chain analysis. Computer Science, 2017, 44(4):56-59.

[44] Li Z, Niu J, Wang K, Xin YY. Optimization of Source Code Search Based on Multi-feature Weight Assignment. Journal of Computer Applications, 2018, 38(3):812-817.

附中文参考文献:

[13] 林泽琦, 邹艳珍, 赵俊峰,曹英魁, 谢冰.基于代码结构知识的软件文档语义搜索方法.软件学报.已接收.

[30] 黎宣, 王千祥, 金芝.基于增强描述的代码搜索方法.软件学报,2017,28(6):1405−1417.

[31] 顾逸圣, 曾国荪. 基于语法和语义结合的源代码精确搜索方法[J].计算机应用, 2017, 37(10):2958-2963.

[40] 付焜, 吴毅坚, 彭鑫,等. 一种基于子图搜索的特征定位方法[J].计算机科学, 2017, 44(4):56-59.

[41] 李阵, 钮俊, 王奎,等. 基于多特征权重分配的源代码搜索优化[J].计算机应用, 2018, 38(3):812-817.