基于排序学习算法的软件错误定位模型研究

何海江

(长沙学院数学与计算机科学系 长沙 410022)

【摘要】基于谱的错误定位(SBFL)是减小软件调试困难的有效方法。SBFL方法将运行特征组合成不同的错误定位模型，可这些模型受特定程序及测试集影响，性能并不稳定。基于此，提出一种排序学习方法LTR-sbfl，针对不同的程序及其测试集，能学习到相应的错误定位模型。LTR-sbfl首先收集已标注错误语句的程序旧版本，采用与SBFL相同的方法，获得程序语句的运行特征。接着，将错误行与正确行代码的运行特征两两相减，确定为训练数据的一个实例。最后，使用分类算法，从训练数据构造排序模型。在3个基准数据集的实验结果证实，LTR-sbfl的定位精度优于其他SBFL方法。

关 键 词 错误定位； 排序学习； 机器学习； 程序谱； 语句可疑度

中图分类号 TP311 **文献标志码** A doi:10.3969/j.issn.1001-0548.2017.03.016

Research of Software Fault Localization Based on Learning to Rank

HE Hai-jiang

(Department of Mathematics and Computer Science, Changsha University Changsha 410022)

Abstract Spectrum-based fault localization (SBFL) has been proven to be the promising technique during software debugging. SBFL combined various factors in the execution profile to form different fault localization models. However, these models are still agnostic to the particular program and dataset. In this paper we propose a learning to rank method referred to as LTR-sbfl which automatically constructs the SBFL model for each program. Like the existing SBFL methods, we first collect program execution traces and results of test cases. Next, it is directly mapped from the coverage difference in a pair of faulty statement and non-faulty statement to an instance in supervised learning. Finally, the conventional classification algorithm is applied to learn a ranking model for a program based on the training data. The proposed method on three benchmark datasets is evaluated through experiment. Results show that LTR-sbfl can outperform other SBFL approaches in the precision of fault localization.

Key words fault localization; learning to rank; machine learning; program spectrum; suspiciousness of statement

调试是软件工程师的一项重要技能，甚至可以说，一个软件项目的成功与否，很大程度上取决于程序员的调试水平。在软件生命周期的开发阶段和维护阶段都有大量的调试活动。调试过程中，需要不断分析变量、运算符、函数、语句等之间的依赖关系，以便定位错误代码。

为解决错误定位的难题，人们提出了许多有效的方法，自动搜索程序故障位置，将搜索结果呈现出来，由程序员决定如何修改代码。这些方法包括两大类：静态方法和动态方法。静态方法不运行程序，依据开发语言的语法语义分析源代码，但难以发现大多数逻辑错误。动态方法则运行程序，依据测试用例执行结果和代码运行轨迹来分析源代码。比较而言，动态方法更有效，也是软件错误定位领域的研究重点。动态的错误定位方法大致分为4类：1) 程序切片方法[1]和谓词方法[2]。设定感兴趣的变量或谓词，检查它们在程序运行过程中值的变化情况，从而缩小错误代码的搜索范围。2) 基于程序谱的方法[3-4]，简称基于谱的方法。程序谱是指测试用例执行结果和程序实体覆盖信息。程序实体可以是语句、分支、代码片段，也可以是谓词、函数、类等。3) 基于模型的方法[5-6]。假定程序实体的失效会遵循一个特定的模型，比如统计学模型、功能依赖模型及时间谱模型等。4) 数据挖掘方法[7]。用数

收稿日期：2015－08－10；修回日期：2016－05－02
基金项目：国家自然科学基金(61100139)；湖南省科技计划项目(2015GK3071)
作者简介：何海江(1970－)，男，副教授，主要从事机器学习、软件测试的研究.

据挖掘算法揭示程序实体引起运行故障的特征模式。

所有方法中，基于谱的错误定位(spectrum-based fault localization, SBFL)最受关注。SBFL只收集少量运行特征，计算也简单，每执行一次新的测试，很容易重新计算所有语句的错误可疑度，故而适应于大规模软件。人们提出许多SBFL方法，将4个运行特征组合成不同的语句可疑度计算公式。但这些方法存在缺陷：1) 对所有程序采用固定的可疑度公式。而特定的程序，受软件规模、测试套件、编程语言、程序员开发水平等因素的影响，可能出错方式与其他程序迥异。2) 若调试过程还收集到其他有用信息，很难集成到现有公式中。

为克服SBFL的两个缺点，提出一种排序学习的SBFL(learning to rank for SBFL, LTR-sbfl)，将错误定位问题转换为信息检索问题，用排序学习算法构造可疑度计算公式。LTR-sbfl需要收集程序的旧版本，将之作为训练数据。针对每一个程序，都能学习到独特而合身的可疑度计算公式。在Siemens程序包、space和grep这3个基准数据集上证实了新方法的有效性，LTR-sbfl比现有的SBFL方法定位性能更佳，还具有学习速度快、容易集成语句任意特征等优势。

1 基于谱的错误定位方法

SBFL方法只需收集两类信息：1) 测试用例的执行结果：成功或失败。2) 程序运行时，语句的覆盖程度，最为常见的是布尔值，即语句被覆盖或未被覆盖。SBFL方法可用于所有程序实体，不失一般性，本文用语句作为研究对象。形式上，每条语句只用少量运行特征，可用四元组<a_{ep},a_{ef},a_{np},a_{nf}>表示。SBFL方法计算可执行语句的4个特征，a_{ep}和a_{ef}表示测试用例执行时覆盖该语句，且程序运行结果分别为成功和失败；a_{np}和a_{nf}表示测试用例执行时未覆盖该语句，且程序运行结果分别为成功和失败。显然，每条语句的四元组元素数值之和等于测试用例数目。

a_{ep}越大，语句包含错误的可能性越小；a_{ef}越大，语句包含错误的可能性越大。相比而言，a_{np}和a_{nf}的大小与错误可能性关联程度小一些。但仍然可以推断：a_{np}越大，语句包含错误的可能性越大；a_{nf}越大，语句包含错误的可能性越小。SBFL方法就是基于以上朴素想法而提出，构造一个映射函数，将4个特征映射为一个实数值。由4个特征值计算每条语句的错误可疑度，可疑度大的语句，包含错误代码的概率较大。在SBFL研究社区，Exam值是最为常见的评估测度，当发现第一条错误语句时，所需要检查代码占整个可执行代码的百分比就定义为Exam值。程序员按照可疑度顺序来检查代码。当语句的可疑度值相同时，有3种策略决定检查次序。Best策略优先检查错误语句，Worst策略最后检查错误语句，这两种策略都是假设情况，现实中简易可行的策略是SOS(statement order based)[8]，程序员按照语句的顺序来检查代码。

NaishO[4]、NaishOp[4]、Kulczynski2[4]、Wong1[3]、Wong3[3]、Jaccard[9]、Tarantula[10]、Ochiai[4]和SIQ[11]是9个代表性的映射函数。以Exam值为评估测度，当程序只有一条错误语句时，文献[12]从理论上证明：NashiO与NashiOp等价，NashiOp> Kulczyuskiz> Ochiai> Jaccard> Tarantula，NashiOp>Wong3，此处>表示优于。而NashiOp与Wong1无法比较。各种SBFL方法无外乎将语句的4个特征组合成不同形式，但组合形式无穷。

2 基于排序学习的软件错误定位

在信息检索系统，用户提交查询要求后，系统按照相关度从高到低返回一系列对象(文档、图片、视频等)。可以将软件错误定位问题看成信息检索问题。调试过程中，程序员要求查询错误语句，以行为单位，错误定位系统依可疑度从高到低返回代码，供程序员判断。

2.1 排序学习算法

信息检索的相关度计算过程中，构造排序模型是关键环节。以$\boldsymbol{x}_i$作为第i个对象d_i的特征向量。以$\boldsymbol{w}$为排序模型的特征权向量，排序学习算法的目标是找到合适的$\boldsymbol{w}$，使得任意两个对象d_i和d_j，当d_i比d_j关联度高时，有$\boldsymbol{w}^* \phi(\boldsymbol{x}_i) > \boldsymbol{w}^* \phi(\boldsymbol{x}_j)$。*是内积运算符，$\phi$是特征映射函数，可以是线性的，也可以是非线性的核函数。若排序模型是线性模型，以$\boldsymbol{w}^*\boldsymbol{x}_i$作为关联度计算公式，判断对象的关联度高低。

从训练样本划分，排序学习大致有3类方式：单样本、样本对和多样本[13]。其中样本对方式将损失函数建立在成对的对象上，将相关度不同的对象样本两两组合成一个实例。已有的分类算法都可加入样本对策略，用于解决排序问题。在学习阶段，组织训练数据时，当d_i排在d_j前，可构造样本(d_i–d_j, +1)；反之，则构造为(d_i–d_j,–1)。学习到分类模型后，若要对d_i、d_j排序，只需计算d_i–d_j的类别，若属于+1类，则d_i排在d_j前；否则，排在d_j之后。

2.2　LTR-sbfl的优化目标

软件错误定位问题的语句错误可疑度对应信息检索问题的对象关联度。以$\boldsymbol{x}_i$作为程序第i条可执行语句的特征向量，以线性模型求解软件错误定位问题，$\boldsymbol{w}_i$是$\boldsymbol{x}_i$的权重。假设程序有n条可执行语句，其中q条语句包含错误代码，语句特征为m维。U_{OK}和U_{Fault}分别表示正确语句集合和错误语句集合，n-q和q是集合U_{OK}和U_{Fault}的元素个数。程序语句d_i和d_j的特征向量用$\boldsymbol{x}_i$和$\boldsymbol{x}_j$表示。基于排序学习的软件错误定位算法LTR-sbfl的优化目标为问题：

$$\min_w \frac{1}{2c}w^2+\sum \xi_{i,j}^2 \tag{1}$$

使得$\forall(d_i\in U_{\mathrm{Fault}},d_j\in U_{\mathrm{OK}})$:

$$\sum_{t=1}^{m}w_t x_{i,t}\geqslant\sum_{t=1}^{m}w_t x_{j,t}+1-\xi_{i,j}$$
$$\forall i,j:\xi_{i,j}\geqslant 0 \qquad i=1,2,\cdots,q$$
$$j=1,2,\cdots,n-q \tag{2}$$

式中，$\xi_{i,j}$是约束条件成立的误差标量，使得当d_i有错，d_j无错时，保证$\boldsymbol{w}^*\boldsymbol{x}_i\geqslant\boldsymbol{w}^*\boldsymbol{x}_j$；$C$是误差和权向量之间的平衡因子；显然有$q(n-q)$个$\xi_{i,j}$。

引入加号函数$(1-y)_+=\max(0,1-y)$，可将上述问题变为无约束问题。加号函数不可微，用分段多项式光滑函数近似$(1-y)_+$，结合牛顿法和一维精确搜索算法可求解该无约束问题[14]。

2.3　排序的软件错误定位模型

当用排序学习算法解决软件错误定位问题，首先确定描述程序语句的特征。程序谱、切片、定义-使用对等，无论是单独，还是混合使用，都可表达$\boldsymbol{x}_i$。接下来的学习过程中，针对程序的某个旧版本，任意故障语句和正确语句两两组合为式(2)的一个不等式；所有程序旧版本形成的不等式构成式(2)。求解式(1)和式(2)后，得到软件错误定位的排序模型$\boldsymbol{w}$，与特征$\boldsymbol{x}$组成动态变化的可疑度计算公式。当定位故障语句时，用可疑度公式计算所有可执行语句的错误可疑度，将语句按可疑度大小排序。定义程序第i条可执行语句的可疑度$\mathrm{Susp}_i=\boldsymbol{w}^*\boldsymbol{x}_i$，语句的Susp越大，就越被怀疑为错误代码。

3　实验结果及分析

本节报告LTR-sbfl与前述9种SBFL技术的实验比较结果。所有实验调用MinGW5.1的gcc和gcov编译程序和收集覆盖信息。

3.1　实验数据集

Siemens程序包很受欢迎，成为软件错误定位领域的基准数据集。其共7个程序：print_tokens和print_tokens2都是词法分析器，replace执行模式匹配及替换，schedule和schedule2用于优先级调度，而tcas是一个避免高空冲突的高程分隔系统，tot_info计算给定数据的统计信息测度。另外，本文还用了程序space和gzip作为实验数据集，比较各种算法的优劣。这些数据集均从SIR软件项目基础资源库(http://sir.unl.edu/portal /index.html)下载。

原数据集的replace、schedule2和space有些缺陷版本或者无失败测试用例，或者无法记录覆盖信息，故而排除在外。SIR软件项目基础资源库为每个程序提供了多种测试覆盖类型，选择universe完成所有实验。

3.2　训练LTR-sbfl的错误定位模型

在LTR-sbfl中，表示可执行语句的特征可以任意设置，个数也不受限制。为了与其他SBFL方法公平比较，使用5个只与四元组$<a_{\mathrm{ep}},a_{\mathrm{ef}},a_{\mathrm{np}},a_{\mathrm{nf}}>$有关的特征：$a_{\mathrm{ef}}/\mathrm{AF}$，$(a_{\mathrm{ef}}-a_{\mathrm{nf}})/(2*\mathrm{AF})$，$a_{\mathrm{nf}}/(\mathrm{AF}+a_{\mathrm{ep}})$，$a_{\mathrm{np}}/(a_{\mathrm{np}}+a_{\mathrm{nf}})$，$a_{\mathrm{ep}}/\mathrm{AP}$，其中$\mathrm{AF}=a_{\mathrm{ef}}+a_{\mathrm{nf}}$，$\mathrm{AP}=a_{\mathrm{ep}}+a_{\mathrm{np}}$。显然，第2个特征的值在实数区间[–0.5,0.5]，其他特征的值处于[0,1]。当错误语句与正确语句特征值相减，实例特征值在实数区间[–1,+1]。

使用四折交叉验证的策略完成所有实验。针对应用程序，错误版本尽量平均分成4部分：训练集2份，验证集和测试集各1份。对每一折，令算法的平衡因子$C=\{10^t,t=-5,-4,\cdots,2\}$，在训练集上使用LTR-sbfl学习错误定位模型，再计算其在验证集上的Exam值。将出现8个Exam值，取对应最小Exam的C为本折最优平衡因子，从而确定本折的错误定位模型。最后使用学习到的错误定位模型，计算测试集上的Exam值。被测程序的最终评价性能等于四折的平均Exam值。

3.3　实验结果及分析

在比较不同技术时，要分别考虑3种策略下的性能。在调试软件的实际过程中，SOS可用，而Best和Worst不可用，因此把比较重点放在SOS策略。Siemens程序包共有7个程序，除非特别声明，Siemens上的实验数据是指这7个程序的平均值。

表1、表2和表3是10种方法分别在Siemens、space和gzip上的Exam值比较。和后文其他表一样，表现最好的数据用黑体字标示，所有数据省略了百分符号。

表1 Siemens程序包上的Exam值比较

程序	NaishO	NaishOp	Kulczynski2	Wong1	Wong3	Jaccard	Tarantula	Ochiai	SIQ	LTR-sbfl
Best	11.52	11.73	12.02	**1.87**	12.12	19.23	19.85	15.85	15.74	9.81
Worst	18.53	18.17	18.45	53.75	18.55	25.49	26.11	22.28	22.65	**16.24**
SOS	14.20	14.26	14.54	26.38	14.65	21.70	22.31	18.37	18.67	**12.33**

表2 程序space上的Exam值比较

程序	NaishO	NaishOp	Kulczynski2	Wong1	Wong3	Jaccard	Tarantula	Ochiai	SIQ	LTR-sbfl
Best	3.69	4.14	2.38	3.21	4.11	2.11	3.46	**1.40**	3.95	3.18
Worst	14.09	4.84	3.08	28.78	4.81	2.80	4.22	**2.09**	4.63	3.64
SOS	8.85	4.56	2.80	20.97	4.53	2.52	3.89	**1.81**	4.37	3.43

如果程序只有一条错误语句，每个失败的测试用例执行时必然覆盖该语句，它的a_{ef}一定最大。Wong1只与a_{ef}有关，在那些只有一条错误语句的程序上，依Best策略，Wong1一定优于其他技术。在Siemens和gzip上，Wong1 Best都显著优于其他技术的Best策略。但是，当失败测试用例特别少时，许多正确语句会和错误语句一同被这些失败测试用例所覆盖。依Worst策略，在检查到错误语句时，要先检查那些与其可疑度相同的正确语句，这也就是Wong1 Worst表现特别差的原因。从表1、表2和表3可看出，总体上Wong1表现并不好，特别是SOS策略，Wong1最差。针对Wong1的缺陷，Wong3和NaishOp做了一些小的调整，前者增加a_{ep}的影响，而后者则加入a_{ep}和a_{np}，它们的表现都还不错。使用关于a_{nf}的阈值后，NaishO剔除掉那些失败测试用例未能覆盖的语句，在Siemens和gzip上表现不错，在space则较为糟糕。与NaishO的结果相反，Tarantula在space上还算过得去，在Siemens和gzip则很差劲。许多研究者喜欢拿Tarantula作为错误代码定位的基准技术，并非好的选择。Ochiai尽管在Siemens和gzip表现不佳，在space上却非常突出。文献[9]在比较40种SBFL技术后，发现Ochiai在许多程序上表现优异。

LTR-sbfl不仅考虑了四元组各元素的影响，还针对不同程序学习与其相适用的错误定位模型。在Siemens上，LTR-sbfl明显优于其他技术。表4是LTR-sbfl和9种SBFL方法在Siemens7种程序的表现，各方法皆使用SOS策略。尽管在每一个程序，LTR-sbfl都非最佳，但都与最佳接近，整体上就显示出优势。

表3 程序gzip上的Exam值比较

程序	NaishO	NaishOp	Kulczynski2	Wong1	Wong3	Jaccard	Tarantula	Ochiai	SIQ	LTR-sbfl
Best	1.84	1.84	3.03	**0.10**	1.38	3.14	4.48	3.03	3.53	1.84
Worst	**5.16**	**5.16**	6.35	14.10	7.09	6.46	12.58	6.35	7.03	**5.16**
SOS	**4.06**	**4.06**	5.24	9.90	4.26	5.35	8.86	5.24	5.91	**4.06**

表4 7种程序上10种方法SOS策略的Exam值比较

程序	NaishO	NaishOp	Kulczynski2	Wong1	Wong3	Jaccard	Tarantula	Ochiai	SIQ	LTR-sbfl
schedule2	41.48	41.48	41.56	37.67	41.57	55.63	55.63	48.89	**25.80**	26.74
schedule	11.08	11.44	11.44	34.24	11.44	5.43	5.67	**5.28**	26.97	6.77
print_tokens2	**1.80**	**1.80**	2.56	31.46	1.90	13.36	13.77	9.25	19.59	2.82
print_tokens	**9.81**	**9.81**	**9.81**	26.31	**9.81**	24.03	25.63	17.79	10.54	14.29
tot_info	**10.45**	**10.45**	10.94	20.15	10.66	22.20	23.84	18.71	16.36	10.73
replace	**4.46**	**4.46**	5.00	12.82	6.05	8.10	8.45	6.10	7.26	4.58
tcas	**20.34**	**20.34**	20.45	21.98	21.09	23.12	23.20	22.55	24.17	20.41

在space上，LTR-sbfl比不上Ochiai，也弱于Kulczynski2和Jaccard。主要原因在于space的V3，LTR-sbfl在V3表现特别差，其SOS的Exam值达到了61.9。V3与其他版本具有不同的特征权向量，第二个特征尤其明显。如果不考虑V3，则space上LTR-sbfl表现最优。在gzip上，LTR-sbfl和NaishO、NaishOp性能相同，相比其他技术有明显的优势。程序gzip大多数版本的错误语句少，未覆盖错误语句的失败测试用例为零，NaishO和NaishOp自然能够脱颖而出，而LTR-sbfl依赖其学习能力亦达到最佳效果。从3个表综合来看，以Exam值为评价指标的话，无疑LTR-sbfl是最为突出的方法。

当程序较大，而错误语句的可疑度靠后，程序员检查较多语句后，仍未能定位到错误语句，会产

生厌烦情绪。通常，SBFL研究社区会限制检查比例P_{check}(已检查语句数占程序总语句数的比率)，计算出$P_{bugVers}$(能发现错误版本数占程序总错误版本数的比率)，来比较各类技术的优劣。

图1是Siemens上比较各类方法的性能，P_{check}从1%～20%，步长1%。图中的所有技术都基于SOS策略。由于Wong1的效果特别差，未录入其数据。在计算Siemens程序包的$P_{bugVers}$时，并非7个程序的平均值，而以7个程序的错误版本数之和作为分母，分子也就是能发现错误的版本数之和。篇幅所限，省略了space和gzip的比较图。

在Siemens程序包，LTR-sbfl、NaishO、NaishOp、Kulczynski2和Wong3明显优于其他技术，而在这5种技术中，又以LTR-sbfl最为突出。在P_{check}=20%处，它们的$P_{bugVers}$分别为76.2%、74.6%、74.6%、73.8%和73.8%。实际上，从20%开始，当P_{check}逐步增加到50%，LTR-sbfl也始终表现最好。在程序space中，Ochiai最优，LTR-sbfl、Kulczynski2和Jaccard稍次之，Tarantula再次之，其他技术则明显处于劣势。在程序gzip中，LTR-sbfl、NaishO、NaishOp和Wong3明显优于其他技术。综合看，以$P_{bugVers}$为评价指标，无疑LTR-sbfl是最有竞争力的方法。

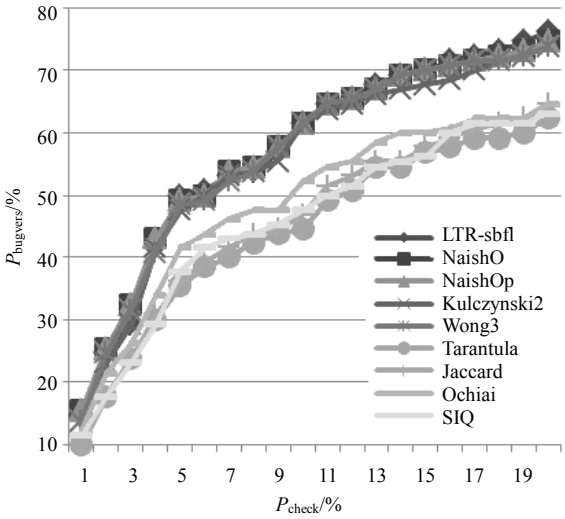

图1　Siemens程序包的$P_{bugVers}$比较

4　时间复杂度和空间复杂度

LTR-sbfl不仅要获取特征数据和计算语句可疑度，与其他SBFL技术不同的是，它还要从程序的旧版本学习到错误定位模型，用以确定后续版本的语句可疑度计算公式。其他SBFL技术需要的时间和空间，LTR-sbfl同样不可缺少。此处，本文只讨论LTR-sbfl额外增加的时间和空间。

在空间复杂度方面，LTR-sbfl要求以文件形式保存程序旧版本所有可执行语句的特征，文件数据的空间复杂度为$O(n*m*$nVers)，每一个版本的可执行语句数用n表示，m是语句的特征维数，nVers是程序旧版本数目。另外在LTR-sbfl的训练过程，还要求将这些特征数据从文件加载至内存，并两两组合成学习器的实例，内存数据的空间复杂度为$O((n-q)*q*m*$nVers)，其中q是错误语句数。以space为例，若5个特征全部用8字节的double型存放，则space的每个版本大约需要8*5*3600≈144 KB来存放文件。若内存充足，可一次读入所有特征数据，space的32个版本共需10.2 MB；若内存有限，则训练集、验证集和测试集的特征数据分别载入，此时需3.4 MB内存空间。

时间复杂度方面。LTR-sbfl在训练过程主要包括I/O时间和学习时间。I/O时间主要是将特征数据从文件读入到内存的时间，它与文件大小成正比。一般来说，对小程序而言，I/O时间超过学习时间；大程序上，学习时间超过I/O时间。理论上，依据实现原理，LTR-sbfl的学习时间复杂度为$O((m+m^2/2)*(n-q)*q*$tVers$+m^3/6)$，tVers是训练集包含的程序旧版本数目，本文实验里通常为nVers的一半。表5是4个程序的训练情况，n和q取每一版本的平均值，tVers和训练时间则是每一折的平均值。每一折的训练时间包括：训练集上学习8个平衡因子C的时间，验证集上计算8个C对应的Exam值。实验数据集其他程序的训练时间皆不到1 ms，故而省略。

表5　4程序上LTR-sbfl的训练时间比较

程序	n	q	tVers	训练时间/s
tot_info	122	1.1	11.5	0.019 5
replace	244	2.3	15.5	0.063 5
gzip	1 731	1.9	8	0.168
space	3 656	2.2	16	1.329 8

从表5可看出，LTR-sbfl的训练时间非常短，即算是space这样规模的程序，训练集版本数非常多，要获取最优定位模型的训练时间不到2 s，完全能达到在线训练要求。实际上，还有许多手段优化LTR-sbfl，这个时间还可以大幅缩短。

5　结 束 语

本文提出一种基于排序学习和程序谱的软件错误定位方法LTR-sbfl，和SBFL技术一样，只收集程序实体覆盖信息和测试用例执行结果。不同的是，SBFL对所有程序采用固定的可疑度计算公式；而

LTR-sbfl针对特定程序，在收集的数据上学习到与之适应的可疑度公式。LTR-sbfl将测试用例执行时错误语句、正确语句的覆盖特征两两组合，构造为训练数据的实例，再用排序支持向量机求解错误定位模型。在Siemens、space和gzip3个基准数据集的实验结果证实，与SBFL社区常用的9种技术相比，LTR-sbfl表现更好一些。更值得称道的是，在LTR-sbfl中很容易加入程序的其他特征，以便尽快找到错误代码。算法的训练速度非常快，方便集成到开发工具进行在线学习，帮助程序员实时调试程序。

LTR-sbfl打开了SBFL研究的新窗口，有许多新内容值得研究。1) 在算法中，可以加入编程语言、程序员出错习惯、程序静态结构信息等特征。2) 进行跨工程的模型研究[15]，将一个程序学习到的可疑度公式应用到其他程序，可能也会产生不错的效果。3) 表征程序语句的特征维数并不多，相对线性方法，核方法[16]或许是更好的选择。以后可在这3方面进一步研究。

本文研究工作还得到长沙市科技计划项目(K1509011-11)的资助，在此表示感谢。

参 考 文 献

[1] ZHANG Xiang-yu, GUPTA N, GUPTA R. A study of effectiveness of dynamic slicing in locating real faults[J]. Empirical Software Engineering, 2007, 12(2): 143-160.

[2] WANG Xiao-yan, LIU Yong-mei. Automated fault localization via hierarchical multiple predicate switching[J].Journal of Systems and Software, 2015,104: 69-81.

[3] WONG W E, DEBROY V, CHOI B. A family of code coverage-based heuristics for effective fault localization[J]. Journal of Systems and Software, 2010, 83(2): 188-208.

[4] NAISH L, LEE H J, RAMAMOHANARAO K. A model for spectra-based software diagnosis[J]. ACM Trans on Software Engineering and Methodology, 2011, 20(3): 11.

[5] LIU Chao, FEI Long, YAN Xi-feng, et al. Statistical debugging: a hypothesis testing-based[J]. IEEE Trans on Software Engineering, 2006, 32(10): 831-848.

[6] 郝鹏, 郑征, 张震宇, 等. 基于谓词执行信息分析的自适应缺陷定位算法[J]. 计算机学报, 2014, 37(3): 500-511.
HAO Peng, ZHENG Zheng, ZHANG Zhen-yu, et al. Self-adaptive fault localization algorithm based on predicate execution information analysis[J]. Chinese Journal of Computers, 2014, 37(3): 500-511.

[7] ZHANG Sai, ZHANG C. Software bug localization with Markov logic[C]//36th Int Conf on Software Engineering NIER Track. Piscataway, NJ: IEEE, 2014.

[8] XU Xiao-feng, DEBROY V, WONG W E, et al. Ties within fault localization rankings: Exposing and addressing the problem[J]. Int Journal of Software Engineering and Knowledge Engineering, 2011, 21(6): 803-827.

[9] LUCIA, LO D, JIANG Ling-xiao, et al. Extended comprehensive study of association measures for fault localization[J]. Journal of Software Evolution and Process, 2014, 26(4): 172-219.

[10] JONES J A, HARROLD M J. Empirical evaluation of the tarantula automatic fault-localization technique[C]//20th IEEE/ACM Int Conf on Automated Software Engineering. New York: ACM , 2005: 273-282.

[11] 丁晖, 陈林, 钱巨, 等. 一种基于信息量的缺陷定位方法[J]. 软件学报, 2013, 24(7): 1484-1494.
DING Hui, CHEN Lin, QIAN Ju, et al. Fault localization method using information quantity[J]. Journal of Software, 2013, 24(7): 1484-1494.

[12] XIE Xiao-yuan, CHEN T Y, KUO F, et al. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization[J]. ACM Trans on Software Engineering and Methodology, 2013, 22(4): 31.

[13] LIU Tie-yan. Learning to rank for information retrieval[J]. Foundation and Trends on Information Retrieval, 2009, 3(3): 225-331.

[14] 何海江, 陈姝. 由排序支持向量机抽取博客文章的摘要[J]. 电子科技大学学报, 2010, 39(4): 593-597.
HE Hai-jiang, CHEN Shu. Extraction of blog summarization by using ranking SVM[J]. Journal of University of Electronic Science and Technology of China, 2010, 39(4): 593-597.

[15] MA Ying, LUO Guang-chun, ZENG Xue, et al. Transfer learning for cross-company software defect prediction[J]. Information and Software Technology, 2012, 54(3): 248-256.

[16] REN Jin-sheng, QIN Ke, MA Ying, et al. On software defect prediction using machine learning[J]. Journal of Applied Mathematics, 2014(3): 201-211.

编 辑 蒋 晓