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

智能化的程序搜索与构造方法综述*

刘斌斌, 董威, 王戟

(国防科技大学 计算机学院,湖南 长沙 410073)

通讯作者: 董威, E-mail: wdong@nudt.edu.cn

摘 要: 互联网、机器学习、人工智能等技术的迅速发展以及大量开源软件和开源社区的出现给软件工程的发展带来了新的机遇和挑战.目前在互联网上已经存在了数十亿行的各类程序代码,这些代码中存在着各种知识,尤其是众多已被广泛使用、高质量的软件代码,由此催生了利用大规模代码资源中蕴涵的众多知识进行智能化软件开发的新思路.它试图充分利用互联网中存在的代码资源、知识和群体智慧,以有效提高软件开发的效率和质量,其核心技术是程序搜索与构造,具有非常重要的理论与应用价值.目前该方向的研究工作主要集中在代码搜索、程序合成、代码推荐与补全、缺陷检测、代码风格改善、程序自动修复等方面.本文从以上几个方面对当前的主要研究工作进行综述,对具体的理论和技术途径进行梳理,并在最后总结了目前该领域研究过程中面临的挑战,给出了建议的研究方向.

关键词: 程序搜索;程序构造;群体智慧;智能化软件开发

中图法分类号: TP311

中文引用格式: 刘斌斌,董威,王戟.智能化的程序搜索与构造方法综述.软件学报. http://www.jos.org.cn/1000-9825/0000.htm

英文引用格式: Liu BB, Dong W, Wang J. A Survey of Intelligent Search and Construction Methods of Program. Ruan Jian Xue Bao/Journal of Software, 2017 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

A Survey of Intelligent Search and Construction Methods of Program

LIU Bin-Bin, DONG Wei, WANG Ji

(School of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: The rapid development of Internet, machine learning and artificial intelligence as well as the appearance of a large number of open-source software and communities have brought new opportunities and challenges to the development of software engineering. There are billions of lines of code on the Internet. These code, especially those of high quality and widely been used contains all kinds of knowledge, which has led to the new idea of intelligent software development. It tries to make full use of code resources, knowledge and collective intelligence on the Internet to effectively improve the efficiency and quality of software development. The key technology, program search and construction which is of great theoretical and practical value. At present, the research work of these areas mainly focuses on the aspects of code search, program synthesis, code recommendation and completion, defect detection, code style improvement, automatic program repair, etc. This paper surveys the current main research work from the above aspects, sorts out the specific theoretical and technical approaches in detail and summarizes the challenges in the current research process. Finally, we propose several directions of research in the future.

Key words: program search; program construction; collective intelligence; intelligent software development

* 基金项目: 国家自然科学基金(61690203, 61532007); 国家 973 重大基础研究项目(2014CB340703).

Foundation item: National Natural Science Foundation of China (61690203, 61532007); 973 National Program on Key Basic Research Project (2014CB340703).

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

CNKI 在线出版时间: 0000-00-00

1 绪言

“软件正吞噬着整个世界”.互联网技术的快速发展给软件工程领域带来了新的机遇和挑战.一方面,互联网技术的发展将软件的生态环境变得更加开放、动态、复杂并且持续演化,使得软件系统的规模和复杂性急剧增长,对提高软件开发的效率和质量提出了严峻的挑战;另一方面,互联网技术的发展催生了以 Linux 等开源软件、GitHub[1]等开源社区、TopCoder[2]等众包软件开发以及 Stack Overflow[3]等编程问答网站等为代表的新型群体化软件开发实践平台和系统,为人类个体之间的交互和协作提供了更好的机会.因此,研究如何充分利用互联网中已有的代码资源以及群体的智慧和能力,对有效提高软件开发的效率和质量具有非常重要的意义.

程序自动构造也常称为代码自动生成,一直以来被认为是提高软件开发自动化程度和最终质量的重要方法,得到了学术界和工业界的广泛关注.程序自动构造是指利用某些机制或方法自动地为计算机软件产生源代码,以达到机器编程的目的.该方法使得程序员能够在更高的抽象层次上进行程序的设计工作,以提高程序开发的效率.另外,自动生成的程序代码具有良好的编程规范,不容易出现错误,有助于提高软件开发的质量,同时也能够降低软件后期维护的成本.

传统的程序自动构造技术多是采用以形式化方法为基础的技术,主要有模型驱动的软件开发以及基于逻辑规约的程序合成(Program Synthesis)等.模型驱动的软件开发能够降低软件开发的复杂性,利用领域特定模型语言(Domain-Specific Modeling Languages)描述系统,然后利用转换器和生成器自动生成各种类型的软件制品(主要指程序代码)[4].基于逻辑规约的程序合成利用用户提供的规约,通过定理证明等形式化方法来产生程序代码[5].这些传统的方法要求开发人员专注于抽象层次更高的模型设计以及逻辑规约的书写,将编写代码的工作交由机器完成,从而给软件开发带来便利,提高工作效率.但同时我们也看到,传统的程序自动构造方法存在着较大局限:

(1) 依然依赖开发人员设计复杂的模型架构或者逻辑规约,对开发人员要求较高;

(2) 特定程序模型和构造技术面向的问题空间受限,通用性差,难以适用于不同类型的需求;

(3) 对于大规模复杂软件系统,传统程序自动构造方法的可扩展能力不足,难以有效生成复杂代码.

近年来,随着大量开源软件项目与开源社区的发展,目前在互联网上已经存在着数十亿行的各类程序代码,甚至在很多大型企业内,积累的代码量也已经非常巨大(如 Google 的代码库已经超过了 20 亿行代码[6]),进而出现了与大数据(Big Data)相对应的大代码(Big Code)[89]的概念.人们认识到这些大量代码中存在着各种知识,尤其是众多已被广泛使用、高质量的软件代码.这些成果都是人类智慧的结晶,如果能够充分加以利用,将为软件开发工作提供很大帮助.正如互联网中海量的数据使得大数据的应用成为热点,如今互联网中大量的代码和资源催生了利用大规模代码资源库中已有的知识来进行智能化软件开发的新思路.随着机器学习技术的快速发展,我们正在进入人工智能的时代.机器学习技术目前在很多方面如语音识别、图像识别、自然语言处理以及棋牌博弈等方面的能力都已接近甚至已经超过了人类,但是对于程序开发这种智力密集型工作的应用还存在着巨大的挑战.

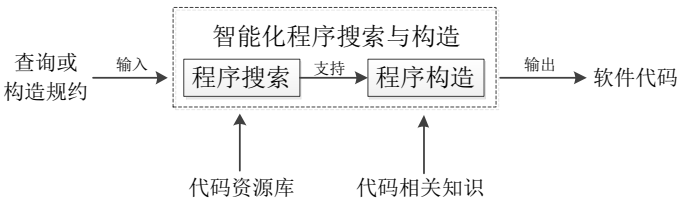

Fig1. Intelligent Program Search and Construction Methods

图 1 智能化的程序搜索与构造方法

互联网、机器学习、人工智能等技术以及开源软件和开源社区的飞速发展,给程序自动构造技术带来了新的技术途径.智能化的程序搜索与构造就是要利用这些新兴技术,让机器根据已经存在的大量代码知识和资源,在人类程序员的一定引导下进行代码构造.这将使得人们传统所追求的程序自动化构造发展为智能化构造.如

图 1 所示,智能化的程序搜索与构造包括智能化的程序搜索与智能化的程序构造.其输入是查询或构造规约,规约的形式可以是自然语言描述、程序结构的描述、测试用例以及其他约束条件等.通过智能化程序搜索与构造输出满足需求的软件代码.其中通过程序搜索能够得到满足特定约束的代码,进而支持智能化的程序构造.这其中还要用到代码资源库以及代码相关知识作为支撑.目前,由机器自动进行代码构造还处于起步阶段,构造的代码还只能够解决一些非常简单的问题,所以短时间内机器还难以取代程序员.但随着机器学习、人工智能等技术在程序构造领域的进一步深入研究和应用,在不久的将来,这一目标极有可能取得突破性进展.

目前,该方向的研究已经得到学术界和工业界的广泛关注.美国国防高级计划署 DARPA 在 2014 年发起了 MUSE(Mining and Understanding Software Enclaves)项目[7],其目的是把大规模程序代码作为数据对待,通过对代码语料库中的知识进行学习和挖掘,以图模型等理论为基础对知识进行建模和表示,再通过对知识模型进行推理来支持软件开发中的各种任务.MUSE 项目已经取得了一些成果,例如莱斯大学牵头的 PLINY 项目研究如何让程序编码变得和上网搜索一样简单[8],该工作宗旨是希望能通过代码自动生成和自动修正功能,形成针对程序开发者的编码自动完成工具.此外,企业界一些著名的 IT 公司,如 Google、Microsoft、Facebook、百度等都在该方向投入了大量的资源开展研究工作,并且已经取得了一定的研究成果.该方向研究已经成为当前软件工程领域最重要的前沿方向之一,软件工程、程序语言的国际顶级会议(如 ICSE、FSE、POPL、PLDI、ASE 等)和专门的研讨会(如 MSR 等)都将该方向作为重要的研究议题,吸引了越来越多的学者投入相关研究.

2 主要研究问题和文献检索方式

2.1 主要研究问题

智能化的程序搜索与构造和传统程序合成或代码自动生成方法的主要不同在于,它希望利用互联网中已有的大量代码知识和群体智慧结晶,将机器学习、数据挖掘与传统程序语义理论相结合,在人工参与辅助或利用少量先验知识的情况下,智能化地辅助程序开发.尽管目前还有一定局限性,智能化构造的程序范围和规约还不足,并且很多时候需要以人机协同的方式进行,但该研究方向对于传统的软件开发方法将是一个显著的变革,预期能够极大地提高软件开发效率,同时也能够提高程序的最终质量.从目前看,这些方法的具体应用包括程序搜索(也称为代码搜索)、程序合成、代码推荐与补全、缺陷检测、代码风格改善、程序自动修复等.具体来讲,在基于海量代码知识的背景下:

(1) **程序搜索**:指根据自然语言描述、程序结构描述、测试用例等查询规约,通过对互联网代码资源库进行搜索,得到满足特定约束的程序代码;
(2) **程序合成**:指根据<输入-输出>数据对、自然语言或领域特定语言描述的规约自动生成相应程序代码;
(3) **代码推荐与补全**:指根据程序上下文为用户推荐候选代码,或根据已有的部分程序补全生成完整程序代码;
(4) **缺陷检测**:本文关注的是数据驱动的,基于机器学习方法的智能化缺陷检测,与传统缺陷检测方法不同,主要指根据从大量代码中自动学习出的质量特征来发现被分析程序中的缺陷;
(5) **代码风格改善**:指在不改变程序外部行为的前提下,根据互联网中具有良好规范的代码为程序代码的编码风格等给出改善建议;
(6) **程序自动修复**:指针对给定的程序问题,自动生成程序补丁,能够修复程序中的错误.

为了对已有大量代码资源和知识进行利用,智能化的程序构造需要把传统软件理论中的程序语义理论和机器学习的知识发现与挖掘理论有效结合.已有的研究工作表明,程序代码中包含着类似自然语言中的统计性规律[54][68],充分利用这些规律能够为程序构造和质量保证提供显著帮助,因此可以把统计语义的相关技术应用到智能化的程序构造中.

本文所关注的智能化程序搜索与构造技术如图 2 所示.我们根据这些技术的主要目标将其分为三类,包括程序搜索技术、程序构造技术以及构造辅助技术.其中程序构造技术包括程序合成、代码推荐与补全以及代码风格改善.由于代码风格改善的目的是在不改变程序外部行为的前提下改善代码的内部结构,属于代码重构的

范畴,因此也将其归入程序构造技术中.构造辅助技术主要有缺陷检测以及程序自动修复等.程序搜索技术通常作为程序构造技术的基础,通过搜索得到满足特定约束的代码来支持程序构造技术.构造辅助技术能够为程序构造技术提供质量保证,也能够对构造出的程序进行度量和评估,从而指导程序构造的逐步精化.三者共同构成了智能化程序搜索与构造的技术谱系.

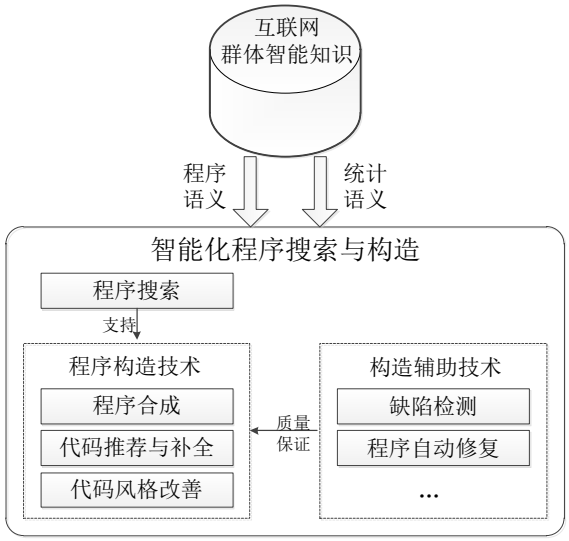

Fig.2 Intelligent search and construction spectrum of program

图 2 智能化程序搜索与构造技术谱系

2.2 文献检索方式

本文参考的文献力图覆盖近几年智能化程序搜索与构造相关的主要研究工作,文献检索的途径是通过 Google 学术搜索、ACM Digital Library、IEEE Xplore Digital Library 以及 Springer Link Online Library 等.内容主要包括软件工程-系统软件-程序设计语言的国际顶级会议 ESEC/FSE、ICSE、PLDI、POPL、ASE 以及专门的研讨会如 MSR 等录用的文章,此外还包括机器学习、数据挖掘相关的会议如 ICML、ICDM 以及 ICLR 等(由于相关工作在此类会议中录用比例较小,故后文图表中未予以统计).本文还对该领域著名研究团队的工作进行了重点跟踪,此外还包括了其他具有历史性或代表性的相关研究工作.

Table 1 Number of scientific articles accepted each year

表 1 历年录用文章结果统计

	ESEC /FSE	ICSE	PLDI	POPL	ASE	总计
2011	0	2	4	0	3	9
2012	3	4	1	0	2	10
2013	2	5	1	0	2	10
2014	5	6	2	0	2	15
2015	5	4	3	1	6	19
2016	7	10	1	4	4	26
2017	8	9	3	1	N/A	21
总计	30	40	15	6	19	110

本文通过对会议 ESEC/FSE、ICSE、PLDI、POPL 与 ASE 中自 2011 年至今录用的文章进行统计,从中筛选出与智能化程序搜索与构造相关的文章数量,统计结果如表 1 所示(其中 ASE 2017 还未召开,暂未统计其文章数).其中包括 ESEC/FSE 论文 30 篇,ICSE 论文 40 篇,PLDI 论文 15 篇,POPL 论文 6 篇,ASE 论文 19 篇.通过对统

计结果进行分析可以看出,ESEC/FSE 与 ICSE 中的文章数量占大部分,约为 63.6%.而且该方向录用的论文数量在随着时间缓步上升,如图 3 所示,尤其是近两年录用的文章居多,约占总体的 42.7%.可见,该方向的研究正在日渐成为一个前沿热点.

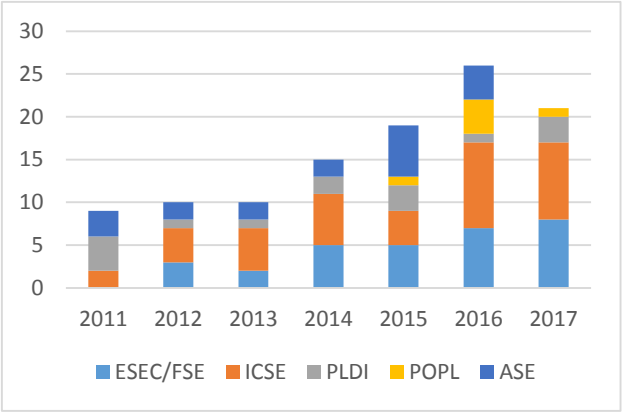

Fig.3 Illustration of literature by year

图 3 文献年代分布

此外,我们对文章所研究的内容进行了统计分析,得到了图 4 的数据.通过分析结果可以看出,已有的研究集中在程序合成、代码推荐和补全、缺陷检测、缺陷预测、代码搜索、代码性质预测等方向.

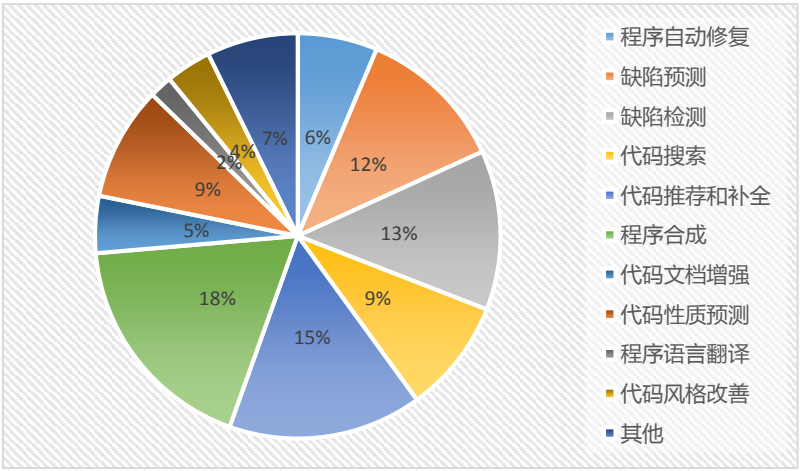

Fig.4 Analysis of the research content of articles

图 4 相关论文研究内容分析

本文后面 3-6 节将分别对代码搜索、程序合成、代码推荐与补全、缺陷检测等方面的最新研究进展进行综述,第 7 节将对代码性质预测、代码风格改善、代码文档增强、程序语言翻译等相关方面的进展进行简要总结.由于程序自动修复是近年来一个热门研究方向,已有大量研究工作并有相关论文进行综述[87][88],因此本文不再展开进行描述.在本文的最后,分析了智能化的程序搜索与构造领域研究面临的主要挑战并给出建议的研究方向.

3 代码搜索

随着大规模开源代码资源库的快速发展,充分利用前人的智慧结晶来辅助软件开发已经成为了当前软件开发人员最常用的做法.在 Janice Singer 等 2010 年做的一项调查中发现,软件开发人员使用最频繁的活动之一

就是代码搜索[9],这已成为增强代码复用、提高软件开发效率的关键环节.代码搜索的一般过程包括搜索规约的选取、搜索方法的选择以及对搜索结果的排序.从搜索规约的角度,通常的代码搜索可以分为基于关键字的搜索、基于程序结构的搜索以及基于测试用例的搜索.基于关键字的搜索需要找到能够精确描述代码特征的关键字,因此通常会得到很多不相关的搜索结果,需要人工进行鉴别.基于程序结构的搜索一般包括方法声明或者程序的逻辑结构等,这种方法需要用户具备一定的领域知识.基于测试用例的搜索用计算机能够理解的方式描述用户想要程序的输入输出行为,然而其瓶颈在于测试用例的编写.在实际中,为了提高搜索的准确性,经常会将三者组合来进行搜索.从代码搜索结果的粒度来看,主要分为代码片段粒度的搜索、代码模块粒度的搜索以及代码文件粒度的搜索.从搜索对象的角度,主要分为对代码库的搜索以及对代码相关文档的搜索.根据选取的搜索规约、搜索结果的粒度以及搜索对象对相关代表性代码搜索工作的分类如表 2 所示.

目前通用的互联网搜索引擎,如 Google、Bing、百度等也能用来搜索代码,但由于这些通用搜索引擎主要针对全文搜索,没有考虑代码的特征和语义,也没有专门的代码资源库,因此在代码搜索方面存在很多问题并且效率很低.对此,这些主流搜索引擎也针对代码搜索进行了专门研究,例如 Google 公司曾开发过代码搜索引擎 Google Code Search[10,11](目前 Google 公司已经关闭了该服务).此外微软也在进行着代码搜索相关的研究工作,如 Bing Code Search[12]等.目前商用的代码搜索引擎主要有 Krugle[13]、GrepCode[14]等,它们大多采用的是基于关键字匹配的搜索策略,搜索的准确性比较低,对于特定的应用很难得到正确的结果.目前常用的基础代码库主要是一些开源社区以及开源代码托管网站的代码资源库,例如 Bitbucket[15]、Github[1]、CodePlex[16]、GitLab[17]、SourceForge[18]等.

Table 2 Categories of Code Search

表 2 代码搜索分类

项目	类别	代表工作
搜索规约	关键字	Krugle、GrepCode、SNIFF[19]、CodeHow[20]、Sourcerer[22]、S6[23]、Portfolio[31]
	程序结构	S6[23]、ParseWeb[30]
	测试用例	S6[23]
搜索粒度	代码片段	Krugle、GrepCode、SNIFF[19]、CodeHow[20]、S6[23]、ParseWeb[30]
	代码模块	Krugle、GrepCode、SNIFF[19]、Portfolio[31]
	代码文件	Krugle、GrepCode、Sourcerer[22]
搜索对象	代码库	Krugle、SNIFF[19]、CodeHow[20]、Sourcerer[22]、S6[23]、ParseWeb[30]、Portfolio[31]
	相关文档	Krugle、SNIFF[19]、CodeHow[20]

为了提高代码搜索的准确性以及搜索效率,目前有很多相关的研究工作,从它们所采取的技术途径分类,主要有以下几种.

3.1 增强代码资源描述信息

该类方法通过对代码和相关文档等进行分析,通过增强代码的索引或标注信息来提高查询的准确率.SNIFF[19]是针对 Java 语言的代码搜索工具.该方法首先分析代码段中所包含的 API 信息,然后利用对应的 API 文档为代码段添加注释,来增强代码的描述信息.搜索时同时搜索代码文本以及添加的注释信息,能够提高代码搜索的准确性,实验发现,与其他工具相比能够加快 40%软件重用的速度.CodeHow[20]是一种基于查询分析和扩展布尔模型的代码搜索方法.该方法首先通过识别代码特征,对源代码进行解析并建立索引.然后利用查询分析,分析查询语句与 API 之间的相关性,识别出与查询语句相关的 API,然后将二者结合起来进行代码搜索.最后运用扩展布尔模型对查询表达式与代码实体进行相似度评估,从而提高了代码搜索性能,实验利用包含了 26000 个 C#项目的大规模代码库进行测试,对于搜索的第一个结果,CodeHow 能够达到 79.4%的准确率.Venkatesh Vinayakarao 等提出了一种利用实体获取技术来提高代码搜索准确率的方法[21].该方法首先利用词

性标注以及模式匹配从 Stack Overflow 的开发者讨论中获取命名实体对应的自然语言术语,并抽取得到相关的句法形式,然后构造出一个实体的知识库.然后利用该知识库对源代码中的每行代码利用相对应的自然语言术语进行注释,来增强代码资源的描述信息.实验表明该方法能够提高 29%的搜索速度.

3.2 提高查询规约描述能力

该类方法通过增强用户的查询规约来提高全面、准确刻画用户需求的能力.Sourcerer[22]是一个基于 Apache Lucene 的大规模源代码搜索引擎,它是集开源软件的爬取、编译、识别以及存储为一体的代码搜索架构,实现了针对代码特征的搜索策略,支持基于关键字以及代码结构的查询搜索,并且易于扩展.S6[23]是一个基于程序语义的代码搜索引擎,它的目标是让用户能够尽可能精确地利用包括关键字、类/方法的声明、测试用例以及一系列其他的约束等来描述所需的代码.它首先通过关键字搜索,得到初始的搜索结果,然后通过与用户提供的查询规约进行比对,对初始的搜索结果进行一系列的扩展和变换,最终得到满足用户需求的代码.此外,还有利用查询扩展来增强用户规约的工作.Meili Lu 等提出了利用 WordNet[24]来进行查询扩展的方法[25],该方法首先利用 WordNet 中的同义词对查询语句进行扩展,然后从源代码中抽取得到自然语言短语,同样利用 WordNet 进行扩展,然后对二者进行匹配查找,以提高代码搜索的效果.Otávio A. L. Lemos 等提出了基于辞典来自动进行查询扩展从而提高代码搜索效果的方法[26].该方法采用了三种辞典来进行查询扩展,分别是 WordNet、用于类型扩展的类型辞典以及只包含软件相关词汇的辞典.Liming Nie 等提出了基于群体知识的查询扩展方法[27],其所说的群体知识就是指 Stack Overflow 中的问答信息.该方法对于给定的自由格式的查询规约,采用了伪相关反馈(Pseudo Relevance Feedback)的方法从 Stack Overflow 中相关问答信息中获取软件相关的词汇,然后对原始的查询规约进行扩展.Mohammad Masudur Rahman 等提出了在集成开发环境中利用众包化的知识来进行代码搜索的方法[28],该方法首先利用用户提供的自然语言查询规约作为输入,然后利用 Stack Overflow 中的问答知识将自然语言查询翻译为相关的 API,然后利用得到的 API 到 GitHub 中搜索得到相应的代码示例返回给用户.

3.3 通过语义分析提高匹配精确度

该类方法通过在搜索中加入程序分析中的一些理论,以提高搜索规约和被搜索程序之间的匹配精确性.典型的一种方式就是 Kathryn T. Stolee 等将代码搜索问题归纳为约束求解问题,提出利用<输入-输出>数据对和约束求解的方法来进行基于语义的代码搜索方法[29].该方法首先对代码库中的代码段进行编码处理,另外需要用户提供<输入-输出>数据对来进行搜索,通过符号化分析,将代码以及<输入-输出>数据对转换成约束条件,最后使用约束求解器 SMT 来从代码库中找到满足条件的程序或者程序片段.该方法的局限性在于不易于扩展,目前只支持 Java 语言的子集.

此外,还有一些通过代码搜索提供给用户相应的方法或 API 调用序列的研究工作.PARSEWeb[30]能够根据用户的查询自动给出方法调用序列.该方法采用"源对象类型→目标对象类型"格式的查询作为输入,首先通过代码搜索引擎获得相关的代码样本,然后通过对其进行静态分析抽取得到相应的调用序列.Portfolio[31]是一个能够在代码搜索中自动获取相关函数及其用法的工具.它结合了自然语言处理技术和 PageRank 的标注技术,能够为用户返回排序后的一组相关函数.Shaowei Wang 等还在 Portfolio 的基础上,将用户对于搜索结果的反馈信息用于返回结果的排序上,能够进一步提高搜索结果的准确性[32].

代码搜索技术经常作为其他软件开发技术的重要支撑技术,比如应用于程序合成技术、代码推荐和补全技术的前端,此外还经常用于程序自动修复中补丁的搜索.代码搜索技术近年来得到了很快的发展,通过多种技术的应用,代码搜索的效率和准确性都有了很大的提升,但是距离程序开发人员心中"拿来就能用"的目标还有一定的差距.

4 程序合成

程序合成是一种针对用户意图,构造出相关程序的软件开发活动.与程序员具体编程完成一项任务不同,它关注让计算机理解用户需要什么,进而去自动化编程,其最终目标是让机器能够自动编写代码,以在一定程度上

取代程序员的工作.Sumit Gulwani 将程序合成的过程分为三个维度[33,34]:用户意图的表述,程序搜索空间的刻画,以及搜索技术的设计.这里所谓的搜索与之前提到的代码搜索不同,并不是指对大量已有代码进行搜索,而是对可能构成程序的解空间进行搜索.目前,程序合成方面的研究主要可以分为以下几种途径.

4.1 归纳程序合成(Inductive Program Synthesis, IPS)

IPS问题的目标是给定一组<输入-输出>数据对,如何产生一段代码能够自动将给定的输入转换为给定的输出.这主要包括两个过程,一是代码空间的搜索,二个是代码结果的排序.Sumit Gulwani提出利用<输入-输出>数据对来进行归纳编程的程序合成方法[35].为提高搜索算法效率,该方法采用了两个策略:一是将搜索空间限制到领域特定语言(Domain Specific Language, DSL)上;二是采用了“分而治之”的思想,将大的程序合成问题分解为小规模的子问题.该技术已经在微软产品Microsoft Excel 2013的快速填充(“Flash Fill”)功能中得到了使用,并取得了很好的效果[36].图5表示的就是Excel 2013的快速填充功能,用户提供若干从Email地址中获取用户名的例子,它就能够从例子中学习得到程序来自动从后续的Email地址中抽取得到用户名.另外,他们还提出了利用自然语言描述的规约来进行程序合成的方法[37].该方法通过对包含<自然语言-领域特定语言>数据对的训练集进行学习,能够将自然语言翻译为领域特定语言,然后再执行程序合成工作.此外,他们还提出了将上述两种方法结合起来进行组合程序合成的方法[38].可以利用自然语言和<输入-输出>数据对二者组合来描述相对复杂的规约,以生成功能相对复杂的程序.

	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	michael neipper
8	Robert.Zare@northwindtraders.com	robert zare
9	Laura.Giussani@adventure-works.com	laura giussani
10	Anne.HL@northwindtraders.com	anne hl

Fig.5 Flash Fill in Microsoft Excel 2013[36]

图5 Microsoft Excel 2013的快速填充(“Flash Fill”)功能[36]

Aditya Desai 等提出了类似的利用自然语言进行程序合成的方法[39],该方法利用<自然语言-领域特定语言>数据对的训练集进行训练,然后能够自动生成用领域特定语言描述的程序.通过对超过 1200 个自然语言描述的规约进行实验,合成得到的程序在排序 Top-1 和 Top-3 的准确率分别能够达到 80%和 90%.Mehdi Manshadi 等提出了将利用<输入-输出>数据对编程与利用自然语言编程两种方法结合在一起,以提高程序合成效果的方法[40].该方法首先利用<输入-输出>数据对的示例得到一个所有满足条件的代码空间,然后再从中筛选出最能满足自然语言规约的代码.此外,他们还提出了利用群体智慧来进行自然语言编程的程序合成方法[41].该方法的核心思想就是对于一个自然语言描述的编程任务,让其他非专业人员来提供大量的<输入-输出>数据对,然后采用<输入-输出>数据对的程序合成方法生成相应程序代码,能够更加全面多角度地刻画程序的行为,从而生成更精确的程序.此外,John Feser 等提出了一种不同的、利用<输入-输出>数据对来进行程序合成的方法[42].该方法首先根据<输入-输出>数据对生成对目标程序结构的假设,然后对每个假设,为缺失的子表达式演绎生成新的<输入-输出>数据对,最后再对每个假设进行程序合成工作.Juan Zhai 等提出了利用 Java 文档中的自然语言表述自动为 API 生成相应代码的工作[43],生成的代码能够与原有代码实现同样功能,并且实现简单易于分析,他们成功对 14 个广泛应用的 Java 类中的 326 个函数进行了建模,取得了很好的效果.

4.2 基于搜索的程序合成

该类方法通常与代码搜索结合在一起,首先通过搜索得到相关的代码片段或者 API 调用序列,然后对其进行一系列的变换和处理,以满足用户的需求.Swim[44]是一个能够根据用户提供的规约自动生成代码片段的工具.该工具根据自然语言描述的查询规约得到与之相对应的 API 名称,然后利用结构化调用序列的形式挖掘得到

API 的使用模式,并根据它生成代码片段.Hunter[45]是针对 Java 语言的程序合成工具.该方法需要提供查询语句以及程序需要通过的测试用例,Hunter 利用代码搜索得到搜索结果,通过计算所得结果与目标结果类型之间的距离值,对结果进行重排序.然后进行接口匹配并合成生成相应的包装代码,从而得到候选结果,最后利用测试用例对候选结果进行筛选.将 Hunter 与集成开发环境 Eclipse 集成在一起,能够极大地提高程序开发的效率.Yu Feng 等提出了对于复杂 API 进行基于构件的程序合成方法[46].该方法利用 Petri-Net 来对 API 中方法间的关系进行建模.对于给定的方法声明,根据 Petri-Net 模型进行可达性分析来确定进行合成的方法调用序列,合成得到的程序还需要通过类型检查以及测试用例.此外,他们还提出了一种示例驱动、基于构件的程序合成方法,用来处理表格的合并以及转换任务[47],该方法将基于构件的程序合成与基于 SMT 的约束求解以及局部评估相结合来缩小搜索空间.

4.3 基于深度学习的自动编程

随着深度学习技术的发展,近年来出现了基于神经网络的程序合成方法.神经网络编程目前主流技术路线分为两类[48].第一类是所谓的“黑盒派”,之所以叫做黑盒,就是不显式地给出程序代码,而是通过从训练数据中学习得到转换的规则,规则以神经网络参数的形式表示.“黑盒派”的工作只关注如何完成某项任务,不利于分析系统中存在的问题.另一类就是“代码生成派”,通过显式地给出程序代码,能够让机器像人类程序员一样,将解决问题的过程用代码的形式表示出来.这里我们关注的是“代码生成派”的工作.

“代码生成派”的神经网络自动编程主要分为两个阶段[48]:模型训练阶段和模型应用阶段.在模型训练阶段,需要使用不同种类编程任务的训练数据来对模型进行训练,训练数据包括任务的输入输出数据和任务所对应的代码片段,深度学习自动编程系统就可以利用随机梯度下降算法,通过训练获得自动编程模型.在模型应用阶段,只需为模型提供新任务的若干输入输出数据,机器能够根据训练阶段学习到的模型预测出各种操作原语语句出现在代码片段中的概率,可以认为出现概率较高的原语是代码片段中应该包含的语句.之后可以采用宽度优先搜索或者动态线性规划等搜索技术在代码组合空间中寻找能够准确地将给定的所有输入正确地转换为对应输出的代码片段.Facebook 的工作就是采用了这种方法[49],在神经网络模型的选择上,他们选取的是层级生成式 CNN 模型(Hierarchical Generative Convolutional Neural Networks,简称 HGCNN 模型),训练用到的程序是随机生成的,对于每个程序随机产生输入,然后运行程序得到相对应的输出,因此该方法属于无监督学习.此外还有 Emilio Parisotto 等提出的基于神经网络的程序合成方法[50],他们选取的是 R3(Recursive-Reverse-Recursive)神经网络模型.DeepCoder 是另外一个采取该方法的深度学习自动编码系统[51],该方法利用了神经网络来提高 IPS 问题的能力,选取的实验对象是编程竞赛的程序.还有根据给定的规约利用深度学习生成相应 API 调用序列的方法.Xiaodong Gu 等提出的一种根据自然语言查询规约自动生成 API 使用序列的方法[52].该方法通过对查询中字的序列以及相对应的 API 序列进行学习,利用了 RNN Encoder-Decoder 的神经网络模型进行建模,将字的序列编码成固定长度的上下文向量,然后基于它生成 API 的使用序列.

除了以上几种程序合成方法外,还有利用众包来进行程序合成的方法[53],该方法将开发者的复杂程序问题利用众包来获得初始的结果,然后将初始结果进行合并以获得更加精确的结果.这种方法能够充分利用互联网中群体的智慧来解决复杂的软件问题.

近些年程序合成技术得到了很大的发展,但同时也要看到当前的程序合成技术还局限于生成规模较小、功能单一的程序,对于规模较大、功能复杂的程序,当前的程序合成技术还能力有限;此外,大部分技术的通用性还不够,对于新的任务需要重新进行计算或者训练;另外,对于合成得到的代码,很难在逻辑上验证其正确性.对于通过给定<输入-输出>数据对得到的程序,由于无法覆盖到程序运行的所有可能情况,因此正确性很难验证.不过,虽然面临着诸多困难,但随着人工智能、机器学习等技术的快速发展,相信在不久的将来,让机器人取代人类程序员部分工作的想法就会实现.

5 代码推荐与补全

代码推荐是指用户在编写代码的时候,根据上下文自动为用户推荐可用的代码,这种方法在各种程序代码

的集成开发环境中会经常见到.代码补全是指将不完整的代码片段补全成为完整的代码.

代码推荐与补全中常用的方法就是利用自然语言处理技术,将代码视为文本,融合机器学习或者深度学习的算法,利用统计的语言模型对代码进行建模并对语句的概率进行预测,对当前代码序列的后续语句、函数进行推荐或者对缺失的代码进行补全.N-gram 模型是一种最常见的统计语言模型,它假设一个序列$S = w_1 w_2 \ldots w_m$中每个字的概率依赖于它前面 n-1 个字,因此有如下公式:

$$\Pr(s) = \prod_{i=1}^{m} \Pr(w_i | w_{i-n+1} \cdot \ldots \cdot w_{i-1})$$

Abram Hindle 等人[54]在 2012 年就提出,程序代码有着与自然语言中类似的统计性规律和知识.因此,可以利用代码中的这种规律性,充分利用自然语言处理的各种技术来有效地支持代码补全工作.Veselin Raychev 等提出了利用统计的语言模型来进行代码补全的方法[55],对于给定的不完整代码,能够利用最有可能的方法调用序列将代码补充完整.该方法首先利用静态分析技术,从大规模代码库中抽取得到方法调用序列,然后将其标注为统计的 n-gram 语言模型,通过寻找排序最高的语句进行代码补全.该方法通过机器学习,能够将代码补全问题归纳为预测语句概率的自然语言处理问题.实验表明,90%的正确补全结果出现在排序前三的候选答案中.

统计的语言模型能够成功地利用源代码中全局的统计性规律,但是没有能够利用源代码的另一个特性:代码的局部特征.Zhaopeng Tu 等人[56]提出了可以在统计语言模型的基础上利用程序代码的局部性特征,建立一个新的统计模型——带“Cache”的统计语言模型,在 n-gram 语言模型的基础上额外增加一个“Cache”部件来挖掘代码的局部性规律.实验表明,增加的“Cache”能够显著地提高 n-gram 语言模型的性能.基于此方法实现了工具 CACHECA,通过与 Eclipse 集成,能够提高程序开发人员的效率.Sebastian Proksch 等提出了利用贝叶斯网络进行代码补全的方法[57].该方法抽取并使用了更多的代码上下文信息来提高补全代码预测的质量,并使用基于模式的贝叶斯网络(Pattern-Based Bayesian Networks)中的聚类方法来解决模型规模增大的问题.Marcel Bruch 等提出了基于示例的代码补全系统[58].该方法利用从示例代码库中挖掘得到的信息进行代码补全,通过利用上下文敏感的推荐以及相关方法调用的推荐来增强已有代码补全工具的功能,同时与集成开发环境集成在一起.

Tung Thanh Nguyen 等[59]在 n-gram 模型的基础上,构造了语义的 n-gram 模型,同时结合了主题模型,能够获取更多的程序信息,用于代码的推荐,与已有的方法相比,能够提高 18%到 68%的准确率.由于程序代码具有良定义的语法和语义,而这些语法和语义多是通过树和图的数据结构表示的,因此他们又提出了基于图的统计语言模型 GraLan[60]用于 API 推荐,实验在 75%的例子中,能够在 5 个候选答案中得到正确结果.Anh Tuan Nguyen 等后面又提出了利用细粒度的代码修改信息中包含的规律性来进行代码推荐的方法[61],该方法的核心思想在于细粒度的代码修改信息也是具有一定的规律性的,类似的修改是为了增加类似的功能,该方法通过从细粒度的代码修改信息以及修改位置的上下文信息进行统计学习,最终实验有 59%的记录在一个候选答案中得到正确的 API,有 77%的几率在五个候选答案中得到正确的 API,与已有的代码推荐方法相比获得了很大的提升.Pavol Bielik 等人提出了一种带概率的高阶文法(PHOG)定义[62],在概率上下文无关文法的基础上通过对产生式规则加入了上下文条件,能够获取更丰富的代码上下文信息,提高代码推荐的准确率.

代码推荐与补全中通常会应用到代码搜索的相关技术.Reid Holmes 等提出的代码示例的推荐工具 Strathcona[63],该工具可以找到与 API 使用方法相关的代码片段或者代码示例来辅助开发人员的工作.该工具首先根据用户环境的上下文结构自动产生查询,利用它在示例代码库中进行搜索并与已有项目的语料库进行比较,返回结构最相近的示例代码供用户筛选使用.Hongyu Zhang 等提出了能够为用户推荐示例代码的工具 Bing Developer Assistant[64].该工具通过从开源软件库(如 GitHub)以及编程问答网站(如 Stack Overflow)中挖掘,能够得到实现一个 API 或者回答一个代码搜索任务的代码示例.该工具目前已经与开发环境 Microsoft Visual Studio 进行了集成,取得了很好的效果.Eran Yahav 等开发的代码推荐工具 Codota[90],该工具能够减少用户搜索相关代码的工作,根据用户在集成开发环境中正在编写的代码来预测下一步的工作,然后自动为用户推荐相关的代码.

6 缺陷检测

缺陷检测是保证软件质量的关键环节,一直是软件工程领域一个重要的研究方向.传统的方法大多是基于程序分析来完成的,根据已有代码中发现的缺陷,将其总结为缺陷模式,然后提出相应的检测算法.程序分析一般可以分为静态分析与动态分析两类[65].从本质上说,软件分析的能力是有限的,对于任何一个特定的软件,希望获得关于它的完备描述通常是不现实的[66],特别是对于自动分析而言,许多问题是不可判定的.

本文所关注的缺陷检测主要是数据驱动的、基于机器学习方法的智能化缺陷检测.传统的缺陷检测方法关注于程序本身,利用程序分析等方法对程序中可能存在的缺陷进行查找.智能化的缺陷检测不止关注于程序本身,利用代码资源库以及代码相关的知识,通过引入机器学习等统计分析的方法来从代码资源库中对缺陷的特征知识进行挖掘从而进行缺陷的分析与查找.与传统的缺陷检测方法相比,能够显著降低对缺陷语义知识的依赖.根据有限的先验知识来进行软件缺陷检测,可以为传统的检测方法提供补充.Foyzur Rahman 等将静态的缺陷检测方法与基于统计预测的缺陷检测方法进行了比较并发现,在一些情况下它们的能力相当,并且在某些设定条件下,充分利用统计的缺陷检测信息能够增强静态缺陷检测工具的能力[67].因此,将大规模代码资源库和互联网群体智慧的能力与传统的代码分析方法相结合,能够有效地提高缺陷检测的效果.

目前,该方向的研究主要技术途径有以下几种.

(1) 利用自然语言处理方法,对程序中违背统计规律的异常位置进行缺陷检测

Baishakhi Ray 等人研究了代码的自然性,认为大规模代码资源库中的代码与自然语言一样,通常是高度重复化或具有可预测性的,尤其是那些被广泛使用的高质量的软件代码.基于这个思想,他们发现那些奇怪的、不太经常出现的代码更有可能是缺陷代码[68].他们通过对 10 个不同的 Java 项目的缺陷修复记录进行统计分析,计算了包含缺陷代码的熵值以及修复后代码的熵值,得出了包含缺陷代码的熵值要比修复后代码的熵值高的结论.利用这个结论,能够用来指导缺陷检测工作.Song Wang 等提出了利用统计的 n-gram 语言模型来进行程序缺陷检测的方法 Bugram[69].该方法通过对方法调用序列进行学习,然后利用得到的概率分布来检测缺陷.实验通过对 16 个开源项目的最新版本进行实验,能够找到基于规则的方法无法找到的缺陷,并且能够提高程序缺陷检测的准确率.

(2) 利用开源社区和问答网站中的知识或者程序的演化历史信息来进行缺陷检测

Fuxiang Chen 等提出利用问答网站中的信息来帮助缺陷检测的方法[70],通过对编程问答网站 Stack Overflow 中重复出现的缺陷信息进行挖掘,并以此为基础对真实的项目进行缺陷检测,得到了显著的效果.目前的编译器普遍存在着错误报告位置不准确的问题, Joshua Charles Campbell 等提出利用语言模型来提高编译器错误报告性能的方法[71].该方法利用 n-gram 语言模型来进行基于统计概率的语法错误位置预测,通过对程序的历史版本进行学习,再对新版本程序中的缺陷进行检测并报告,能够提高编译器语法错误定位报告的性能.Quinn Hanam 等提出了通过从代码资源库中挖掘得到缺陷模式的方法[72],该方法通过从 134 个服务器端的 JavaScript 项目的 10.5 万的 commit 信息中挖掘得到了常见的缺陷模式,能够用来提高相关工具的缺陷检测能力.

(3) 利用机器学习和数据挖掘算法来进行缺陷检测

Zhenmin Li 等提出了 PR-Miner[73]的方法,利用数据挖掘中的频繁项集挖掘技术,在少量人工参与并且不需要软件先验知识的情况下,就能够从大规模软件代码中抽取得到隐含的程序规则.可以利用得到的程序规则对程序进行自动验证,程序中违背规则的位置存在缺陷的可能性很高.Bin Liang 等提出了一种名为 AntMiner 的缺陷挖掘方法[74],能够通过切片技术尽量消除噪声代码来提高缺陷挖掘的精确度,该方法从 Linux-2.6.39 中检测出 52 个真实的缺陷,其中 24 个被内核开发人员确认为真实的缺陷,其中有 2 个已经存在了 8 年以上.Vijayaraghavan Murali 等提出了一种贝叶斯的方法来进行缺陷检测,该方法通过从大规模代码语料库中学习得到规约,进而利用规约对程序进行分析来检测程序中的缺陷[75].实验通过对 Android 应用中 API 的使用进行检测,与现有的方法相比能够检测出不易发现的错误并且具有较高的精确度.

7 其他方面的研究

互联网中存在的大量代码知识和群体智慧在用于程序构造时,除了以上几个方面的研究工作之外,还包括一些其他方面的研究,例如代码性质预测、代码风格改善、代码文档增强、程序语言翻译和提高开发环境对程序员的帮助能力等等.

7.1 代码性质预测

代码性质预测指利用从大量代码中学习到的知识,对被开发代码中的标识符名称、变量类型以及函数的前置条件等性质进行预测,目前多是利用机器学习中的概率模型来进行预测.Veselin Raychev 等人提出了利用大规模代码资源库对程序性质进行预测的方法[76],其关键思想在于将程序性质的预测问题形式化为机器学习中的结构预测问题,该方法从已有的训练数据集中学习到一个概率的条件随机场模型.对于输入的程序,首先构造出数据依赖网络,得到已知性质与未知性质之间的依赖关系,然后利用条件随机场模型对性质进行预测,得到包含预测性质的输出程序.在此基础上实现了相应工具 JSNice,可以对 JavaScript 代码中标识符名称给出预测,也可以预测变量的类型并给出相应的注释.Hoan Anh Nguyen 等提出了从已有的大规模开源代码资源库中挖掘得到 API 前置条件的方法[77].其核心思想是 API 前置条件在 API 被大量使用的代码库中会频繁出现,而项目内其他条件的出现频率会相对较低.通过对 1.2 亿行来自 SourceForge 和 Apache 项目的源码挖掘 JDK 方法的前置条件,并与 JML 中人工书写的前置条件作比较,表明该方法具有较高的准确率.Zhaogui Xu 等提出了利用统计方法对 Python 代码中的变量类型进行预测的方法[78],该方法的主要思想是 Python 代码中隐含了很多变量类型的隐含信息,通过利用这些信息在代码中的传播、汇聚,最后利用概率推理的方法就可以对变量的类型进行预测.实验通过对 18 个真实的 Python 项目进行预测,结果表明该方法效果优于基于抽象解释的类型预测方法.

7.2 代码风格改善

代码风格改善指利用从规范良好的代码中挖掘得到的相关知识来提升代码风格一致性以及规范性.Miltiadis Allamanis 等提出了从已有具有良好规范的程序项目中挖掘出代码惯用语的方法[79].代码惯用语是指跨项目中重复出现并且语义功能单一的语法片段,该方法挖掘出的惯用语可以用于编写代码时进行代码片段的推荐.他们还提出了一种通过从已有的代码库中进行学习,获得一致的代码风格,进而对新的代码给出修改建议来增强代码风格的一致性的方法[80].该方法采用了自然语言处理技术对源代码进行处理,可以对标识符名称和格式的一致性给出建议.据此方法实现的工具 Naturalize,能够对 Java 代码风格一致性给出建议.此外,他们还提出了对源代码进行概率的神经网络语言模型建模,从而自动对程序中的类或方法给出相应功能性描述命名的方法[81],能够提高代码的可读性和程序的可维护性.

7.3 代码文档增强

如何基于已有知识增强代码文档和开发环境的帮助能力,对提高开发效率有重要价值.例如,Stack Overflow 中包含了很多程序员的编程问答信息,能够给开发人员提供很多的帮助,如何利用这些知识得到广泛关注.Christoph Treude 等提出了利用 Stack Overflow 中的信息,自动增强 API 文档的方法[82].该方法基于有监督的机器学习方法,从 Stack Overflow 中自动获得与特定 API 类型相关的句子,将其加入到 API 文档中以丰富 API 文档的内容,为后续的程序开发和研究工作提供便利.Mani Senthil 等提出了利用了无监督的机器学习为缺陷报告自动生成摘要的方法[83],该方法能够降低缺陷报告中的噪声干扰,取得了比有监督学习更高的准确率.

7.4 程序语言翻译

机器学习在自然语言翻译领域近年取得了显著的效果,借鉴相关方法利用统计方法促进不同程序语言之间的转换也被关注.Svetoslav Karaivanov 等提出了将基于短语的统计机器翻译方法应用于程序语言之间的翻译[84],为了能够适用基于短语的统计机器翻译方法,首先将上下文无关文法优化为一个前缀语法,然后在基于短语的翻译方法中同时考虑了语言的语法.此外,Anh Tuan Nguyen 也提出将基于短语的统计机器翻译方法用在基于词法的程序语言翻译上[85],实验表明该方法具有较高的词法翻译准确度.Yusuke Oda 等研究了将统计的机器翻

译方法用于伪代码的自动生成上[86],他们利用<源代码-伪代码>对进行训练,实验中将 Python 代码分别生成了英文和日文的伪代码,获得了很高的精确度.

8 面临的主要挑战和研究方向

总体来看,目前智能化的程序搜索与构造技术还处于起步阶段,前面介绍的相关工作表明利用互联网中大量代码知识、现有机器学习算法和基于搜索的技术来支持智能化的软件开发是可行的,能一定程度上帮助程序员高效高质量地生成代码,未来有可能对软件开发的方式和软件工程环境产生颠覆性的影响.同时我们也应该看到,目前的研究工作难以满足实际软件开发的需要,面临着以下的挑战:

(1) 传统程序构造方法中所需的知识多是通过建立程序语义的方式获得的,由于该方式所固有的局限性,难以有效处理数以亿行计的海量代码和相关信息,因此考虑采用机器学习和数据驱动等方式帮助代码知识的获取.但对于人的高层需求描述和具体程序语义之间的映射目前还存在鸿沟,在进行智能化搜索和构造时,需要从海量代码中获取哪些隐含知识,以及如何对这些知识进行提取和表示,是结合机器学习和统计分析方法的智能化程序搜索与构造面临的一项挑战.

(2) 当通过机器学习和统计分析等手段将从海量代码中获取的知识用于程序搜索和构造时,为了达到正确性或精确性要求,显然需要建立统计语义和传统程序语义之间的联系.但目前对这两种不同维度语义之间的关系还缺乏研究,例如对代码进行深度学习得到的神经网络模型或统计学习得到的概率模型,与实际正确的程序之间的关系还难以清晰描述.因此如何建立二者之间相关性与因果性的联系,以及由于二者在表示上的异构性,如何对二者进行转换与融合也成为一项挑战.

(3) 当前复杂软件的开发实际也是一定数量人员的群体智慧结晶,从互联网海量代码中获取的知识也是大量不同人员的智慧成果,但这两种智能的表示和发挥作用的方式存在较大区别.但实际软件开发中如果要实现智能化的程序构造,需要把海量代码中获取的知识与开发团队人员的群体智慧有效融合,形成人机共融、高效协同、持续学习的过程,并建立系统的技术体系来支持软件开发的全过程,这也成为面临的一项重大挑战.

(4) 从前面对各种方法的综述可以看到,目前已有的智能化程序搜索和构造方法在应用目标、原理、算法、处理对象、数据来源、实验方式、评价方法等方面存在很大差异,难以对不同方法的有效性进行比较.其主要原因在于目前还缺乏被广泛认可的智能化程序构造评价标准和通用的测试集.如何针对智能化程序搜索与构造建立具有普适性的评价度量体系和测试集,以及在度量中如何将精确判据与统计判据相结合也是面临的一项挑战.

针对当前智能化的程序搜索与构造技术面临的上述挑战,以下几个方面可能将会成为进一步深入研究的重要方向:

(1) 异源异构代码知识的获取与建模

- 研究互联网上大量代码、文档、开发社区、问答网站包含的众多信息中,哪些是对程序搜索与构造具有显著促进作用的知识,以及这些知识如何获取和表示;
- 针对大量的部分程序代码(无法编译通过),需要研究有效的部分程序编译与理解技术,以及相应的上下文分析技术,以有效支持代码知识的获取和进一步的搜索与构造;
- 需要研究如何综合运用自然语言处理技术、贝叶斯网络等概率图模型、以深度学习为代表的机器学习技术、基于程序语义的分析技术,对异源异构代码知识进行处理和融合、建模,并形成有效支持程序搜索与构造的代码知识图谱.

(2) 智能化程序搜索与构造

- 需要研究机器学习、统计分析获取的代码知识语义和支持软件运行的程序语义之间存在的内在联系与映射机理,以有效支持程序搜索与构造的正确性和精确性;
- 需要研究面向程序搜索与构造的查询语言、规约语言或领域特定语言,能够在较抽象层次表达人脑

中形成的开发需求,同时支持高效的程序搜索与构造过程;

- 为了提高各种程序搜索与构造方法的效率和效果,需要针对程序语义的特征研究机器学习、统计分析、代码空间搜索等相关算法,而不仅是把已有算法直接用于程序搜索与构造;
- 为了构造具有一定复杂程度的程序,需要研究将数据驱动的构造(如概率图模型)、基于搜索的构造(如归纳构造、概要编程)与软件开发中已被证明行之有效的分解、抽象与逐步求精等原则有机结合的方法.

(3) 智能化的代码质量保证技术

- 形成的代码知识和智能化算法还可以用来分析程序中存在的不足之处,例如通过代码统计知识来检测和诊断程序缺陷、改善代码风格等,但需要研究智能化的代码质量保证技术的优势和能达到的程度,以及与程序分析等传统代码质量保证技术之间的互补性;
- 对于某些方法即使构造出模型也是一种不含可读代码的黑盒,如深度学习技术,还有一些方法构造出的代码即使通过测试,但对人来说缺乏可读性,如何对这样构造出的程序/模型质量进行分析和改善,需要进行深入的研究.

(4) 程序搜索与构造方法的评价以及协同开发环境

- 智能化的程序搜索与构造希望达到什么最终目标,发展过程中应该分为那几个阶段,每个阶段的标志和能力是什么,这都是需要去深入研究和探讨的内容,使得相关研究工作能具有明确的指向;
- 为了对众多采用不同技术原理的方法进行评价,需要研究建立对智能化的程序搜索和构造方法进行度量和评价的标准,并形成有较高公认度的测试集;
- 智能化开发依然不能忽略人的参与,研究如何将智能化程序搜索与构造方法与软件设计、编码、测试等活动有机结合,与主流软件开发环境相融合,形成高效的人机协同开发过程和支撑环境,对建立未来新的软件开发方法学具有重要意义.

9 结束语

互联网技术的持续发展给软件工程领域带来了深远的影响.随着大量开源软件和开源社区的发展,目前在互联网上已经存在的各类程序代码及其相关资源包含着各种知识,尤其是众多已被广泛使用、高质量的软件代码.正如互联网中的海量数据使得大数据的应用成为了热点,如今互联网中的大量代码也催生了利用大规模代码资源库中已有的知识来解决软件工程问题的新思路.如何利用互联网群体智慧和知识来支持智能化的程序搜索与构造技术已经得到了越来越多科研工作者的重视.

本文通过对该领域近些年国内外的研究工作进行调研分析,从代码搜索、程序合成、代码推荐与补全、缺陷检测以及其他一些方面,对当前智能化的程序搜索与构造领域研究进展进行综述,对这些工作的原理和技术途径进行了梳理.本文还总结了该领域研究过程中面临的一些主要挑战,最后给出了建议的研究方向.

智能化的程序搜索与构造作为一个新兴的研究方向,具有广阔的发展前景,需要更多的科研工作者投入到这一领域中来,以推进该领域研究更好更快地发展.

References:

[1] https://github.com/

[2] Lakhani K R, Garvin D A, Lonstein E. Top coder (A): Developing software through crowdsourcing. Harvard Business School Case 610–032. 2016.

[3] https://stackoverflow.com/

[4] Schmidt D C. Model-driven engineering. COMPUTER-IEEE COMPUTER SOCIETY-, 2006, 39(2): 25.

[5] Manna Z, Waldinger R J. Toward automatic program synthesis. Communications of the ACM, 1971, 14(3): 151-165. [doi:10.1145/362566.362568]

[6] Potvin R, Levenberg J. Why Google stores billions of lines of code in a single repository. Communications of the ACM, 2016, 59(7): 78-87. [doi:10.1145/2854146]

[7] http://www.darpa.mil/Our_Work/I2O/Programs/Mining_and_Understanding_Software_Enclaves_(MUSE).aspx/

[8] http://pliny.rice.edu/index.html/

[9] Singer J, Lethbridge T, Vinson N, et al. An examination of software engineering work practices. CASCON First Decade High Impact Papers. IBM Corp., 2010: 174-188. [doi:10.1145/1925805.1925815]

[10] http://code.google.com/codesearch/

[11] https://swtch.com/~rsc/regexp/regexp4.html/

[12] http://codesnippet.research.microsoft.com/

[13] http://www.krugle.com/

[14] http://grepcode.com/

[15] https://bitbucket.org/

[16] http://www.codeplex.com/

[17] https://about.gitlab.com/

[18] https://sourceforge.net/

[19] Chatterjee S, Juvekar S, Sen K. Sniff: A search engine for java using free-form queries. Int'l Conf. on Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2009: 385-400. [doi:10.1007/978-3-642-00593-0_26]

[20] Lv F, Zhang H, Lou J, et al. CodeHow: Effective Code Search Based on API Understanding and Extended Boolean Model. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. IEEE, 2015: 260-270. [doi:10.1109/ase.2015.42]

[21] Vinayakarao V, Sarma A, Purandare R, et al. Anne: Improving source code search using entity retrieval approach. In: Proc. of the 10th ACM Int'l Conf. on Web Search and Data Mining. ACM, 2017: 211-220. [doi:10.1145/3018661.3018691]

[22] Bajracharya, S., Ngo, T., Linstead, E., Rigor, P., Dou, Y., Baldi, P., & Lopes, C. Sourcerer: A Search Engine for Open Source Code. Conf. on Object-Oriented Programming, Systems, Languages, and Applications. 2006: 681–682. [doi:10.1145/1176617.1176671]

[23] Reiss S P. Semantics-based code search. In: Proc. of the 31st Int'l Conf. on Software Engineering. IEEE, 2009: 243-253. [doi:10.1109/icse.2009.5070525]

[24] Leacock C, Chodorow M. Combining local context and WordNet similarity for word sense identification. WordNet: An electronic lexical database, 1998, 49(2): 265-283.

[25] Lu M, Sun X, Wang S, et al. Query expansion via wordnet for effective code search. In: Proc. of the 22nd Int'l Conf. on Software Analysis, Evolution and Reengineering. IEEE, 2015: 545-549. [doi:10.1109/saner.2015.7081874]

[26] Lemos O A L, de Paula A C, Zanichelli F C, et al. Thesaurus-based automatic query expansion for interface-driven code search. In: Proc. of the 11th Working Conf. on Mining Software Repositories. ACM, 2014: 212-221. [doi:10.1145/2597073.2597087]

[27] Nie L, Jiang H, Ren Z, et al. Query expansion based on crowd knowledge for code search. IEEE Trans. on Services Computing, 2016, 9(5): 771-783. [doi:10.1109/tsc.2016.2560165]

[28] Rahman M M, Roy C K, Lo D. RACK: code search in the IDE using crowdsourced knowledge. In: Proc. of the 39th Int'l Conf. on Software Engineering Companion. IEEE, 2017: 51-54. [doi:10.1109/icse-c.2017.11]

[29] Stolee K T. Finding suitable programs: Semantic search with incomplete and lightweight specifications. In: Proc. of the 34th Int'l Conf. on Software Engineering, 2012: 1571-1574. [doi:10.1109/icse.2012.6227034]

[30] Thummalapenta S, Xie T. Parseweb: a programmer assistant for reusing open source code on the web. In: Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated software engineering. ACM, 2007: 204-213. [doi:10.1145/1321631.1321663]

[31] McMillan C, Grechanik M, Poshyvanyk D, et al. Portfolio: finding relevant functions and their usage. In: Proc. of the 33rd Int'l Conf. on Software Engineering. ACM, 2011: 111-120. [doi:10.1145/1985793.1985809]

[32] Wang S, Lo D, Jiang L. Active code search: incorporating user feedback to improve code search relevance. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated software engineering. ACM, 2014: 677-682. [doi:10.1145/2642937.2642947]

[33] Gulwani S. Dimensions in program synthesis. In: Proc. of the 12th Int'l ACM SIGPLAN Symp. on Principles and practice of declarative programming. ACM, 2010: 13-24. [doi:10.1145/1836089.1836091]

[34] Gulwani, S. Program Synthesis. Software Systems Safety 2014: 43-75.

[35] Gulwani S. Automating string processing in spreadsheets using input-output examples. ACM SIGPLAN Notices. ACM, 2011, 46(1): 317-330. [doi:10.1145/1925844.1926423]

[36] Gulwani S, Esparza J, Grumberg O, et al. Programming by Examples (and its applications in Data Wrangling). Verification and Synthesis of Correct and Secure Systems. IOS Press, 2016.

[37] Desai A, Gulwani S, Hingorani V, et al. Program synthesis using natural language. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM, 2016: 345-356. [doi:10.1145/2884781.2884786]

[38] Raza M, Gulwani S, Milic-Frayling N. Compositional Program Synthesis from Natural Language and Examples. In: Proc. of the 24th Int'l Joint Conf. on Artificial Intelligence. 2015: 792-800.

[39] Desai A, Gulwani S, Hingorani V, et al. Program synthesis using natural language. In: Proc. of the 38th Int'l Conf. on Software Engineering. 2016: 345-356. [doi:10.1145/2884781.2884786]

[40] Manshadi M H, Gildea D, Allen J F. Integrating Programming by Example and Natural Language Programming. In: Proc. of the 27th AAAI Conf. on Artificial Intelligence. 2013.

[41] Manshadi M, Keenan C, Allen J. Using the crowd to do natural language programming. In: Proc. of the 26th AAAI Conf. on Artificial Intelligence, Workshop on Human-Computer Interaction. 2012.

[42] Feser J K, Chaudhuri S, Dillig I. Synthesizing data structure transformations from input-output examples. ACM SIGPLAN Notices. ACM, 2015, 50(6): 229-239. [doi:10.1145/2737924.2737977]

[43] Zhai, J., Huang, J., Ma, S., Zhang, X., Tan, L., Zhao, J., & Qin, F. Automatic Model Generation from Documentation for Java API Functions. In: Proc. of the 38th Int'l Conf. on Software Engineering. IEEE, 2016: 380–391. [doi:10.1145/2884781.2884881]

[44] Raghothaman M, Wei Y, Hamadi Y. SWIM: synthesizing what I mean: code search and idiomatic snippet synthesis. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM, 2016: 357-367. [doi:10.1145/2884781.2884808]

[45] Wang Y, Feng Y, Martins R, et al. Hunter: next-generation code reuse for Java. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2016: 1028-1032. [doi:10.1145/2950290.2983934]

[46] Feng Y, Martins R, Wang Y, et al. Component-based synthesis for complex APIs. In: Proc. of the 44th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. 2017: 599-612. [doi:10.1145/3093333.3009851]

[47] Feng Y, Martins R, Van Geffen J, et al. Component-based synthesis of table consolidation and transformation tasks from examples. In: Proc. of the 38th ACM SIGPLAN Conf. on Programming Language Design and Implementation. ACM, 2017: 422-436. [doi:10.1145/3062341.3062351]

[48] http://blog.csdn.net/malefactor/article/details/72853720/

[49] Gong Q, Tian Y, Zitnick C L. Unsupervised Program Induction with Hierarchical Generative Convolutional Neural Networks. In: Proc. of the 5th Int'l Conf. on Learning Representations. 2017.

[50] Parisotto E, Mohamed A, Singh R, et al. Neuro-Symbolic Program Synthesis. In: Proc. of the 5th Int'l Conf. on Learning Representations. 2017.

[51] Balog M, Gaunt A L, Brockschmidt M, et al. DeepCoder: Learning to Write Programs. In: Proc. of the 5th Int'l Conf. on Learning Representations. 2017.

[52] Gu X, Zhang H, Zhang D, et al. Deep API learning. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2016: 631-642. [doi:10.1145/2950290.2950334]

[53] Cochran, R. A., D'Antoni, L., Livshits, B., Molnar, D., & Veanes, M. Program Boosting: Program Synthesis via Crowd-Sourcing. In: Proc. of the 42nd Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, 2015, 677–688. [doi:10.1145/2676726.2676973]

[54] Hindle A, Barr E T, Su Z, et al. On the naturalness of software. In: Proc. of the 34th Int'l Conf. on Software Engineering. IEEE, 2012: 837-847. [doi:10.1109/icse.2012.6227135]

[55] Raychev V, Vechev M, Yahav E. Code completion with statistical language models. ACM SIGPLAN Notices, 49(6), 419–428. [doi:10.1145/2594291.2594321]

[56] Tu Z, Su Z, Devanbu P. On the localness of software. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2014: 269-280. [doi:10.1145/2635868.2635875]

[57] Proksch S, Lerch J, Mezini M. Intelligent code completion with Bayesian networks. ACM Trans. on Software Engineering and Methodology. 2015, 25(1): 1-31. [doi:10.1145/2744200]

[58] Bruch M, Monperrus M, Mezini M. Learning from examples to improve code completion systems. In: Proc. of the the 7th joint meeting of the European software engineering Conf. and the ACM SIGSOFT Symp. on The foundations of software engineering. ACM, 2009: 213-222. [doi:10.1145/1595696.1595728]

[59] Nguyen, T. T., Nguyen, A. T., Nguyen, H. A., & Nguyen, T. N. A statistical semantic language model for source code. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. 2013, 532. [doi:10.1145/2491411.2491458]

[60] Nguyen, A. T., & Nguyen, T. N. Graph-Based Statistical Language Model for Code. 2015 IEEE/ACM 37th IEEE Int'l Conf. on Software Engineering, 858–868. [doi:10.1109/icse.2015.336]

[61] Nguyen A T, Hilton M, Codoban M, et al. API code recommendation using statistical learning from fine-grained changes. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2016: 511-522. [doi:10.1145/2950290.2950333]

[62] Bielik P, Raychev V, Vechev M. PHOG: probabilistic model for code. Int'l Conf. on Machine Learning. 2016: 2933-2942.

[63] Holmes R, Walker R J, Murphy G C. Strathcona example recommendation tool. In: Proc. of the 10th European Software Engineering Conf. Held Jointly with 13th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. 2005. [doi:10.1145/1081706.1081744]

[64] Zhang H, Jain A, Khandelwal G, et al. Bing developer assistant: improving developer productivity by recommending sample code. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2016: 956-961. [doi:10.1145/2950290.2983955]

[65] Mei H, Wang QX, Zhang L, Wang J. Software Analysis: A Road Map. Ji Suan Ji Xue Bao/Chinese Journal of Computers, 2009,32(9):1697−1710 (in Chinese with English abstract). [doi:10.3724/SP.J.1016.2009.01697]

[66] Shaw M. Truth vs. knowledge: The difference between what a component does and what we know it does. In: Proc. of the 8th Int'l Workshop on Software Specification and Design. IEEE, 1996: 181-185. [doi:10.1109/iwssd.1996.501165]

[67] Rahman F, Khatri S, Barr E T, et al. Comparing static bug finders and statistical prediction. In: Proc. of the 36th Int'l Conf. on Software Engineering. ACM, 2014: 424-434. [doi:10.1145/2568225.2568269]

[68] Ray B, Hellendoorn V, Godhane S, et al. On the naturalness of buggy code. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM, 2016: 428-439. [doi:10.1145/2884781.2884848]

[69] Wang S, Chollak D, Movshovitz-Attias D, et al. Bugram: bug detection with n-gram language models. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2016: 708-719. [doi:10.1145/2970276.2970341]

[70] Chen F, Kim S. Crowd debugging. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 320-332. [doi:10.1145/2786805.2786819]

[71] Campbell J C, Hindle A, Amaral J N. Syntax errors just aren't natural: improving error reporting with language models. In: Proc. of the 11th Working Conf. on Mining Software Repositories. ACM, 2014: 252-261. [doi:10.1145/2597073.2597102]

[72] Hanam Q, Brito F S M, Mesbah A. Discovering bug patterns in JavaScript. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2016: 144-156. [doi:10.1145/2950290.2950308]

[73] Li Z, Zhou Y. PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code. ACM SIGSOFT Software Engineering Notes. ACM, 2005, 30(5): 306-315. [doi:10.1145/1081706.1081755]

[74] Liang B, Bian P, Zhang Y, et al. AntMiner: mining more bugs by reducing noise interference. In: Proc. of the 38th Int'l Conf. on Software Engineering. IEEE, 2016: 333-344. [doi:10.1145/2884781.2884870]

[75] Murali V, Chaudhuri S, Jermaine C. Finding Likely Errors with Bayesian Specifications. In: Proc. of the 25th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2017.

[76] Raychev V, Vechev M, Krause A. Predicting program properties from big code. ACM SIGPLAN Notices. ACM, 2015, 50(1): 111-124. [doi:10.1145/2676726.2677009]

[77] Nguyen H A, Dyer R, Nguyen T N, et al. Mining preconditions of APIs in large-scale code corpus. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2014: 166-177. [doi:10.1145/2635868.2635924]

[78] Xu Z, Zhang X, Chen L, et al. Python probabilistic type inference with natural language support. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2016: 607-618. [doi:10.1145/2950290.2950343]

[79] Allamanis M, Sutton C. Mining idioms from source code. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2014: 472-483. [doi:10.1145/2635868.2635901]

[80] Allamanis M, Barr E T, Bird C, et al. Learning natural coding conventions. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2014: 281-293. [doi:10.1145/2635868.2635883]

[81] Allamanis M, Barr E T, Bird C, et al. Suggesting accurate method and class names. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 38-49. [doi:10.1145/2786805.2786849]

[82] Treude C, Robillard M P. Augmenting API documentation with insights from Stack Overflow. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM, 2016: 392-403. [doi:10.1145/2884781.2884800]

[83] Mani S, Catherine R, Sinha V S, et al. Ausum: approach for unsupervised bug report summarization. In: Proc. of the ACM SIGSOFT 20th Int'l Symp. on the Foundations of Software Engineering. ACM, 2012: 11. [doi:10.1145/2393596.2393607]

[84] Karaivanov S, Raychev V, Vechev M. Phrase-based statistical translation of programming languages. In: Proc. of the 2014 ACM Int'l Symp. on New Ideas, New Paradigms, and Reflections on Programming & Software. ACM, 2014: 173-184. [doi:10.1145/2661136.2661148]

[85] Nguyen, A. T., Nguyen, T. T., & Nguyen, T. N. Lexical Statistical Machine Translation for Language Migration. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. 2013: 651–654. [doi:10.1145/2491411.2494584]

[86] Oda Y, Fudaba H, Neubig G, et al. Learning to generate pseudo-code from source code using statistical machine translation. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. IEEE, 2015: 574-584. [doi:10.1109/ase.2015.36]

[87] Xuan JF, Ren ZL, Wang ZY, Xie XY, Jiang H. Progress on approaches to automatic program repair. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):771−784 (in Chinese). http://www.jos.org.cn/1000-9825/4972.htm [doi:10.3724/SP.J.1016.2009.01697]

[88] Monperrus M. A critical review of automatic patch generation learned from human-written patches: essay on the problem statement and the evaluation of automatic software repair. In: Proc. of the 36th Int'l Conf. on Software Engineering. ACM, 2014: 234-242. [doi:10.1145/2568225.2568324]

[89] Vechev M, Yahav E. Programming with "Big Code". Foundations and Trends® in Programming Languages, 2016, 3(4): 231-284. [doi:10.1561/2500000028]

[90] https://www.codota.com/

附中文参考文献:

[65] 梅宏, 王千祥, 张路, 等. 软件分析技术进展. 计算机学报, 2009, 32(9): 1697-1710. [doi:10.3724/SP.J.1016.2009.01697]

[87] 玄跻峰,任志磊,王子元,谢晓园,江贺.自动程序修复方法研究进展.软件学报,2016,27(4):771−784. http://www. jos.org.cn/1000-9825/4972.htm [doi:10.3724/SP.J.1016.2009.01697]