

基于图模型的克隆代码演化痕迹构建及模式识别

葛广帅, 刘东升, 张丽萍, 侯 敏

(内蒙古师范大学 计算机与信息工程学院, 呼和浩特 010022)

摘 要: 针对克隆跟踪不精确、演化模式识别繁琐以及克隆群合并现象处理困难等问题, 提出一种改进的克隆代码演化痕迹构建及模式识别方法。在相邻版本使用主题概率模型实现克隆群初步映射, 计算代码位置重叠率及文本相似度完成克隆片段映射, 修复克隆群映射得到精确的相邻版本克隆映射结果。依据相邻版本间建立映射的克隆群数量关系及交叉程度识别短期演化模式, 构建图模型, 将克隆群作为点、映射关系作为边, 并根据产生形式为克隆群标注短期演化模式。使用广度优先搜索算法提取克隆家系, 按照克隆家系中包含的克隆群种类及是否有环识别长期演化模式。对 5 款开源软件的 70 个版本进行实验, 结果表明, 运用该方法约 95% 的克隆在演化中保持稳定, 约 1% 的克隆经历了合并复合, 并且 80% 左右克隆代码的生命周期未超过发布版本总数的一半。

关键词: 图模型; 克隆跟踪; 演化模式; 克隆家系; 克隆代码

中文引用格式: 葛广帅, 刘东升, 张丽萍, 等. 基于图模型的克隆代码演化痕迹构建及模式识别[J]. 计算机工程, 2017, 43(5): 47-54, 59.

英文引用格式: Ge Guangshuai, Liu Dongsheng, Zhang Liping, et al. Clone Code Evolution Trace Building and Pattern Recognition Based on Graph Model[J]. Computer Engineering, 2017, 43(5): 47-54, 59.

Clone Code Evolution Trace Building and Pattern Recognition Based on Graph Model

GE Guangshuai, LIU Dongsheng, ZHANG Liping, HOU Min

(College of Computer and Information Engineering, Inner Mongolia Normal University, Hohhot 010022, China)

[Abstract] Aiming at the problems that clone code tracking is inaccurate and evolution pattern recognition is tedious, especially clone class merging is difficult, this paper proposes a method of clone code evolution traces building and pattern recognition based on graph model. Clone class mapping is implemented preliminarily between adjacent versions using theme probability model, location overlap rate of clone code and cloned fragment text similarity are calculated to map clone fragments, clone class mapping is repaired to get accurate mapping results between adjacent versions. Short-term evolution pattern is recognized and figure model is established on the basis of relationship of clone class mapping quantity and the degree of cross. Considering clone class as points, mapping relationship as edges, and according to the form of production, short-term evolution model of clone class is indicated. The clone genealogy is extracted using breadth-first search algorithm, and the long-term evolution model is identified according to types of clone class included in clone genealogy and whether there is loop among them. Experiment is conducted on 70 versions of 5 open source software. Results show that about 95% of the clone code is stable in evolution, about 1% of clone code undergoes merging or recombination, and about 80% of clone code's life cycle does not exceed half of the total quantity of released version.

[Key words] graph model; clone tracking; evolution pattern; clone genealogy; clone code

DOI: 10.3969/j.issn.1000-3428.2017.05.008

0 概述

克隆代码是软件系统中的一些代码片段, 这些

代码片段之间存在着相同或相似语法及语义特征^[1]。有研究表明, 一个软件系统中存在 9% ~ 17% 的克隆代码, 有时甚至高达 50% 以上。克隆

基金项目: 国家自然科学基金(61462071, 61363017); 内蒙古自然科学基金(2014MS0613, 2015MS0606, 2016MS0612); 内蒙古自治区高等学校科学研究计划项目(NJZY16045)。

作者简介: 葛广帅(1988—), 男, 硕士研究生, 主研方向为软件系统分析; 刘东升(通信作者)、张丽萍, 教授; 侯 敏, 讲师、硕士。

收稿日期: 2016-08-11 **修回日期:** 2016-09-28 **E-mail:** 15248084113@163.com

代码产生原因主要有:1)程序员在软件开发、维护过程中“复制、粘贴、修改”等活动产生的主动克隆^[2];2)使用设计模式、框架、相似 API 等产生的被动克隆。克隆代码与软件工程领域的各种问题密切相关,例如软件质量、软件复用等问题^[3]。克隆代码的存在还给软件开发和维护带来了双重影响^[4],一方面复制一段无缺陷的代码,可降低开发新代码的成本;另一方面复制一段含有潜在 Bugs 的代码,将使得 Bugs 大量繁殖。然而,软件的单个版本中不包含克隆代码的变化信息,要对克隆代码分析和管理的,需要了解克隆代码在多版本中的演变情况^[5]。为了更好地进行克隆代码分析和管理的,在多版本间跟踪克隆,研究克隆进化受到极大关注。研究者们通过分析克隆代码在版本升级过程中的变化特点,为程序员理解、管理克隆代码提供帮助。

为研究克隆代码在整个生命周期中的变化情况,本文提出一种基于图模型的克隆代码演化痕迹

构建及模式识别方法,在软件多版本间跟踪克隆并识别演化模式。

1 相关研究

1.1 克隆相关定义

克隆代码是指语法或语义上相同或相似的代码片段^[1]。根据代码文本和功能的相似性程度,将克隆代码分为 4 种类型^[6]:Type-1 克隆,Type-2 克隆,Type-3 克隆,Type-4 克隆。图 1 展示了 4 种克隆类型的具体样例,其中,Type-1 克隆(CF 与 CF₁)除了空白符和注释外完全相同的代码段;Type-2 克隆(CF 与 CF₂)除了空白符、注释、标识符、类型外句法结构完全相同的代码段;Type-3 克隆(CF 与 CF₃)除了空白符、注释、标识符、类型外句法结构基本相同,但含有少量的语句修改的代码段;Type-4 克隆(CF 与 CF₄)功能相同但句法结构不同的代码段。

01:intSUM(intn) 02:{ 03:intsum=0;//CF 04:for(inti=1;i<=n;i++) 05:{ 06:sum=sum+i; 07;} 08:returns; 09:}	01:intSUM(intn) 02:{ 03:intsum=0;//CF1 04:for(inti=1;i<=n;i++) 05:{ sum=sum+i; } 06:returnsum;//return 07:}	01:doubleSUM(intn) 02:{ 03:double s=0;//CF2 04:for(inti=1;i<=n;i++) 05:{ 06: s=s+i;//add 07;} 08:returns; 09:}	01:doubleSUM(intn){ 02:double sum=0;//CF3 03:for(inti=1;i<=n;i++) 04:{ 05:sum+=i; 06;} 07:return sum; 08:}	01:int SUM(int n) 02:{ 03:int sum=0;//CF4 04:sum=n*(n+1)/2; 05:return sum; 06:}
(a)CF	(b)CF ₁	(c)CF ₂	(d)CF ₃	(e)CF ₄

图 1 4 种克隆类型的源代码示例

其中,Type-1 克隆被称为完全克隆;Type-2 克隆和 Type-3 克隆被称为近似克隆;Type-4 克隆被称为语义克隆。2 个克隆代码片段在语法或语义上相同或相似被称为一个克隆对。所有相同或相似的克隆代码的集合称为一个克隆群,克隆群中任意 2 个克隆代码都是克隆对。

1.2 克隆进化及演化模式

克隆演化研究是以软件系统的多个版本为基础,探究克隆代码在整个生命周期的进化及演变情况。它超越了以往只限于一个版本内的研究,从进化的角度探索克隆代码存在、发展、变化的规律。

1997 年,Lague 等人^[7]提出克隆进化的概念,并在一款电信系统上进行了克隆代码进化的跟踪,研究了 2 种类型的变化对软件进化的影响。2005 年,Kim 等人^[8]提出克隆家系模型,并利用版本管理工具(CVS)中的修改日志信息推算版本间的变化,得到克隆群的映射关系,从软件进化角度分析克隆变化,也是目前研究克隆代码演化的主要方法。他们依据版本管理工具中的修改日志,建立克隆群映射,提取克隆家系,分析了克隆群在软件多版本中的演化情况,但该方法必须依靠版本管理工具,并且该传统克隆家系模型将克隆群的演化视

为一个有向无环图,对于实际演化过程中出现克隆群复合或合并不能处理。文献[9]使用增量克隆检测方法构建克隆家系,但是该方法只能处理给定版本的软件,新版本加入需要重新执行,时间复杂度偏高。文献[10]利用克隆家系模型研究了克隆代码迁移对代码缺陷的影响。文献[11]对克隆家系做了进一步的研究,开发了一款克隆家系提取工具 gCad,重点研究了 4 种克隆家系(活家系、死家系、句法相似家系、一致变化家系)在软件系统中所占的比例,由于该工作首先进行函数映射,在此基础上再进行克隆映射,当函数被重命名或移动将影响映射正确率。2007 年,E. Duala-Ekoko 等人^[12]提出用 CRD(Clone Region Descriptor)表示克隆区域,开发了 CloneTracker 工具进行克隆代码跟踪。2013 年,慈萌等人^[13]改进 CRD 实现相邻版本克隆群的映射,最后基于二元关系合成克隆家系。如果代码某些部分被修改发生变化,会导致 CRD 失效,影响映射正确率。2015 年,涂颖等人^[14]提出一种基于克隆代码 Token 及位置属性跨版本克隆跟踪方法,分层次导出克隆群演化模式,最终实现了一款克隆家系自动提取工具 FCG,该工具能够有效地实现 Type-1 和 Type-2 类型的克隆群映射,但对于 Type-3 类型的克隆群映射有一定困难。

1.3 主题模型的应用

文献[15]最先使用主题模型的方法挖掘代码中包含的主题,对软件产品进行聚类。文献[16]基于主题建模技术研究软件的演化,通过主题的演化信息来分析软件在多版本的变化情况。文献[17]将主题模型应用在文档和代码间的相关性分析和可追踪性问题上。文献[18]应用主题建模技术分别提取 bug 报告和源代码的主题,根据主题信息实现 bug 在源代码中的定位。文献[19]使用主题概率模型提取直系克隆主题,预测克隆代码不一致性变化,对软件质量和可信性评估。文献[20]使用主题建模技术实现了克隆群映射,但他们是每个克隆群提取一个主题,最后使用该主题包含的前 20 个单词代替克隆群,实质就是词频最高的 20 个单词,没有使用主题概率潜在的语义信息,当克隆群修改稍大时,克隆群映射准确率会受到很大影响。

通过对现有研究的总结分析,发现现有研究在实现克隆代码跟踪及模式识别方面存在以下 7 点不足:

- 1) 严重依赖版本管理工具。
- 2) 映射时间复杂度高。
- 3) 可移植性差、编程语言不可扩展。
- 4) 无法处理新增克隆代码。
- 5) 无法处理演化过程中克隆群合并现象。
- 6) 演化过程中变化复杂时跟踪错误。
- 7) 演化模式识别复杂、繁琐。

2 克隆代码演化痕迹的构建及模式识别

本文提出的克隆代码演化痕迹构建及模式识别分为 5 个步骤:1) 克隆代码检测;2) 相邻版本克隆映射;3) 短期演化模式标注;4) 基于 BFS 克隆家系提取;5) 长期演化模式识别。本文方法的工作流程如图 2 所示。

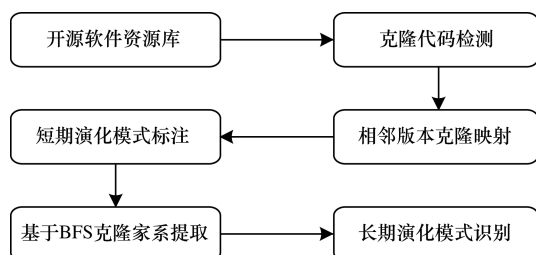


图 2 本文方法工作流程

在实现相邻版本克隆映射阶段,首先通过对相邻版本的 2 个克隆群提取 2 个主题,计算 2 个克隆群主题概率的欧式距离衡量相似度实现克隆群初步映射。与基于版本修改日志的方法相比,不需要依赖版本管理工具;与基于文本相似度的方法相比,时

间复杂度较低;与基于 AST,Token 的方法相比,实现简单,且与编程语言无关;与基于增量检测方法相比,可以处理后期版本新增的克隆;与基于 CRD 的方法相比,避免了稍大修改带来的映射错误现象。其次在克隆群映射的基础上,实现克隆片段映射。与先进行克隆片段映射再进行克隆群映射的方法相比,缩小了查找范围,降低了时间复杂度。最后依据克隆片段映射结果,修复克隆群映射。与传统的克隆映射相比,对于演化过程中变化复杂时,可去除一些错误映射,映射准确度更高。在克隆家系提取及长期演化模式识别阶段,本文方法建立图模型,将克隆群作为点、映射关系作为边,根据克隆群产生形式标注短期演化模式,使用广度搜索算法从非连通图中提取连通子图(即克隆家系),并依据克隆家系包含的克隆群标签种类及是否包含环识别长期演化模式。与传统克隆演化模式识别相比,该方法将短期演化模式标注在克隆群上,使得长期演化模式识别更简单、容易。并且该方法实现多版本克隆跟踪使用图模型,将有关联的克隆群都提取在一个克隆家系中,可以处理克隆群合并现象。

2.1 克隆代码检测

克隆代码检测是研究克隆的基础工作,并且克隆检测结果直接影响后期对克隆的研究,所以本文使用目前较成熟的克隆检测工具 NiCad^[21]进行克隆检测。NiCad 是由加拿大皇后大学开发的克隆检测工具,它能够检测 Type1, Type2, Type3 类型的克隆代码,而且能够设置不同的粒度:函数粒度,块粒度,以 xml 的形式返回克隆检测信息。本文研究采用 NiCad 默认的配置信息,以块为粒度对多个版本单独检测。为了方便完成克隆检测工作,本文编写了 python 脚本实现对一款软件多个版本自动化检测。

2.2 相邻版本克隆映射

构建克隆代码演化痕迹,也就是在多版本间跟踪克隆代码,实现相邻版本克隆映射是克隆代码跟踪的基础。本文分 3 步实现相邻版本的克隆映射:1) 基于主题概率分布克隆群映射;2) 依据位置重叠率及文本相似度建立克隆片段映射;3) 根据克隆片段映射结果修复克隆群映射,得到精确的相邻版本克隆映射。

2.2.1 基于主题概率分布的克隆群映射

由于同源克隆群描述的代码差异性很小甚至相同,而非同源克隆群包含的代码差异性很大,因此本文采用基于主题概率分布的方法,对 2 个克隆群 A, B 进行主题概率模型训练,提取 2 个主题 topic1, topic2, 如果克隆群 A, B 是非同源的,那么克隆群 A 描述的是一个“故事”,而克隆群 B 描述是另一个“故事”,克隆群 A, B 必然在主题 topic1, topic2 上分布差异明显;

相反,如果克隆群 A,B 是同源的,那么克隆群 A,B 讲的是同一个“故事”,虽然要分布在 2 个主题 topic1, topic2 上,但是在这 2 个主题上的分布也是极度相似的。表 1、表 2 描述了 xorriso 软件中非同源克隆群的距离和同源克隆群的距离,多次主题训练虽然有些波动,但是同源克隆群距离一直很小,而非同源克隆群距离一直保持很大,且差距明显。

表 1 非同源克隆群之间的距离

N	克隆群信息	Topic1	Topic2	欧式距离
1	xorriso-0.4.9:1	0.000 126	0.999 874	1.413 843
	xorriso-0.4.9:2	0.999 864	0.000 136	
2	xorriso-0.4.9:1	0.000 120	0.999 880	1.413 861
	xorriso-0.4.9:2	0.999 871	0.000 129	
3	xorriso-0.4.9:1	0.000 120	0.999 880	1.413 860
	xorriso-0.4.9:2	0.999 870	0.000 130	
4	xorriso-0.4.9:1	0.000 137	0.999 863	1.413 792
	xorriso-0.4.9:2	0.999 839	0.000 161	
5	xorriso-0.4.9:1	0.999 882	0.000 118	1.413 867
	xorriso-0.4.9:2	0.000 127	0.999 873	

表 2 同源克隆群之间的距离

N	克隆群信息	Topic1	Topic2	欧式距离
1	xorriso-0.4.9:1	0.400 585	0.599 415	0.004 135
	xorriso-0.5.0:1	0.397 661	0.602 339	
2	xorriso-0.4.9:1	0.378 655	0.621 345	0.026 878
	xorriso-0.5.0:1	0.397 661	0.602 339	
3	xorriso-0.4.9:1	0.558 480	0.441 520	0.053 757
	xorriso-0.5.0:1	0.520 468	0.479 532	
4	xorriso-0.4.9:1	0.600 877	0.399 123	0.022 743
	xorriso-0.5.0:1	0.584 796	0.415 204	
5	xorriso-0.4.9:1	0.479 532	0.520 468	0.035 149
	xorriso-0.5.0:1	0.504 386	0.495 614	

在版本升级过程中,可能会对同一个克隆群包含的克隆片段不一致进行修改,超出了克隆检测工具 NiCad 设置的阈值,使得 1 个克隆群分裂成 2 个克隆群或多个克隆群,但后期也可能再次同步修改,使多个克隆群合成一个克隆群。因此,本文在相邻版本间实现克隆群映射的思路是:对于前期版本的一个克隆群 CC 与后期版本的任何一个克隆群都要计算它们之间的距离,只要小于等于阈值,就建立映射。具体的过程如算法 1 所示。

算法 1 相邻版本克隆群映射算法

函数功能:在相邻版本间建立克隆群映射

输入 前期版本克隆群集合,后期版本克隆群集合

输出 相邻版本间克隆群映射列表

```

1: function CLONECLASSMAPPING ( preversion,
lastversion)
2:  ccmlist = []
3:  for precc in preversion do

```

```

4:    for lastcc in lastversion do
5:      distance = DISTANCE( precc, lastcc)
6:      if distance <= Θ then //Θ is threshold
7:        ccmlist. append( ( precc, lastcc, distance) )
8:      end if
9:    end for
10:  end for
11:  return ccmlist
12: end function

```

2.2.2 克隆片段映射

为了更细致地跟踪克隆演化,在克隆群映射的基础上,再进行克隆片段映射,既不会丢失克隆跟踪信息,又提高了映射速度。

对于版本 $V(i)$ 中克隆群 CG 的一个克隆片段 CF,在后期版本 $V(i+1)$ 中寻找映射片段的过程如下:

1) 收集 $V(i+1)$ 版本中和克隆群 CG 有映射关系的所有克隆群,构成集合 mappingCGSet。

2) 如果克隆群集合 mappingCGSet 为空,那么克隆片段 CF 在后期版本不存在映射对象,结束程序。

3) 克隆群集合 mappingCGSet 不为空,将克隆群集合 mappingCGSet 中的每个克隆群包含的克隆片段汇总到候选克隆片段集合 candidateCFSet。

4) 计算候选克隆片段集合 candidateCFSet 中的每个与 CF 同文件的克隆片段与 CF 之间的位置重叠率(如式(1)所示)及文本相似度(如式(2)所示)。计算位置重叠率与文本相似度之和,选出最匹配的映射克隆片段。

$$overlap(CF_1, CF_2) = \frac{\min(e_1, e_2) - \max(s_1, s_2)}{e_1 - s_1} \quad (1)$$

$$similarity(CF_1, CF_2) = \frac{2EditDis(CF_1, CF_2)}{|CF_1| + |CF_2|} \quad (2)$$

其中, s_1, e_1 分别表示 CF_1 克隆片段的起始结束行; s_2, e_2 分别表示 CF_2 克隆片段的起始结束行; $|CF_1|, |CF_2|$ 分别表示克隆片段 CF_1, CF_2 的长度; $EditDis(CF_1, CF_2)$ 表示克隆片段 CF_1, CF_2 的编辑距离。

2.2.3 克隆群映射结果修复

克隆映射有效性直接决定克隆代码跟踪效果,也是后期克隆演化特征分析的基础,目前传统的克隆群映射分为 2 种方法(下面的图例中椭圆代表克隆群,矩形框代表克隆片段)。

方法 1 对相邻版本中的克隆群两两对比,计算相似度,在阈值范围之内就建立映射。本文即采用这种方法,但是会出现如图 3 所示的错误映射,由于克隆群 B,C,D,E 都是来源克隆群 A,4 个克隆群极其相似,会出现版本 $V(i+1)$ 中的克隆群 B 与版本 $V(i+2)$ 中的克隆群 E 误映射、版本 $V(i+1)$ 中的克隆群 C 与版本 $V(i+2)$ 中的克隆群 D 误映射。

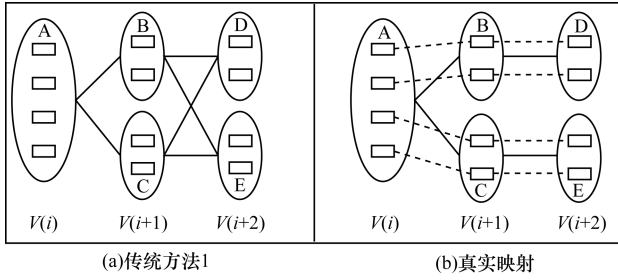


图 3 传统克隆群映射方法 1

方法 2 对于版本 $V(i)$ 中的克隆群 CG, 在后期版本 $V(i+1)$ 中找一个最相似的克隆群建立映射, 其他克隆群不再考虑, 会出现如图 4 所示的遗漏映射, 克隆群 B, C 都来源于克隆群 A, 结果由于只选最相似的克隆群建立映射, 漏掉克隆群 A 到克隆群 C 的映射。

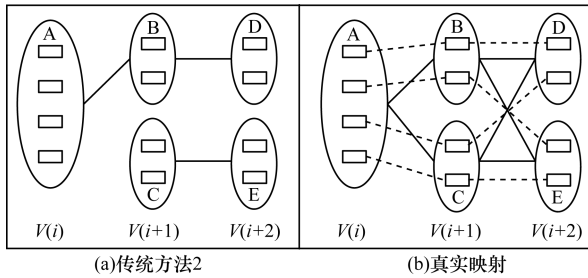


图 4 传统克隆群映射方法 2

所以, 在克隆片段映射之后, 还要根据克隆群内部的实际克隆片段映射情况, 整理修复克隆群映射。

2.3 短期演化模式标注

克隆群短期演化模式可用于研究克隆代码在相邻版本间演化特征, 帮助开发人员及时了解克隆群动态变化规律。在相邻版本克隆群映射的基础上, 按照具有映射关系的克隆群在前后版本的数量比例及交叉程度, 克隆群短期演化模式分为 6 种, 即新生、死亡、简单、分离、合并、复杂。图 5 展示了克隆群短期演化模式模型, 椭圆代表克隆群, 虚椭圆表示克隆群不存在, 虚线上方为版本 $V(i)$ 的克隆群, 虚线下方为版本 $V(i+1)$ 的克隆群, 椭圆之间的实线表示 2 个克隆群具有映射关系。

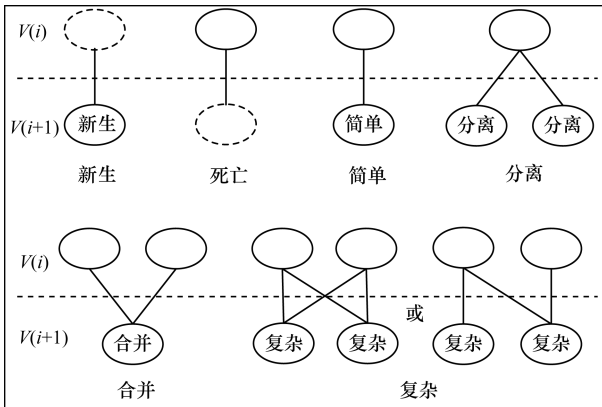


图 5 克隆群短期演化模式

为了方便在整个生命周期跟踪研究克隆群的演化特点, 本文建立图模型, 将克隆群看成图中的点, 映射关系看成图中的边。依据克隆群产生形式, 将克隆群短期演化模式标注在克隆群上, 由于死亡模式不能表达产生形式, 这里不予考虑, 只标注新生、简单、分离、合并、复杂 5 种短期演化模式。克隆群短期演化模式标注步骤如下:

第 1 步 如果克隆群 CC 不存在前驱, 则该克隆群 CC 标注为“新生”。

第 2 步 如果克隆群 CC 存在一个前驱, 并且该前驱克隆群只有一个后继 (即克隆群 CC), 则克隆群 CC 标注为“简单”。

第 3 步 如果克隆群 CC 存在一个前驱, 但是该前驱克隆群有多个后继 (即除了克隆群 CC 之外还有其他克隆群), 而这些后继克隆群都只有一个前驱, 则克隆群 CC 标注为“分离”。

第 4 步 如果克隆群 CC 存在多个前驱, 并且所有的前驱都只有一个后继 (即克隆群 CC), 则克隆群 CC 标注为“合并”。

第 5 步 如果克隆群 CC 经过前 4 步都没有被标注, 那么克隆群 CC 为复杂模式, 标注为“复杂”。

2.4 基于 BFS 的克隆家系提取

同源克隆群在多版本中的不同表现构成了克隆家系, 克隆家系反映了克隆群在软件演化过程中的变化情况。在图模型的基础上, 本文基于 BFS 算法从非连通图中提取连通子图 (即克隆家系), 具体步骤如算法 2 所示。

算法 2 克隆家系提取算法

函数功能: 提取克隆家系

输入 克隆群列表, 映射关系

输出 带克隆家系编号的克隆群列表

```

1: function EXTRACTGENEALOGY( cclist, maplist)
2:   for cc in cclist do
3:     cc.genid = -1
4:   end for
5:   genid = 0 //genealogy id
6:   for cc in cclist do
7:     if cc.genid == -1 then
8:       genid = genid + 1
9:       //begin extract one genealogy
10:      BFSGENEALOGY( cc, cclist, maplist, genid)
11:    end if
12:  end for
13: end function
14: function BFSGENEALOGY( cc, cclist, maplist, genid)
15:   q = Queue()
16:   cc.genid = genid
17:   cclist[cc].genid = genid
18:   q.put(cc)
19:   while q.empty() == False do
20:     front = q.get()

```

```

21:   mapcclist = GETMAPCCLIST( front, maplist)
22:   for tmpcc in mapcclist do
23:       if tmpcc. genid == -1 then
24:           tmpcc. genid = genid
25:           cclist[ tmpcc ]. genid = genid
26:           q. put( tmpcc )
27:       end if
28:   end for
29: end while
30: end function
31: function GETMAPCCLIST( cc, maplist)
32:   mapcclist = []
33:   for mappair in maplist do
34:       if mappair. Acc == cc then
35:           mapcclist. append( mappair. Bcc )
36:       end if
37:       if mappair. Bcc == cc then
38:           mapcclist. append( mappair. Acc )
39:       end if
40:   end for
41: return mapcclist
42: end function

```

2.5 长期演化模式识别

根据克隆家系中克隆群经历的短期演化模式(即克隆群标签),可以将克隆群长期演化模式(或称为克隆家系模式)分为6种:即常规、独立、合并、自由、复合、混合,如图6所示。

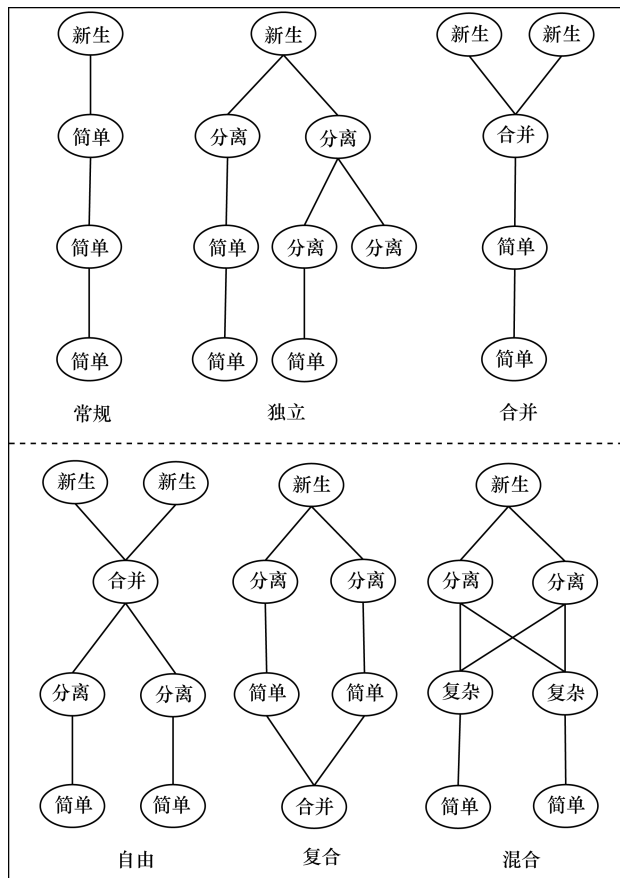


图6 克隆群长期演化模式

统计克隆家系中各种短期演化模式标签的克隆群数量及是否有环识别长期演化模式步骤如下:

第1步 如果包含“复杂”克隆群,那么为“混合”演化模式。

第2步 如果只包含“新生”“简单”标签的克隆群,那么为“常规”演化模式。

第3步 如果包含“分离”克隆群,但是不包含“合并”、“复杂”克隆群,那么为“独立”演化模式。

第4步 如果包含“分离”“合并”克隆群,但不包含“复杂”克隆群,那么需要判断是否有环,如果有环为“复合”演化模式,无环为“自由”演化模式。

3 实验结果与分析

3.1 实验数据

本文根据前面提到的克隆代码演化痕迹构建及模式识别方法实现了一个自动化克隆代码跟踪工具CloneCodeTracer,并在5款开源软件上进行了克隆代码跟踪及模式识别实验。选用的实验软件来自不同的应用领域,软件规模大小不一,而且还考虑了不同开发语言的差异性,并且这些开源代码均拥有较多的发布版本一直被良好地开发和维护。实验软件的基本信息如表3所示。

表3 目标软件基本信息

软件名称	开发语言	版本数量	应用领域
bluefish	C	13	Web 设计编辑器
ctags	C	18	索引工具
ffmpeg	C	20	多媒体视频处理
jedit	Java	11	文本编辑器
argouml	Java	8	UML 模型工具

3.2 关键参数选取

3.2.1 评价方法

为了说明本文选取参数的有效性,本文采用人工验证的方法从查准率、查全率2个方面进行衡量。

$$precision = \frac{TT}{TT + TF} \quad (3)$$

$$recall = \frac{TT}{TT + FT} \quad (4)$$

其中,TT表示发现的正确的数量;TT+TF表示所有发现的数量;TT+FT表示实际存在的数量。

3.2.2 克隆群初步映射阈值选取

相邻版本克隆群初步映射,是通过计算2个克隆群在2个主题概率上的分布向量的欧式距离,当

欧式距离在阈值范围之内就建立映射。通过在多款软件实验,并经过人工验证,发现同源克隆群之间的欧式距离明显小于非同源克隆群之间的欧式距离,为了表明阈值选取的合理性,本文选取不同阈值,计算映射查全率、查准率变化如图 7 所示。由于本文在克隆群初步映射之后,还要进行克隆片段映射,最后根据克隆片段实际映射情况修复克隆群映射,因此查全率是比较重要的,在保证查全率为 100% 的情况下,本文选取较小的阈值,获得较高的查准率,尽量缩小后期克隆片段映射的计算量,最终欧式距离阈值选取 0.25。

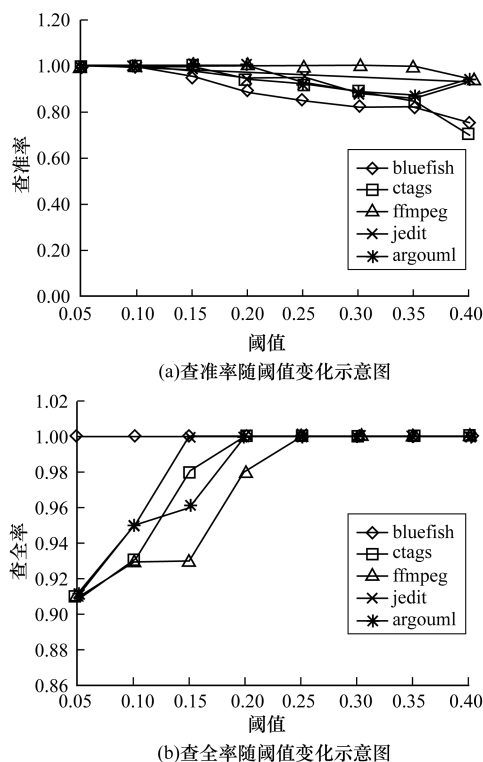


图 7 查准率、查全率随阈值变化示意图

3.3 结果分析

通过对表 3 提到的 5 款软件构建克隆代码演化痕迹及模式识别,得到每款软件克隆群在进化过程中的短期演化模式和长期演化模式。表 4 统计了各短期演化模式克隆群所占比例,可以看出本文提到的五种短期演化模式确实存在。除了新生模式,基本上都是简单模式,一定程度上反映了克隆代码的稳定性。合并模式存在很少,但占有一定的数量。例如 ctags 软件 18 个版本演化过程中,就出现了 1.83% 的合并模式的克隆群。表 5 从克隆群的整个生命周期统计了各种长期演化模式克隆家系比例。合并模式、复合模式确实存在,克隆群在演化过程中

先出现分离,后期又合并,这种往复操作很可能引起 Bugs,应该引起开发人员的高度关注。

表 4 克隆群短期演化模式比例 %

软件名称	新生	简单	分离	合并	复杂
bluefish	15.78	83.70	0.00	0.51	0.00
ctags	23.08	72.89	2.20	1.83	0.00
ffmpeg	17.95	78.51	0.02	0.88	2.63
jedit	21.30	76.32	0.00	0.63	1.75
argouml	40.20	56.84	0.36	0.75	1.86

表 5 克隆群(克隆家系)长期演化模式比例 %

软件名称	混合	常规	独立	合并	复合	自由
bluefish	0.00	97.75	0.00	2.25	0.00	0.00
ctags	0.00	94.83	1.72	1.72	0.00	1.72
ffmpeg	2.27	96.21	0.06	1.17	0.14	0.14
jedit	0.61	97.56	0.00	0.61	0.00	1.22
argouml	1.38	97.16	0.28	0.73	0.46	0.00

除了关注克隆群的演化模式外,本文还统计分析了克隆家系的生存寿命。如表 6 所示,大部分克隆家系只是短暂存在,很少克隆家系一直稳定存活。生存时间低于软件整个生命周期 50% 的占 80% 左右,生存时间超过整个软件生命周期 75% 的不足 15%,当然也有例外,如 bluefish 软件生存时间超过 75% 的就达到 37%。

表 6 克隆群生存周期分布 %

软件名称	克隆群生存周期区间			
	0 ~ 25	25 ~ 50	50 ~ 75	75 ~ 100
bluefish	7.87	39.33	15.73	37.08
ctags	74.14	8.62	3.45	8.99
ffmpeg	61.64	20.32	8.95	9.09
jedit	21.34	45.73	18.29	14.63
argouml	59.45	30.55	5.87	4.13

3.4 同类实验对比分析

目前,已有不少克隆跟踪工具被开发,其中加拿大皇后大学开发的克隆家系提取工具 gCad 影响力最大。检测阶段都使用 NiCad 检测工具,参数使用默认值,对表 3 提到的 5 款开源软件在 gCad 和 CloneCodeTracer 上做了对比实验,并经过人工验证,结果如表 7 所示,其中,CCT 表示 CloneCodeTracer;“-”表示不能识别。从表 7 可以看到出,gCad 不能识别合并、复合等演化模式,而经过人工检验,这些演化模式确实存在,并且比较都能识别的演化模式(例如常规、独立),CloneCodeTracer 的准确率明显高于 gCad。

表 7 不同克隆群(克隆家系)长期演化模式下准确率对比

软件名称	衡量参数	混合		常规		独立		合并		复合		自由	
		gCad	CCT	gCad	CCT	gCad	CCT	gCad	CCT	gCad	CCT	gCad	CCT
bluefish	precision	-	1.000	0.974	0.990	1.000	1.000	-	0.983	-	1.000	-	1.000
	recall	-	0.000	0.997	0.992	1.000	1.000	-	0.994	-	1.000	-	1.000
ctags	precision	-	1.000	0.972	0.989	0.979	0.995	-	0.984	-	1.000	-	0.500
	recall	-	1.000	0.990	0.985	0.953	1.000	-	0.935	-	1.000	-	1.000
ffmpeg	precision	-	0.900	0.975	0.978	1.000	1.000	-	1.000	-	1.000	-	0.000
	recall	-	1.000	0.984	0.989	1.000	1.000	-	1.000	-	1.000	-	1.000
jedit	precision	-	1.000	0.951	0.964	1.000	1.000	-	1.000	-	1.000	-	1.000
	recall	-	1.000	0.977	0.980	1.000	1.000	-	1.000	-	0.000	-	0.900
argouml	precision	-	0.800	0.949	0.987	1.000	1.000	-	1.000	-	0.500	-	1.000
	recall	-	1.000	0.968	0.992	1.000	1.000	-	1.000	-	1.000	-	1.000

4 结束语

本文提出一种基于图模型的克隆代码演化痕迹构建方法,用于识别一些演化模式,分析各模式所占比例及克隆家系寿命。通过实验发现合并、复合现象确实存在,95% 克隆代码稳定演化,80% 的克隆代码生存周期都不超过发布版本总数的一半。本文克隆代码演化痕迹构建方法可以为克隆代码的跟踪、分析提供基础数据,为克隆管理提供参考信息。本文的研究内容与实验仍然存在不足之处,如演化模式只从克隆群角度分析粒度太粗,在下一步研究中将陆续加以改进、完善,并根据克隆对、克隆片段内容变化分析克隆演化。

参考文献

- [1] Bettenburg N, Shang W, Ibrahim W M, et al. An Empirical Study on Inconsistent Changes to Code Clones at the Release Level [J]. Science of Computer Programming, 2012, 77(6): 760-776.
- [2] Zibran M F, Roy C K. The Road to Software Clone Management: A Survey; Technical Report 2012-03 [R]. Canada, Saskatoon: The University of Saskatchewan, 2012: 1-66.
- [3] Rainer K, Elmar J, Juergen R. 7th International Workshop on Software Clones (IWSC 2013) [C]// Proceedings of International Conference on Software Engineering. San Francisco, USA: IEEE Press, 2013: 269-270.
- [4] Roy C K. Detection and Analysis of Near-miss Software Clones [C]// Proceedings of IEEE International Conference on Software Maintenance. Canada, Kingston: Queen's University Press, 2009: 447-450.
- [5] Pate J R, Tairas R, Kraft N A. Clone Evolution: A Systematic Review [J]. Journal of Software Maintenance & Evolution Research & Practice, 2013, 25(3): 261-283.
- [6] Roy C K, Cordy J R, Koschke R. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach [J]. Journal of the International Society for Twin Studies, 2009, 74(7): 470-495.
- [7] Lague B, Proulx D, Mayrand J, et al. Assessing the Benefits of Incorporating Function Clone Detection in a Development Process [C]// Proceedings of International Conference on Software Maintenance. [S. l.]: IEEE Computer Society, 1997: 314-321.
- [8] Kim M, Sazawal V, Notkin D, et al. An Empirical Study of Code Clone Genealogies [J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(5): 187-196.
- [9] Gode N, Koschke R. Incremental Clone Detection [C]// Proceedings of the 13th European Conference on Software Maintenance and Reengineering. Piscataway, USA: IEEE Press, 2009: 219-228.
- [10] Keivanloo I. An Empirical Study on the Fault-proneness of Clone Migration in Clone Genealogies [C]// Proceedings of Software Evolution Week-IEEE Conference on Software Maintenance. [S. l.]: Reengineering and Reverse Engineering, 2014: 94-103.
- [11] Saha R K, Roy C K, Schneider K A. gCad: A Near-miss Clone Genealogy Extractor to Support Clone Evolution Analysis [C]// Proceedings of the 29th IEEE International Conference on Software Maintenance. Eindhoven, the Netherlands: IEEE Press, 2013: 488-491.
- [12] Duala-Ekoko E, Robillard M P. Tracking Code Clones in Evolving Software [J]. Software Engineering, 2007, 116(67): 158-167.
- [13] Meng C, Su X H, Wang T T, et al. A New Clone Group Mapping Algorithm for Extracting Clone Genealogy on Multi-version Software [C]// Proceedings of the 3rd International Conference on Instrumentation, Measurement, Computer, Communication and Control. [S. l.]: IEEE Press, 2013: 848-853.
- [14] 涂颖, 张丽萍, 王春晖, 等. 基于软件多版本演化提取克隆谱系 [J]. 计算机应用, 2015, 35(4): 1169-1173.
- [15] Kuhn A, Ducasse S, Girba T. Semantic Clustering: Identifying Topics in Source Code [J]. Information & Software Technology, 2007, 49(3): 230-243.
- [16] Thomas S W, Adams B, Hassan A E, et al. Studying Software Evolution Using Topic Models [J]. Science of Computer Programming, 2014, 80: 457-479.

(下转第 59 页)

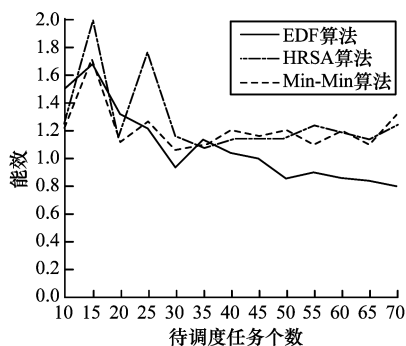


图3 不同调度算法的能效比较

3 结束语

本文在 Min-Min 算法中融合了 LSF, EDF 和 HVF 算法的调度策略并加以改进,提出一种可缩短系统对高实时性任务响应时间的调度算法 HRSA。下一步将研究如何减少调度算法能耗和颠簸次数,解决多核异构处理器上的混合任务调度问题。

参考文献

- [1] 汪少敏,赵 猛.基于多核处理器并发计算软件构架设计与实现[J]. 计算机科学,2008,35(7):283-285.
- [2] Tu Shengru, Flanagan M, Wu Ying, et al. Design Strategies to Improve Performance of GIS Web Services[C]//Proceedings of International Conference on Information Technology: Coding and Computing. Washington D. C., USA: IEEE Press, 2004:444-448.
- [3] 王 鑫.一种负载均衡的网格任务调度算法[J]. 计算机工程与科学,2008,30(9):86-88.
- [4] 朱晓敏,祝江汉,马满好.异构集群系统中具有 QoS 需求的实时任务容错调度[J]. 软件学报,2011,22(7):1440-1456.
- [5] Yang Ping, Cao Ying, Evan S J. Web Map Server Performance and Client Design Principles[J]. GIScience & Remote Sensing, 2007,44(4):320-333.
- [6] 赵 英,李 栋.改进的 Min-Min 网格任务调度算法[J]. 电子设计工程,2012,20(13):1-3.

- [7] Etminani K, Naghibzadeh M. A Min-Min Max-Min Selective Algorithm for Grid Task Scheduling [C]//Proceedings of the 3rd IEEE/IFIP International Conference in Central Asia. Washington D. C., USA: IEEE Press, 2007: 1-7.
- [8] 刘立平,牛 熠.遗传算法综述[J]. 东莞理工学院学报,2005,12(3):48-52.
- [9] 李建峰,彭 舰.云计算环境下基于改进遗传算法的任务调度算法[J]. 计算机应用,2011,31(1):184-186.
- [10] Braun T D, Siegel H J, Beck N. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems [J]. Journal of Parallel and Distributed Computing, 2001, 61(1):810-837.
- [11] 邓树强,刘 波,庞雄文.改进的 Min-Min 算法研究[J]. 计算机与现代化,2011(6):1-4.
- [12] 罗宇平.基于 Min-Min 改进后的网格调度算法[J]. 微电子学与计算机,2009,26(3):86-88.
- [13] 薛庆吉,李春艳.一个改进的 Min-Min 网格任务调度算法[J]. 武汉理工大学学报,2009,31(3):379-390.
- [14] 张俊祥,冯金富,于心一.异构多处理器系统的混合任务调度算法[J]. 电光与控制,2011,18(12):1-5.
- [15] Panduranga R M V, Chandrashekar S K, Balakrishna R, et al. Development of Scheduler for Real Time and Embedded System Domain [C]//Proceedings of the 22nd International Conference on Advanced Information Networking and Applications-Workshops. Washington D. C., USA: IEEE Press, 2008:1-6.
- [16] Kinebuchi Y, Sugaya M, Oikawa S. Task Grain Scheduling for Hypervisor-based Embedded System [C]//Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications. Washington D. C., USA: IEEE Press, 2008:190-197.
- [17] 张 杰.最早截止期优先实时调度算法研究[D]. 武汉:华中科技大学,2009.
- [18] 王多强,鲁剑锋,李庆华.实时调度中基于多特征参数的任务优先级设计方法[J]. 计算机工程与科学, 2008,30(1):1-4.

编辑 金胡考

(上接第54页)

- [17] Han X D, Wang X B, Liu C. Retrieval Method for Traceability Links Between Source Code and Chinese Documentation [J]. Journal of Hefei University of Technology, 2010,33(2):188-187.
- [18] Nguyen A T, Nguyen T T, Alkofahi J, et al. A Topic-based Approach for Narrowing the Search Space of Buggy Files from a Bug Report [C]//Proceedings of IEEE/ACM International Conference on Automated Software Engineering. Washington D. C., USA: IEEE Press, 2011:263-272.
- [19] 尹丽丽,张丽萍,王春晖,等.基于潜在狄利克雷分配模

型预测克隆代码不一致变化的可能性[J]. 计算机应用,2014,34(6):1788-1791.

- [20] 张瑞霞,张丽萍,王春晖,等.基于主题建模技术的克隆群映射方法[J]. 计算机工程与设计,2015,36(6):1524-1529.
- [21] Roy C K, Cordy J R. NICAD: Accurate Detection of Near-miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization [C]//Proceedings of ICPC'08. Washington D. C., USA: IEEE Press, 2008: 172-181.

编辑 索书志