基于缺陷相似度与再分配图的软件缺陷分配方法

史高翔 赵逢禹
(上海理工大学光电信息与计算机工程学院 上海 200093)

摘 要 准确地将缺陷分配给最合适的修复者对大型软件项目的缺陷修复具有重要意义。当前缺陷自动分配技术的研究主要利用历史缺陷报告的描述信息、缺陷关联信息、历史分派信息等,但这些方法都没有将缺陷报告信息充分挖掘。提出在缺陷报告分配时将缺陷历史分派信息和缺陷文本相似信息相结合。首先根据缺陷历史分派信息生成再分配图;然后计算新缺陷报告与历史缺陷报告缺陷的文本相似度,找出相似度最高的前 K 个缺陷报告所对应的修复者;最后,根据这些修复者在再分配图中的依赖关系生成预测再分配路径。为了验证该方法的有效性,利用 Eclipse 和 Mozilla 的缺陷报告集进行实验,实验表明提出的方法在预测的准确度上明显优于其他方法。

关键词 历史缺陷报告,缺陷相似度,再分配图,预测再分配路径
中图法分类号 TP311.52 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.11.048

Software Defect Assignment Method Based on Defect Similarity and Tossing Graph

SHI Gao-xiang ZHAO Feng-yu
(School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China)

Abstract It has important significance to assign a bug report to the most suitable developer to repair in large software projects. At present, there are some methods using in automatic distribution for bug reports, such as utilizing description information of the historical bug reports, associating of bug reports, and historical information of bug report assignment. However, these methods do not fully exploit the information of defect report. This paper proposed to combine historical repair information with history assignment information. Firstly, a tossing graph is built up based on historical information of bug report assignment. Secondly, the similarity of new bug reports and historical bug reports is calculated, and the K final solvers who are corresponding to the K bug reports which have the highest similarity to the new bug report are selected. Finally, according to the dependent relations of these solvers in the tossing graph, a prediction assignment path is generated. To verify the validity of this method, we performed experiments with Eclipse and Mozilla defect report set. Experiments show that the method we proposed is superior to other methods in the accuracy of prediction.

Keywords Historical bug report, Defect similarity, Tossing graph, Predicted assignment path

及时识别和修正缺陷对于软件工程而言是十分重要的,为了处理大规模的缺陷,缺陷跟踪系统被广泛使用,如 Bugzilla。然而,目前大部分的缺陷却是手动分配给修复者,这是一个耗时且繁琐的工作,尤其对于大型的软件项目更是如此。如 Eclipse 和 Mozilla 项目每天都会接受数以百计的缺陷报告,这些缺陷报告中的每一个都必须分配给几千号开发人员中的一个。这并不是一个简单的任务,而且极易出错。

当一个缺陷报告被分配后,修复者可以将其指派给其他修复者,称之为缺陷报告再分配。Jeong 等人[1]研究指出 Eclipse 和 Mozilla 的约 450000 个缺陷报告中,分别有 44%和 37%的缺陷报告被再分配给其他修复者。这些导致再分配的一个主要原因是指派修复者出错,例如不属于其领域或专业知识不足以排除。统计发现,一个再分配将平均消耗 50 天左右的时间[2],因此,精确而准确的分配对项目进度和减少开发成本具有重要意义。多年来缺陷分配问题一直是软件工程研究领域的热点问题。

缺陷分配问题可以利用缺陷跟踪系统中的历史缺陷报告来解决。而对于每个已经修复的缺陷报告,其信息可以分为两个部分:文本信息和历史分配信息。其中文本信息有缺陷号、缺陷模块、版本号、描述信息、缺陷紧急程度等。而分配信息是指该缺陷报告的再分配路径,即该缺陷报告在修复者中的流动情况。

1 研究背景和相关工作

目前,关于缺陷分配的技术主要有机器学习、信息检索技术和缺陷再分配图等方法。很多缺陷报告分配策略使用机器学习的方法以达到文本分类的目的。通常,这些方法利用先前已经被分配的缺陷报告去训练分类器,然后利用训练的分类器去分配新的缺陷报告。如 Cubranic 和 Murphy[2]于 2004 首先提出了基于机器学习进行缺陷报告分配的方法,他们把

到稿日期:2015-11-01 返修日期:2016-01-26
史高翔(1990—),男,硕士,主要研究方向为软件工程、软件可靠性及缺陷分派,E-mail:752037951@qq.com;赵逢禹(1963—),男,博士,教授,主要研究方向为软件工程、软件可靠性及缺陷定位。

缺陷报告的分配视为一个有监督的学习，已修复的缺陷报告历史信息(如开发者和修复者)被作为训练数据以达到找到最适合的修复者的目的。通过实验验证，他们使用贝叶斯分类器分配了 15859 个缺陷报告，达到了 30%的准确率。随后，Anvik 等人[3]通过过滤外部噪音如删除重复的缺陷报告，并将标记不能修复的缺陷报告和不再继续参与项目的修复者排除在外，与先前的方法相比，去噪后有更高的准确率(已超过 50%)。Xuan 等人[4]首先提出了一种半监督的机器学习方法，他们将贝叶斯分类方法和最大期望(EM)算法相结合，为缺陷报告生成一个带权重的推荐修复者列表。随后，Podgurski 等人[5]也基于类似方法进行缺陷自动分类，但改善效果不够明显。

信息检索技术也被广泛用于缺陷分配，因为缺陷报告以文件的形式记录可能用于缺陷分配的信息。如 Matter 等人[6]提出了一种新颖的基于开发者源代码词频贡献率的缺陷分配方法。他们从缺陷报告中提取信息，找出关键词，并通过信息检索得出开发者在源代码中对这些关键词的贡献率，进而得出最佳推荐者，该方法不依赖先前缺陷的修复者，并且能将缺陷分配给合适的开发者，即使推荐的修复者之前从未修复过这种缺陷。

再分配图进行缺陷分配方法是一种全新的缺陷分配方法，如 Jeong 等人[1]利用已修复的缺陷报告中的再分配信息构造了缺陷再分配图，利用缺陷再分配图生成了新缺陷报告的分配路径。用 Mozilla 和 Eclipse 的 445000 个缺陷报告进行实验，这种模型减少了 72%的缺陷报告传递事件，并提高了 23%的分配精度。Bhattacharya 和 Neamtiu [7]扩展了 Jeong 的方法，在边和节点上添加了附件的属性，通过实验评估，这种方法减少了 86.3%的传递事件，缺陷分配的准确率达到了 83%。Chen 等人[8]提出利用文本信息和再分配信息进行自动缺陷分配，其再分配信息用于生成再分配图，文本信息被用于生成再分配子图时筛选相关修复者。但是该方法没有将文本信息用于再分配路径生成中，仅仅依靠再分配信息来决定再分配路径的生成，会导致再分配路径不够准确有效。

基于以上分析，本文将缺陷报告的再分配信息、缺陷文本相似度信息及修复人员特征信息三者相结合来实现缺陷报告的自动分配。历史分配信息用于再分配图的生成，再分配图反映了项目成员之间的投递依赖关系。缺陷文本相似度有两个作用：1)将缺陷文本相似度与修复人员特征信息一起用于再分配子图的生成。本文将缺陷文本相似度排名前 K 位的历史缺陷报告对应的修复者作为生成子图的保留节点，根据开发人员特征信息，删除不能快速准确修复缺陷的节点。2)将缺陷文本相似度用于再分配路径生成过程。在生成再分配路径时，缺陷文本相似度和传递频率会被综合考虑，从而决定路径的生成。

本文第 2 节介绍了再分配图生成及计算缺陷报告相似度的方法；第 3 节介绍了基于缺陷文本相似度和再分配图进行缺陷自动分配的过程和算法；第 4 节利用 Eclipse 和 Mozilla 项目的缺陷报告集对所提方法进行验证和分析，实验表明，所提方法确实可以更加有效地进行自动缺陷分配；最后描述了本文的研究成果和待改进部分。

2 缺陷再分配图与缺陷报告相似度

本文的缺陷再分配图基于 Jeong 等人[1]提出的再分配图的概念，具体阐述请见 2.1 节和 2.2 节。

2.1 缺陷再分配路径

软件在运行过程会源源不断地出现新的缺陷报告，这些缺陷报告会被返回给技术人员进行修复，这是软件不断优化的过程。被返回的缺陷报告首先分配给第一个修复者 d_1，若 d_1 由于各种原因不能解决该缺陷，那么该缺陷被分派给第二个修复者，以此类推，直到最终的修复者 d_f，将参与修复这一缺陷的修复者按时间先后排列起来，形成修复路径，也称之为缺陷再分配路径。

定义 1(缺陷再分配路径) 在缺陷修复过程中，缺陷在修复者之间的传递路径记为 $T=d_1\rightarrow d_2\rightarrow\cdots\rightarrow d_f$，称为缺陷再分配路径。$d_1$ 为第一个分配的修复者，d_f 为最终修复者。$|T|-1$ 称为再分配路径长度。相邻两个节点之间所花费的时间称为再分配时间。再分配路径中修复者集合 $R=\{d_1, d_2,\cdots,d_f\}$称为相关修复者。

可以对再分配路径进行分解以便获得其各种属性特征。假设存在一条再分配路径 $A\rightarrow B\rightarrow C\rightarrow D$，则 D 为该缺陷的修复者，$A\rightarrow B, B\rightarrow C, C\rightarrow D$ 为其实际路径模型。为了快速发现或预测能修复该缺陷的修复者，可以定义一种以目标为导向的模型，称为目标路径模型，如上面的再分配路径可得到目标路径模型 $A\rightarrow D, B\rightarrow D, C\rightarrow D$。多条路径的目标路径模型的集合称为目标路径集，其后的数字表示目标路径模型出现的次数，表 1 是再分配路径集合及其目标路径集。

表 1 再分配路径集合及其目标路径集

再分配路径	目标路径集
A→B→C→D→F	A→F(2),B→F(2)
B→D→E→F	C→F(2),D→F(2)
A→C→F	E→F(1),B→E(2)
B→C→D→E	C→E(1),D→E(2)
B→E	
D→E	

定义 2(传递频率) 目标路径集中，修复者 d 指向所有其他修复者 $d_1, d_2,\cdots, d_n$，分别为 $N_1, N_2,\cdots, N_n$ 次，则修复者 d 到其他修复者 d_i 的传递频率 t_i 如式(1)所示。

$$t_i=\frac{N_i}{\sum_{k=1}^{n}N_k} \tag{1}$$

基于表 1 的再分配路径集合及其目标路径集可得到传递频率，如表 2 所列。

表 2 基于目标路径集的传递频率

分配者	总次数	修复者	修复次数	传递频率
A	2	F	2	1
B	4	E	2	0.5
B	4	F	2	0.5
C	3	E	1	0.33
C	3	F	2	0.67
D	4	E	2	0.5
D	4	F	2	0.5
E	1	F	1	1

2.2 缺陷再分配图

定义 3(再分配图) 再分配图是一个有向图 $G=(R, \{E\})$，其中 R 是修复者集合，$\{E\}$是修复者之间的依赖传递关系，其元素可以用三元组$<d_i, d_j, t_{ij}>$表示，其中 t_{ij} 表示 d_i 到 d_j 的传递频率。

基于表 2 的目标路径集的传递频率可构造出如图 1 所示

的缺陷再分配图。图1中，修复者 C 在缺陷不能解决的情况下可以将 E 和 F 作为传递对象，显然 C 对 F 的依赖概率要明显大于 E，因此 C 在不能修复缺陷的情况下会考虑将缺陷投递给 F 进行修复。

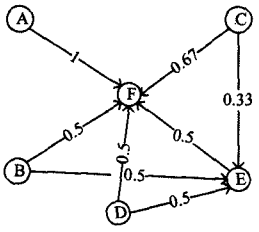

图1 缺陷再分配图

显然，再分配图反映了所有开发者之间的传递依赖关系，但对于具体的一个缺陷，其相关修复者是有限的，如果通过再分配图去寻找新缺陷报告的再分配路径会导致路径过长。若对于每一个新缺陷报告，在原有的再分配图上将不相关的修复者及其依赖关系删除，生成图 G'，在 G' 的基础上生成的再分配路径将更加简短和精确，G' 也称为再分配子图。本文3.3节将详细地描述再分配子图的生成方法。

2.3 缺陷文本相似度

2.3.1 向量空间模型

本文通过缺陷相似度找到相关的已修复的缺陷报告，因此，缺陷相似度的准确性十分重要。常用的文本相似性算法有向量空间模型(VSM)[9]、潜在语义索引(LSI)[10]、词频-逆向文件频率(TF-IDF)等。VSM 的向量余弦可作为缺陷文本相似度，简化了本文的模型。因此本文采用 VSM 作为搜索相似文本的方法。

在向量空间模型中，文本用 Text 表示，特征项(Term)是指出现在文档 Text 中且能够代表该文档内容的基本语言单位，本文采用语片为特征项来计算文本的相似度[10,11]。其中语片包含能反映文本局部语义片段的中心词语法结构。其全集构成完整的语义。

对于一个文本 Text，其由若干句子组成，用 $S_k(1\leqslant k\leqslant p)$ 表示构成文本的句子，则 $Text=S_1S_2\cdots S_3\cdots S_P$。句子可以形式化为 $S_k=C_1B_1C_2B_2\cdots C_nB_n$，其中 $C_i=\{$子句$\}$、$B_i=\{$标点符号$\}$。子句 $C_r=(1\leqslant r\leqslant n)$ 可以表示为若干词语(单词)的集合 $C_r=w_1\cdots w_i\cdots w_m$，用 $t_i=(w_a,w_b)$ 表示语片。其中，w_a，w_b 是文本中窗口单元内符合一定的相关术语和语法规则的两个词语，w_a 是表达语义的关键词，w_b 是与其有关的附属词，不区分他们的先后顺序。语片的集合 T 构成了文本的完整语义 $T=(t_1,\cdots,t_i,\cdots,t_y)$。

通常会将每个语片作为特征项，统计各语片的词频，并对其赋予一定的权重，简记为 $T(w_1,w_2,\cdots,w_i,\cdots,w_y)$。其中 w_i 是 t_i 的权重，$1<i<y$。一个特征项 t 在文档 d 中的权重可以用式(2)计算：

$$w_{i\in d}=(\lg(f_{td})+1)\times(\lg\frac{N}{1+n_t}) \tag{2}$$

其中，t 表示特征项，d 表示一个缺陷报告，f_{td} 是词语 t 出现在 d 中的次数，N 是缺陷报告总数，n_t 是含有词语 t 的缺陷报告总数。在向量空间模型中，两个缺陷报告 P_1 和 P_2 之间的相似度 $Sim(P_1,P_2)$ 常用向量之间夹角的余弦值表示，公式为：

$$Sim(P_1,P_2)=\frac{\sum_{i=1}^{n}w_{1i}w_{2i}}{\sqrt{\sum_{i=1}^{n}w_{1i}^2*\sum_{i=1}^{n}w_{2i}^2}} \tag{3}$$

2.3.2 用 VSM 寻找匹配的历史缺陷报告

通过向量空间模型，计算出新缺陷报告语历史缺陷报告库中已有缺陷报告的相似度，取相似度排名前 K 的历史缺陷报告由高到低进行排序，生成与新缺陷报告最相似的已修复历史缺陷报告的列表。该列表应为这样的二元数据对：$\langle bp_i,s_i\rangle,i=1,\cdots,k$。其中，$bp_i$ 是缺陷报告的 id，s_i 为其与新缺陷报告的相似度。

3 基于缺陷文本相似度和再分配图的软件缺陷自动分配方法

本文提出的软件缺陷自动分配方法包括4步，其结构如图2所示。

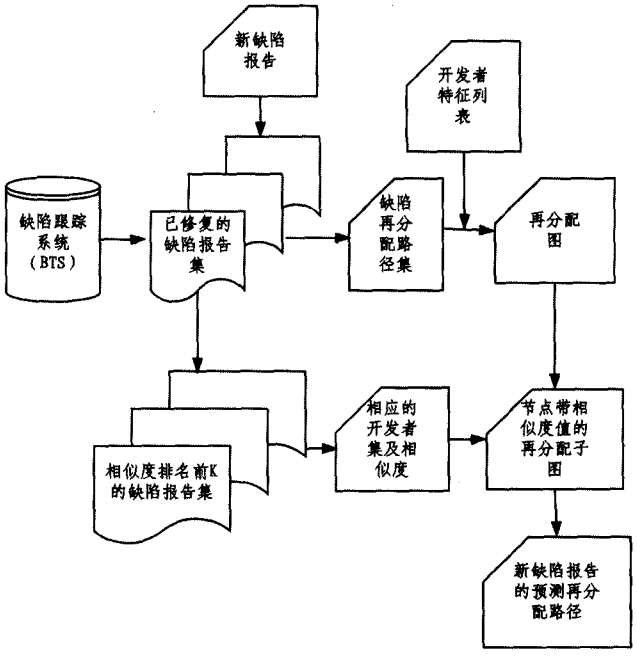

图2 基于缺陷相似度和再分配图的软件缺陷分配的过程

3.1 构造缺陷再分配图

大型的项目团队都有缺陷跟踪系统(BTS)，用于对缺陷报告的管理。本文的方法首先从 BTS 中取出已修复的缺陷报告作为训练集，提取缺陷报告上的再分配路径，然后根据再分配路径生成再分配图 G。在生成再分配图的过程中，会考虑将不能参与修复的修复者从再分配路径中删除，如该修复者已经离职或调任其他职位等。由此生成的再分配图中节点就代表了能真正参与修复的技术人员。

3.2 用 VSM 找出排名前 K 的历史缺陷报告

将新的缺陷报告与缺陷报告库中的历史缺陷报告进行相似性对比，取出相似度排名前 K 的缺陷报告，并记录其缺陷文本相似度和对应的修复者。生成三元数据对集 $Set=\{\langle bp_i,s_i,d_i\rangle,i=1,\cdots,k\}$，$bp_i$ 为历史缺陷报告的 id，s_i 为 bp_i 与新缺陷报告的相似度，d_i 是 bp_i 对应的修复者。K 的取值决定了预测再分配路径长度，若 K 太大，无关修复者会被放入，导致预测再分配路径增长，分配准确性下降；若 K 太小，会导致相关的修复者被拒绝，分配失败的概率增大。因此必须选择适当的 K 值。

3.3 生成再分配子图

首先，找到 Set 包含的修复者，通过搜索再分配图 G，保留这些修复者和连接它们的边，删除剩余的节点和边，则生成

了再分配子图 G_1。然后，将 *Set* 中三元组的 s_i 加入到子图 G_1 中对应的修复者 d_i 节点上，生成节点带相似度值的再分配子图 G'。算法 1 给出了获取带缺陷文本相似的再分配子图。

算法 1 获取带缺陷文本相似度的再分配子图

SubGraph(G,matchedBPList,dev_traits)

输入：G：再分配图

matchedBPList：匹配的缺陷报告列表

dev_traits：缺陷修复者特征列表

输出：再分配子图 G′

算法描述：

1. 将新缺陷报告与历史缺陷报告库中的缺陷报告进行相似度对比，找到排名前 K 的历史缺陷报告，记录三元数据集 Set={⟨bp_i, s_i, d_i⟩, i=1,…,K}。
2. 在再分配图 G 中，寻找 Set 包含的修复者，保留这些修复者和连接它们的边，删除其他修复者和边，生成再分配子图，记为 G_1。
3. 把匹配缺陷报告相似度放入 G_1 中对应的修复者的节点上，称其为节点带权重的再分配子图，记作 G′。

4、输出 G′，算法结束。

过程 $G \rightarrow G_1 \rightarrow G'$ 就是再分配子图的生成过程。如某一新缺陷报告的匹配缺陷报告有 4 个，其修复者分别是 B，D，E，F，其对应的缺陷报告相似度分别为 0.8，0.75，0.65，0.52，基于算法 1，生成节点带权重再分配子图如图 3 所示。

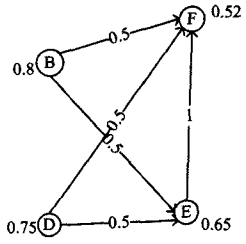

图 3 节点带相似度值的再分配子图

3.4 生成预测再分配路径

根据节点带相似度值的再分配子图，可以得到预测再分配路径。利用预测再分配路径可以对新缺陷报告进行分配。如某一新缺陷报告得到的预测再分配路径是 $d_1 \rightarrow d_2 \rightarrow d_3 \rightarrow d_4$，则首先将新缺陷报告分配给 d_1，若 d_1 能修复，结束；否则，分给 d_2，就这样一直下去。若到 d_4 仍然不能修复，则搜索失败。根据节点带相似度值的再分配子图 G' 生成预测再分配路径，首先找到最大相似度的节点对应的修复者 d_1 并将其作为预测再分配路径 *Path* 的第一个元素。然后循环下面过程：取出 *Path* 中的最后一个元素 e，若 e 在 G' 中不指向任何一个节点，则算法结束，返回路径；否则按 e 在 G' 中指向节点的相似度和传递频率乘积最大的节点作为下一个分配者。预测再分配路径生成过程如算法 2 所述。

算法 2 预测再分配路径生成算法

PredictedPath(G′)

输入：G′：再分配子图

输出：预测再分配路径 Path

算法描述：

1. 在图 G′中找到节点值（相似度）的最大的节点 d_1 作为第一个分配者，将其加入再分配路径 Path。
2. 从 Path 中取出最后一个修复者 d_i，在图 G′中找出 d_i 指向的 q 个节点。
3. 若 q=0，转步骤 5；否则，取 $\max_{j=1}^{q}\{w_{ij} * s_j\}$ 作为下一个分配者加入 Path 中，其中 w_{ij} 是 Path 最后一个节点的指向的第 j 个节点的传递概率，s_j 是指向第 j 个节点的值（相似度），j=1,…,q。
4. 重复步骤 2 和步骤 3。
5. 输出 Path，算法结束。

基于算法 2，以图 3 的节点带相似度的再分配组图作为算法输入，可得到预测再分配路径 $Path=\{B,E,F\}$。

4 实验

4.1 实验数据集

为了验证基于节点带相似度值的再分配图的软件缺陷自动分配方法的有效性，本文提取了 Eclipse 项目在 2005 年全年所提交的所有缺陷报告（共计 40237 个）和 Mozilla 项目 2004 年全年所提交的缺陷报告（共计 34423 个）作为实验数据。在实验过程中，将 Eclipse 的前 20119 个缺陷报告作为训练集，将后 20118 个缺陷报告作为测试用例；同样将 Mozilla 的前 17212 个缺陷报告作为训练集，将后 17211 个缺陷报告作为测试用例。

4.2 实验流程和分析

本文实验实施流程：1）利用 python 爬虫程序将 Eclipse 项目和 Mozilla 项目的缺陷报告爬入本地数据库。在本地数据库建立对应的表，每行存储一个缺陷报告。这样做的好处是对其的处理速度比处理文件的速度大有提升。2）根据训练缺陷报告生成再分配图，用邻接矩阵存储。3）对于每一个测试缺陷报告，首先用 VSM 将其与训练集缺陷报告进行相似度对比，取相似度最高的前 K 个缺陷报告，找出其相应的修复者。根据这些修复者，生成带相似度值的再分配子图，然后根据预测路径算法生成预测再分配路径。4）对预测再分配路径进行评估。在验证实验有效性之前，本文给出以下定义。

定义 4（平均再分配路径长度 *MLTP*） 对于 m 个给定的再分配路径 $\{T_i\}_{i=1,\cdots,m}$，$(\sum_{i=1}^{m}|T_i|)/m$ 为平均再分配路径长度。

定义 5（平均首次遇到修复者长度 *MLFF*） 对于测试集中一个缺陷报告 b，其修复者为 f。利用所提方法生成了再分配路径 T，路径中第一次出现 f 的位置下标被定义为首次遇到修复者长度，记为 $|Q|$，记平均首次遇到修复者长度为 *MLFF*。若搜索 T 时直到最后都没发现 f，则称搜索失败。

定义 6（搜索失败） 在测试缺陷报告生成的预测再分配路径中，如果没有找到该缺陷报告的真正修复者，则称搜索失败 *SF*。搜索失败的用例数与总测试用例数的占比称为搜索失败率（*SFR*）。

实验的任务是评估预测再分配路径的有效性，其中两个指标是十分重要的，即首次遇到修复者长度和搜索失败率。如一个测试用例的实际再分配路径为 $A \rightarrow B \rightarrow C \rightarrow D$，预测路径为 $A \rightarrow D \rightarrow B \rightarrow C$。由实际路径可知，$D$ 为其修复者。预测路径第二个分配给 D，其后的修复者 B 和 C 不需要再考虑。其首次遇到修复者路径长度为 1（下标从 0 开始）。若其预测路径为 $A \rightarrow B \rightarrow C$，即修复者没有出现在预测路径中，则称之为搜索失败。因此，本文选择将首次遇到修复者长度和搜索失败率作为评估的标准。

本文将 Eclipse 项目的 20118 个缺陷报告和 Mozilla 项目的 17212 个缺陷报告作为测试集，其实验结果统计如表 3 所列（K=8,9,10）。

分析数据发现：

(1)取 $K=9$ 时,Eclipse 的测试集经过一次再分配找到修复者的概率已达到 89%,经过两次及以上的再分配的概率为 8%。而 Mozilla 项目经过一次再分配找到修复者的概率已达到 86%,经过两次及两次以上再分配的概率为 9%。这比 Jeong 等人[1]经一次再分配找到修复者的概率 84%和 85%有了一定的提升。这一结果表明所提方法将 *MLFF* 控制在 0,1。

(2)预测路径首次遇到修复者的平均长度比实际路径平均再分配路径长度明显缩短,反映了总体的分配路径被缩短,再分配得到优化。

(3)测试集较少的 Mozilla 项目比 Eclipse 项目有更高的搜索失败率,表明训练的数据集多少影响对预测路径的平均长度和搜索失败率有一定的影响。

表 3 测试集的原始再分配路径和预测再分配路径数据统计

K	Project	缺陷报告中原始传递统计信息						预测再分配路径统计信息					搜索失败率(SFR)
		测试用例个数	MLTP	|T|=0	|T|=1	|T|=2	|T|>2	MLFF	|Q|=0	|Q|=1	|Q|=2	|Q|>2	
8	Eclipse	20118	2.8	40%	36%	18%	6%	1.2	50%	35%	4.2%	7%	3.8%
	Mozilla	17212	3.2	36%	26%	20%	18%	1.5	55%	22%	7%	9.8%	6.2%
9	Eclipse	20118	2.8	40%	36%	18%	6%	1.5	51%	38%	6%	2.1%	2.9%
	Mozilla	17212	3.2	36%	26%	20%	18%	1.9	58%	28%	5%	4%	5%
10	Eclipse	20118	2.8	40%	36%	18%	6%	2.1	46%	40%	6%	3.7%	2.3%
	Mozilla	17212	3.2	36%	26%	20%	18%	2.3	44%	38%	9%	5.6%	3.4%

在获取再分配子图时,选择相似度排名前 K 的缺陷报告的修复者作为再分配子图的节点。显然 K 的大小对首次遇到修复者长度(*MTFF*)和搜索失败有重大的影响,当 K 较大时,很多无关的修复者被放入子图中,导致首次遇到修复路径变长,同时搜索失败的概率会减小。当 K 较小时,许多相关的修复者被删除,这会导致首次遇到修复者长度缩短,但同时搜索失败的概率会增加。因此,较低的 *MTFF* 同时意味着较高的搜索失败率。通过设置不同的 K 值,得到的实验数据如图 4、图 5 所示。

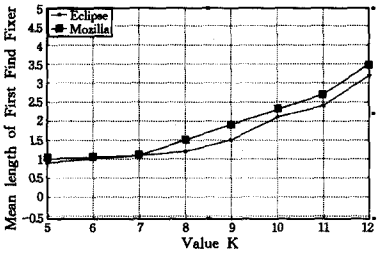

图 4 不同的 K 值对应的平均首次遇到修复者长度 *MLFF*

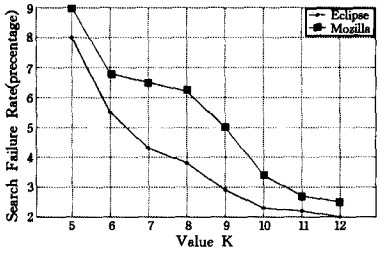

图 5 不同的 K 值对应的分配失败率

4.3 实验评估

为了说明本文方法的效率,将所提出的方法与 Jeong 等人[2]和 Chen 等人[8]的方法进行对比分析,得出的数据如表 4 所列。通过比较平均首次遇到修复者长度(*MLFF*)为 0 和小于或等于 1 的测试用例与总测试用例的占比,可以得到如下结论:

(1)当 *MLFF* 等于 0,即直接命中修复者,Jeong 采用了目标导向模型,而 Chen 和本文方法均采用选取最相似的缺陷文本。通过实验可以看到所提方法和 Chen 的方法在数据上很接近。

(2)*MLFF* 小于或等于 1,即将经过一次再分配的缺陷报告和直接命中的缺陷报告相加后所占占比。通过比较可知,当所提方法取 $K=9$ 时,Eclipse 项目的命中率已经达到 89%,而 Jeong 的方法只有 84%,Chen 的方法只有 87%。

(3)*MLFF* 小于或等于 2,所提方法已经能找出 90%以上的测试用例的修复者。

综上所述,可以判定所提方法已经在一定程度上实现了优化。

表 4 *MLFF* 小于或等于 0,1,2 次的概率

Method		Project	MLFF=0	MLFF≤1	MLFF≤2
Jeong		Eclipse	56%	84%	—
		Mozilla	63%	85%	—
Chen(Similarity Threshold is 0.65)		Eclipse	55%	87%	—
		Mozilla	61%	85%	—
our	K=8	Eclipse	50%	85%	89.2%
		Mozilla	55%	77%	84%
	K=9	Eclipse	51%	89%	95%
		Mozilla	58%	86%	91%
	K=10	Eclipse	46%	86%	92%
		Mozilla	44%	82%	91%

结束语 文本将缺陷报告的文本信息与分配活动信息相结合,充分挖掘了历史缺陷报告的有用信息。将缺陷文本相似度与开发人员修复依赖关系综合考虑作为再分配路径生成的依据。并结合开发人员自身特点,对生成预测再分配路径进行进一步优化,过滤干扰信息,从而使得再分配路径更加准确有效。实验证明,本文的方法相对其他未将缺陷文本相似度考虑进再分配的方法有一定的性能提升。但不足之处在于缺陷文本相似度算法参照了其他已实现方法,其相似度无法充分保证,当然对结果的预测也有一定的影响;本文实验部分数据量较小,生成的再分配图还不能代表真正的传递依赖关系;本文的实验仅仅对两个开源项目进行预测,对结论的印证还不充分,还需要将一些大型的开源项目进行实验从而进一步验证本文方法的有效性。

参 考 文 献

[1] Jeong G, Kim S, Zimmermann T. Improving bug triage with bugtossing graphs[C]//Proceedings of the 7th Joint Meeting of the European software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineer-

ing on European Software Engineering Conference and Foundations of Software Engineering Symposium. ACM, 2009: 111-120

[2] Cubranic D, Murphy G C. Automatic bug triage using text categorization[C]// Proceedings of the International Conference on Software Engineering & Knowledge Engineering. Alberta, 2004: 92-97

[3] Anvik J, Hiew L, Murphy G. Who should fix this bug? [C]// Proceedings of the 28th International Conference on Software Engineering. ACM, 2006: 361- 370

[4] Xuan J, Jiang H, Ren Z, et al. Automatic bug triage using semi-supervised text classification[C]// Proceedings of International Conference on Software Enginee-ring & Knowledge Engineering. Redwood City, 2010: 209-214

[5] Podgurski A, Leon D, Francis P. Automated support for classify software failure report [C] // Proceedings 25th International Conference on Software Engineering. IEEE, 2003: 465-475

[6] Matter D, Kuhn A, Nierstrasz O. Assigning bug reports using a vocabulary-based expertise model of developers[C]// 6th IEEE International Working Conference Mining Software Repositories (MSR'09). 2009: 131-140

[7] Bhattacharya P, Neamtiu I. Fine-grained incremental learning and multi-feature tossing graphs to improve bugtriaging[C]// Proceedings of the IEEE International Conference on Software Maintenance. Timisoara, 2010: 1-10

[8] Chen Li-guo, Wang Xiao-bo, Liu Chao. Improving Bug Assignment whit Bug Tossing Graphs and Bug Similarities [J]. Journal of Software, 2011, 6(3): 421-427

[9] Pang Jian-feng, Bu Dong-bo, Bai Suo. Automatic text classification system based on vector space model of the research and implement. [J]. Computer Application Resarch, 2001, 4(9): 23-26 (in Chinese)
庞尖峰，卜东波，白硕. 基于向量空间模型的文本自动分类系统的研究与实现[J]. 计算机应用研究，2001，4(9)：23-26

[10] Li Yuan-yuan, Ma Yong-qiang. The weight of text's trait word calculate method base on Latent semantic indexing[J]. Computer Application, 2008, 28(6): 1460-1464(in Chinese)
李媛媛，马永强. 基于潜在语义索引的文本特征词权重计算方法[J]. 计算机应用，2008，28(6)：1460-1464

[11] Pang Guo-qing. Using word piece as trait calculate text similarity in VSM[J]. Computer and Digital Engineering, 2007, 35(10): 24-36(in Chinese)
潘国清. VSM 中用语片为特征项计算文本相似度[J]. 计算机与数字工程，2007，35(10)：24-36

[12] Pang Guo-qing. An improved method and application about trait word extract [J]. Journal of Hunan College of Engineering, 2009, 19(2): 38-41(in Chinese)
潘国清. 一种向量空间模型种对特征项的改进方法及应用[J]. 湖南工程学院学报，2009，19(2)：38-41

(上接第 245 页)

[6] Becker J, Delfmann P, Knackstedt R. Reference Modeling [M]. Berlin: Springer, 2007: 27-58

[7] Weidlich M, Mending J, Weske M. Efficient consistency measurement based on behavioral profiles of process models [J]. IEEE Transactions on Software Engineering, 2011, 37(3): 410-429

[8] Weidlich M, Mending J. Perceived consistency between process models [J]. Information Systems, 2012, 37(2): 80-98

[9] Weidlich M, Polyvyanyy A, Mendling J, et al. Causal behavioural profiles-efficient computation, applications, and evaluation [J]. Fundamenta Informaticae, 2011, 113(3/4): 399-435

[10] Weidlich M, Polyvyanyy A, Mendling J, et al. Efficient computation of causal behavioural profiles using structural decomposition[M]// Applications and Theory of Petri Nets-Lecture Notes in Computer Science. Berlin: Springer, 2010: 63-83

[11] Polyvyanyy A, Weidlich M, Conforti R, et al. The 4C spectrum of fundamental behavioral relations for concurrent systems: Application and Theory of Petri Nets and Concurrency-Lecture Notes in Computer Science[C] // International Conference on Application and Theory of Petri Nets and Concurrency. Switzerland: Springer International Publishing, 2014: 210-232

[12] van der Aa H, Leopold H, Reijers H A. Detecting Inconsistencies between Process Models and Textual Descriptions: Business Process Management Business Process Management- Lecture Notes in Computer Science[C] // International Conference on Business Process Management. Switzerland: Springer International Publishing, 2015: 90-105

[13] Goltz U, Reisig W. Processes of place/transition-nets: Automata, Languages and Programming-Lecture Notes in Computer Science[D]. Berlin: Springer-Verlag, 1983: 264-277

[14] Jiang Chang-jun, Yan Chun-gang. Research on Process Characteristics of Synchronous Composition Nets [J]. Acta Electronica Sinica, 1997, 25(2): 57-60(In Chinese)(in Chinese)
蒋昌俊，闫春钢. 同步合成网的进程特性研究[J]. 电子学报，1997，25(2)：57-60

[15] Lin Chuang, Wei Ya-ya. Stochastic Process Algebras and Stochastic Petri Nets [J]. Journal of Software, 2002, 13(2): 203-213(in Chinese)
林闯，魏丫丫. 随机进程代数与随机 Petri 网[J]. 软件学报，2002，13(2)：203-213

[16] Wu Zhe-hui, Zhang Ji-jun. Process Grammar and Process Language of Petri Nets [J]. Computer Science, 2002, 29(12): 31-33 (in Chinese)
吴哲辉，张继军. Petri 网的进程文法和进程语言[J]. 计算机科学，2002，29(12)：31-33

[17] Martinik I. Modelling of Distributed Programming Systems with Using of Property-Preserving Petri Net Process Algebras and P/T Petri Net Processes[C]// 2013 Second International Conference on Informatics and Applications(ICIA). Lodz: IEEE, 2013: 258-263

[18] van Glabbeek R J, Goltz U, Schicke J W. Abstract Processes of Place/Transition Systems [J]. Information Processing Letters, 2011, 111(13): 626-633

[19] Murata, Tadao. Petri nets: Properties, analysis and applications [J]. Proceedings of the IEEE, 1989, 77(4): 541-580