计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
50(3): 619-625, 2013

神经网络在软件多故障定位中的应用研究

何加浪[1]　　张　宏[2]
[1](中国电子科技集团第三十八研究所　合肥　230088)
[2](南京理工大学计算机科学与技术学院　南京　210094)
(jialanghe@126.com)

Application of Artificial Neural Network in Software Multi-Faults Location

He Jialang[1] and Zhang Hong[2]
[1](*East China Research Institute of Electronic Engineering*, *Hefei* 230088)
[2](*Institute of Computer Science and Technology*, *Nanjing University of Science and Technology*, *Nanjing* 210094)

Abstract　There is no bug-free program because of the complexity of software. It is always challenging for programmers to effectively and efficiently debug program and remove bugs. Software fault location is one of the most expensive activities in program debugging. So there is a high requirement for automatic fault localization techniques that can guide programmers to the locations of faults with minimal or no human intervention. Various techniques have been proposed to meet this requirement. However, the interactions between multi-faults which have not been fully considered in previous studies make the fault location more complicated. In order to solve this problem, a novel neural-network-based multi-faults location model is proposed in this paper. By fault relation analysis, the model calculates the support degree of the input for each fault. And then it learns the relationship between the faults and the candidate locations of faults using the constructed neural network. Constructing an ideal input as the input of learned neural network, the model can calculate the suspicious degree of each candidate location of fault, then obtain the sequence sorting by the suspicious degree, and complete the task of multi-faults location. Experimental results show that compared with traditional methods, the proposed method has strong ability to distinguish fault locations and can improve the efficiency of software debugging for multi-faults.

Key words　suspicious degree; multi-faults location; neural network; program debugging; fault symptoms

摘　要　针对软件多故障定位问题,提出一种基于神经网络的多故障定位模型.通过故障相关性分析,计算故障定位使用的输入对每个故障的支持度分量.利用神经网络模型学习输入的覆盖位置与各故障间的关系,针对每个可能包含故障的位置,构建理想输入作为已学习神经网络的输入,计算出该位置包含各故障的支持度,最终对每个故障确定其按支持度排序的位置序列,从而完成多故障定位的任务.实验结果表明,较传统方法,该模型对各故障可疑位置具有很强的分辨能力,表现出较大的优越性,对于提高软件多故障调试效率有很大帮助.

关键词　可疑度;多故障定位;神经网络;程序调试;故障征兆

中图法分类号　TP311.53

收稿日期:2011-01-10;**修回日期**:2012-01-05

基金项目:国家自然科学基金重大研究计划项目(90718021)

软件修复作为提高软件可信性的重要手段,一般需要经过故障定位、分析、修复、验证和部署几个阶段,由于软件的复杂性不断增大,程序员花费在故障定位阶段的时间越来越多,这严重阻碍软件质量的提升.因而如何有效地帮助程序员快速定位故障,从而缩短软件修复周期就成为现在国内外研究的一个热点.

Jones 等人[1]提出的 Tarantula 技术将语句 s_i 在失败测试用例中被执行的相对次数与在所有测试用例中被执行的相对次数之比作为 s_i 可疑度,然后将语句按可疑度排序后最终由程序员逐个检查确定故障位置.之后 Wong 等人[2]按成功和失败的测试用例对故障的支持度不同对各测试用例进行区分,改进了 Jones 等人的工作.文献[3-4]通过插桩技术收集软件执行相关谓词信息,利用统计分析方法确定各谓词处的可疑度;而 Zhang 等人[5]通过关键谓词状态转换技术确定故障位置,其主要思想是在失败执行中强制改变某谓词状态后若执行恢复正常,则认为该谓词处就是可能的故障位置.文献[6]利用 BP 神经网络学习测试用例与故障间的复杂关系,最后针对每个可疑位置构造一个虚测试作为已学习神经网络的输入,神经网络的输出作为该可疑位置的可疑度,从而得到按可疑度排序的位置序列来指导程序员进行故障定位,并随后使用径向基函数(radial basis function, RBF)网络对该方法进行了改进[7].还有利用其他如切片技术、形式概念分析等进行故障定位的方法,限于篇幅,本文不再赘述.

上述研究成果很大程度上提高了调试效率,但大多集中在单故障定位问题.事实上,程序单故障只是一定条件下的理想假设,到目前为止还没有一种技术可以揭示程序中存在的所有潜在故障,在实际的调试过程中往往是多故障情况.因而部分研究尝试按 one-bug-at-a-time 的策略将其方法应用于多故障情境中,其结果不甚理想.另有部分研究者[8-9]采用不同的聚类方法提高多故障定位并行性,由于其本质上是单故障的并行处理,回避故障间的相互影响,因而定位效果也不十分理想.其实正是由于故障之间的复杂关系使多故障定位问题变的十分复杂,要从本质上解决多故障定位问题,必须充分考虑故障相关信息,因此本文重点研究故障间相关性,通过深入分析故障间的征兆相关信息,计算出输入对各个故障的支持度分量,构造合适的神经网络模型学习输入与故障位置的关系,通过理想输入确定每个位置对各故障的支持度并最终完成多故障定位的目的.值得一提的是与本文相近文献[6-7]也使用了神经网络处理输入与故障位置之间的关系,但如前所述与本文方法存在本质上的区别.在实验阶段进行的对比结果表明,故障相关信息对多故障定位的定准率和分辨率等指标具有显著影响,本文提出方法在处理多故障定位问题上具有明显的优越性和实用价值.

1 基本概念

程序可以看作是作用于输入的函数,设程序输入集合为 U,$D\subset U$ 为选取的测试用例集合,V 为所有可能输出的集合,Y 为理想输出集合,则程序 P 定义为函数 $P:U\to Y$.同时将故障集定义为函数集 $F=\{f_1,f_2,\cdots,f_n\}$.在程序故障定位中,目的是根据程序发生故障信息寻找程序故障位置,而故障一般是程序缺陷(bug)被触发产生的,因而故障定位实际上是寻找与故障对应的缺陷(bug)所在位置.为描述方便,本文做如下约定:

故障和缺陷(bug)、故障 A 和 f_A、程序(软件) P 和函数 P 在不引起混淆的情况下均不加区分地使用.

定义 1. 设 $f\in F$,则 $P\circ f\triangleq P_f:U\to V$;该定义的实际意义是包含故障 $f\in F$ 的程序 P 是 P 和 f 的复合函数,据此可进一步递归定义 $P\circ f_1\circ f_2\triangleq P_{f_1}\circ f_2\triangleq P_{f_1f_2}$.

定义 2. 若 $\exists u\in U$,使得 $P_f(u)\in(V-Y)$,则称故障 f 可被 u 触发,并称 u 为 f 的一个触发,记为 u_f.

实际上,一个故障可以有多个触发,一个 $u\in U$ 也可能会触发多个故障,这正是目前程序多故障定位问题复杂的一个重要原因.在多故障定位中,提供相应触发,要求定位可被触发的故障位置.因而故障定位的过程就是选定输入集合 D,执行函数 $P_{f_1f_2\cdots f_m}$,获取相应的程序行为信息和故障征兆信息,最终确定与故障 $f_1,\cdots,f_m$ 对应的目标位置有序序列,在图 1 示例中,$D=\{t_1,t_2,t_3,t_4,t_5\}$,程序故障函数为 $P_{f_Af_Bf_C}$,征兆 $O=\{o_1,o_2,o_3,o_4\}$ 是故障所表现出来的征兆集合,目标位置 $L=\{l_1,l_2,\cdots,l_k\}$ 为可能包含故障的位置序列,最理想的定位结果就是使该序列经排序后的首个元素为故障位置;行为 $B=\{\boldsymbol{\pi}_{t_1},\boldsymbol{\pi}_{t_2},\boldsymbol{\pi}_{t_3},\boldsymbol{\pi}_{t_4},\boldsymbol{\pi}_{t_5}\}$ 是 $\forall t\in D$ 所对应的目标位置的覆盖向量集合.

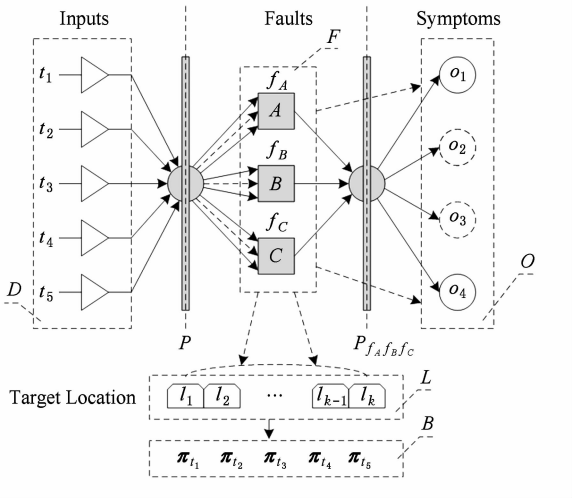

Fig. 1 Multi-faults location scheme.

图 1 多故障定位示意图

2 相关性分析

某个故障发生可能会伴随着其他故障,称为故障间的相关性.相关性分析的目的是为了在故障定位模型中,输入集合 D 中的元素以不同权值支持不同的故障,事实上即使一个输入没有触发任何故障,也从另一个方面提供了多故障定位所需要的信息.传统的研究思想主要是正向的,即寻找证据判断哪些位置更可能包含故障;而若一个输入未触发故障,则可以反向地判断哪些位置更不可能是故障位置.在多故障定位中,若一个输入触发某故障,则根据故障相关性分析,可以分析该输入对其他未触发故障的支持度,这其中包括正向的和反向的.根据图 1 中故障定位示意图,设 O_f 为故障 f 对应的征兆集合,可做如下定义.

定义 3. 若 $O_A \cap O_B \neq \varnothing$,则 A 与 B 相关,记为 $A \oplus B$.

性质 1. 故障相关具有自反性和对称性,即 1) $A \oplus A$;2)若 $A \oplus B$,则 $B \oplus A$.

设 $L=\{l_1, l_2, \cdots, l_k\}$ 为位置集合,与 L 对应的覆盖向量 $\boldsymbol{\pi}_{t_i}=(b_1, b_2, \cdots, b_k) \in \{0,1\}^k$,其中若 $P(t_i)$ 经过 L 中的 l_j,则 b_j 为 1,否则为 0,则:

定义 4. 若 $\exists u \in D, P_{f_A}(u) \in (V-Y)$,则称 $\boldsymbol{\pi}_u$ 为故障 A 的一个特征向量,记为 $\boldsymbol{\pi}_u^A$.

如前所述,相关性分析的目的是确定任意的输入对每个故障的支持分量,由此引入支持相关度的概念.

定义 5. 输入 $u \in D$ 对某个故障 A 的支持程度称为支持度 $\lambda_{u \to A}$,取值为 $[-1,1]$;具体地,若 $\boldsymbol{\pi}_u$ 为 A 的一个特征向量,则 $\lambda_{u \to A}=1$;若 $\neg(A \oplus B)$ 且 $\lambda_{u \to B}=1$,则 $\lambda_{u \to A}=-1$;其他情况下,本文给出其概率量化定义 $\lambda_{u \to A} \triangleq \begin{cases} p(A \mid u) \\ -p(\overline{A} \mid u) \end{cases}$,其实际意义是若执行某 $u \in D$,没有激发故障 A,则 u 对 A 发生($\lambda_{u \to A}>0$)或不会发生($\lambda_{u \to A}<0$)的支持程度,我们将在下一节构造合适的神经网络来进行计算.

理想情况下,判断哪些位置更可能是故障位置的信息和判断哪些位置更不可能是故障位置的信息对于故障定位的输出序列是等价的,这也是传统方法一般只使用正向信息(即 $\lambda_{u \to A} \in [0,1]$)的原因,而下面的定理 1 说明了使用 $\lambda_{u \to A} \in [-1,1]$ 信息进行故障定位具有更高的定准率.

不失一般性,分别记对应的输出序列为 $L_A^{\lambda_{u \to A} \in [0,1]}$ 和 $L_A^{\lambda_{u \to A} \in [-1,1]}$,为了更好地描述定理 1,接下来首先给出相关定义和引理.

定义 6. 理想输出位置序列 $L_A^*=\{l_{n1}^*, l_{n2}^*, \cdots, l_{nk}^*\}$ 是按各位置包含故障 A 的真实可能性大小排序的有序序列.

定义 7. 序列相似度 φ_A^{XY} 是衡量两个序列 L_A^X 和 L_A^Y 的相似程度的量.根据实际意义,可以有不同的计算公式,在本文中采用如下计算方法:

$$\Phi=\sum_{i=1}^{n} \frac{1}{2^i}(\text{bool}[l_i^X == l_i^Y]),$$

其中,$\text{bool}[l_i^X == l_i^Y]$ 表示布尔表达式的值,l_i^X 与 l_i^Y 相同时为 1,反之为 0;对于和式每项中的权值,其物理意义是对于实际定位方法输出的有序序列,其越靠前的元素相同与否对相似度的影响越大;进一步作归一化处理可得:

$$\varphi_A^{XY}=\Phi \Big/ \sum_{i=1}^{n} \frac{1}{2^i}.$$

定义 8. 定准率 $\xi_A=\dfrac{n_L-n_C}{n_L-1}$,其中 n_L 是位置序列长度,n_C 是按顺序检查序列直到确定出真实故障位置时所检查过的位置数目,ξ_A 是衡量对故障 A 定位准确性指标;例如对于某定位方法确定的有序序列 L_A^X,如果首个位置就是真实的故障位置,即 $n_C=1$,则 $\xi_A^X=1$;如果直到检查到该序列最后一个位置才确定故障位置,即 $n_C=n_L$,则 $\xi_A^X=0$,这时该定位方法不仅不能帮助程序员寻找故障位置反而降低了调试效率.

引理 1. 若 $\varphi_A^{X^*} \leqslant \varphi_A^{Y^*}$,则 $\xi_A^X \leqslant \xi_A^Y$.证明见附录 A.

引理 1 直观上说明了对于不同故障定位方法的

输出序列,与理想序列越相似,则其对应方法的故障定准率也越高,在此基础上,我们给出如下定理.

定理 1. $\xi_A^{\lambda_{u\to A}\in[0,1]}$ 与 $\xi_A^{\lambda_{u\to A}\in[-1,1]}$ 分别为与输出序列 $L_A^{\lambda_{u\to A}\in[0,1]}$ 和 $L_A^{\lambda_{u\to A}\in[-1,1]}$ 对应的定准率,则 $\xi_A^{\lambda_{u\to A}\in[0,1]}\leqslant\xi_A^{\lambda_{u\to A}\in[-1,1]}$.

证明. 理想情况下,$\varphi_A^{\lambda_{u\to A}\in[0,1]^*}=\varphi_A^{\lambda_{u\to A}\in[-1,1]^*}$,所以 $\xi_A^{\lambda_{u\to A}\in[0,1]}=\xi_A^{\lambda_{u\to A}\in[-1,1]}$;一般情况下,设对应序列分别为 $L_A^{\lambda_{u\to A}\in[0,1]}=\{l_1^+,\cdots,l_k^+\}$,$L_A^{\lambda_{u\to A}\in[-1,0]}=\{l_1^-,\cdots,l_k^-\}$,$L_A^{\lambda_{u\to A}\in[-1,1]}=\{l_1,\cdots,l_k\}$. 不失一般性,设 $\exists\, p\in\{1,\cdots,k\}$,$l_p^+\neq l_p^*$,$l_p^-\neq l_p^*$,而对于 $q\in\{1,\cdots,p-1\}$,$l_q^+=l_q^*$,$l_q^-=l_q^*$,即在位置 p 处出现了按可疑度排序的错排,此时由于 l_q 是按 l_q^+ 与 l_q^- 的可疑度差值(记为 Δl_q)进行排序,对 $q\in\{1,\cdots,p-1\}$,Δl_q 保序,而当 $q=p$ 时,$P(l_q=l_q^*)\geqslant 0$,所以 $\varphi_A^{\lambda_{u\to A}\in[0,1]^*}\leqslant\varphi_A^{\lambda_{u\to A}\in[-1,1]^*}$,根据引理 1,$\xi_A^{\lambda_{u\to A}\in[0,1]}\leqslant\xi_A^{\lambda_{u\to A}\in[-1,1]}$. 证毕.

3 神经网络的构造

神经网络具有很强的泛化能力,可通过对有限个样本的学习,学习到隐含在样本中的内在规律性. 本文结合不同神经网络的特点,将网络结构构建为两层,完成相应的计算任务.

第 1 层网络层选用概率神经网络(probabilistic neural networks, PNN),用于判断输入对各故障的支持度. PNN 是将贝叶斯统计方法映射到前馈神经网络结构,该类型网络收敛速度快、可以完成任意的非线性变换,非常适用于描述数据间的相关性,从而确定 PNN 在该任务中的高度可用性. 此时输入向量是 $\boldsymbol{x}_i=(x_1,\cdots,x_k)$,$i\in\{1,2,\cdots,m\}$,输出向量是 $\boldsymbol{\lambda}_u=(\lambda_1,\cdots,\lambda_m)$,网络结构如图 2 所示. 训练过程是将 $\boldsymbol{x}_i$ 由输入层直接送到模式层各模式单元中进行点积运算,完成非线性处理后送入累加层,依据 Parzen 方法计算相应的概率作为输出.

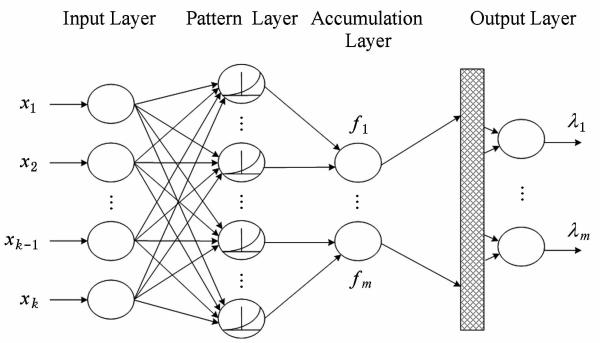

Fig. 2 Probabilistic neural networks.

图 2 概率神经网络

第 2 层选用径向基函数网络. 该网络具有结构自适应确定、输出与初始权值无关的优良特性,在曲面重构、故障诊断等领域有着广泛应用. 在确定了输入对不同故障支持权值后,利用 RBF 强大的逼近能力,计算每个位置对各故障的支持程度,从而确定了 RBF 在多故障定位中的可用性. 网络结构如图 3 所示. 采用与文献[7]类似的方法训练,此时输入向量组为 $(\boldsymbol{x}_1,\cdots,\boldsymbol{x}_m)=(\boldsymbol{\pi}_{t_i})^{\mathrm{T}}\times\boldsymbol{\lambda}_u$,输出为 $(y_1,\cdots,y_m)$.

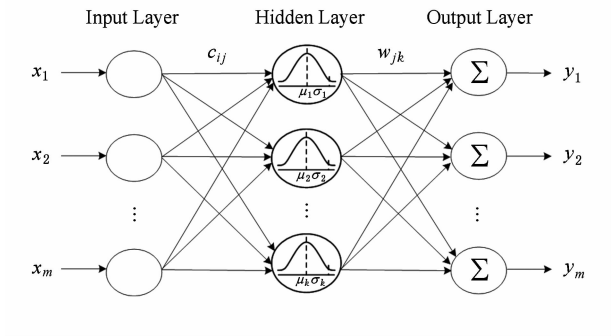

Fig. 3 Radial basis function neural networks.

图 3 RBF 神经网络

在故障定位时,对每一层神经网络构建理想输入:对于 PNN 网络,假设存在理想 u^* 对所有征兆都出现,此时输出 $\boldsymbol{\lambda}_u^*=(\lambda_1,\cdots,\lambda_m)$即为 u^* 对所有故障的理想支持度,将此输出作为构建下一层网络输入的分量;对第 2 层网络,构建理想覆盖 $\boldsymbol{\pi}_{t_i}^*=(b_1,b_2,\cdots,b_k)$,$b_i=1$,$b_j=0$,$j\neq i$ 作为 u^* 的覆盖向量,此时输入为 $(\boldsymbol{\pi}_{t_i}^*)^{\mathrm{T}}\times\boldsymbol{\lambda}_u^*$,输出 $\boldsymbol{y}_i^*=(y_1^i,\cdots,y_m^i)$,最后对任意故障 f_q,按 $(y_q^1,\cdots,y_q^k)$ 对位置 $\{l_1,\cdots,l_k\}$ 排序,进而最终完成软件的多故障定位过程.

4 实验分析

为验证本文方法的有效性、可用性及其优越性,本文设计 A,B 两组实验,按照软件规模不同选取 6 个真实软件作为实验对象;A 组针对 single-fault,B 组针对 multi-faults,如表 1 所示. 实验将主要关注以下几点:

1) PNN 网络的输出 $\boldsymbol{\lambda}_u=(\lambda_1,\cdots,\lambda_m)$与相关性分析;

2) 对 single-fault 的有效性和可用性,与文献[7-8]方法(RBF-FL)进行对比分析;

3) 对 multi-faults 的有效性和可用性,将文献[7]方法使用 one-bug-at-a-time 的策略进行多故障

定位和文献[8]方法(PARA-FL)与本文方法进行对比分析.

Table 1 Fault Software Version Information

表 1 故障软件版本相关信息

Software	LOC	Num of u Passed	Num of u_f	Num of Fault
Zunebug	22	8	3	1
Uniq ultrix4.3	1146	10	3	1
Looksvr4.0.1.1	1363	10	5	3
Deroff ultrix4.3	2236	20	10	3
Nullhttpd0.5.0	5575	50	20	5
Flex2.5.4	18775	80	30	5

分别运行软件的正常和故障输入,通过相关工具[10-11]获取软件各次运行时数据,采用本文方法进行后续分析.

表 2 是根据定义 3 得到的 Nullhttpd0.5.0 故障相关性数据,如前文所述,本文方法故障相关是通过故障征兆关联起来的,随着定位粒度的不同故障征兆具有不同表现形式,本文采用转移地址对-系统调用混杂序列作为观察的征兆序列.

Table 2 Fault Correlation for Nullhttpd0.5.0

表 2 Nullhttpd0.5.0 故障相关性

Fault	Fault				
	f_1	f_2	f_3	f_4	f_5
f_1	⊕	−	⊕	−	⊕
f_2	−	⊕	−	⊕	−
f_3	⊕	−	⊕	−	−
f_4	−	⊕		⊕	−
f_5	⊕	−	−	−	⊕

Note: ⊕ Related; − Unrelated

理论上表 2 中相关性数据应在 PNN 网络输出上有相对应的表现,例如在表中 $f_2 \oplus f_4$,则对于 f_2 的一个触发 u_{f_2},即使 u_{f_2} 未触发 f_4,$\lambda_{u_{f_4}}$ 也应该较大.

表 3 是针对 Nullhttpd0.5.0 第 1 层神经网络的输出,我们选取了其中的 7 个正常的输入和 5 个故障输入.在表 3 中,$\lambda_{u_1 \to f_2}=0.713$ 表明 u_1 对故障 f_2 的支持度较大,直观上就是执行 u_1 所覆盖的路径包含故障的可能性较大;$\lambda_{u_4 \to f_1}=-0.674$ 表明 u_1 对故障 f_2 不发生支持度较大,也就是执行 u_1 所覆盖的路径不包含故障的可能性较大;而传统的故障定位认为对于正常输入覆盖的路径一般包含故障位置可能性都较小.该实验结果表明,对于某些正常输入,尽管没有触发相关故障,其覆盖路径包含该故障的可能性也较大;进一步的 $\lambda_{u_8 \to f_1}$,$\lambda_{u_8 \to f_3}$,$\lambda_{u_9 \to f_2}$,$\lambda_{u_{10} \to f_3}$ 等均在 0.95 以上,这些输入对相应的故障具有很大的支持度,在实验中这些输入均触发了相应故障,与预期相符;同时表 3 中 u_f 列相关数据与表 2 故障相关性数据也基本一致,符合预想的实验结果;限于篇幅,其他软件版本相应的 PNN 网络输出值不再赘述.

图 4 是对不同方法针对 single-fault 的定准率 ξ 的比较,从图中可以看出 RBF-FL 与本文方法的 ξ 值较为接近,均高于 PARA-FL 相应值,这一方面说明对于 single-fault 使用故障相关信息的有效性,另一方面也说明了神经网络在软件故障定位问题中的优势,同时随着软件规模的增大,ξ 均不同程度地下降,而本文方法下降略微平缓,这也支持了相关领域的软件规模越大故障定位越困难的观点.

本文方法主要针对 multi-faults 的定位,对于 single-fault 可以看作是 multi-faults 的特例,因而上述实验结果与预期是一致的.对于 multi-faults 的定位,选取表 1 中的 Looksvr4.0.1.1,Deroff ultrix4.3,Nullhttpd0.5.0,Flex2.5.4 作为实验对象,结果如图 5 所示.在图中本文方法平均定准率从 0.883 到 0.732,对每个故障位置均能很好地分辨,而 PARA-FL 平均定准率从 0.830 到 0.538 且对每个故障位置均能较好地分辨,然而通过 one-bug-at-a-time 的

Table 3 Output $\lambda_u=(\lambda_1,\cdots,\lambda_m)$ for Nullhttpd0.5.0

表 3 Nullhttpd0.5.0 输出 $\lambda_u=(\lambda_1,\cdots,\lambda_m)$

Fault	u Passed							u_f				
	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}
f_1	0.231	0.013	−0.072	−0.674	−0.344	0.237	−0.342	0.981	0.317	0.768	0.015	0.854
f_2	0.713	−0.365	0.246	−0.258	0.215	−0.787	−0.521	−0.246	0.968	−0.321	0.564	−0.214
f_3	−0.215	−0.233	−0.134	0.358	−0.241	0.023	−0.046	0.933	0.218	0.979	0.476	0.487
f_4	0.296	0.021	−0.542	0.423	0.357	−0.542	0.321	−0.374	0.214	−0.213	0.993	−0.356
f_5	−0.324	0.267	0.654	−0.753	0.365	0.237	0.235	0.654	0.321	0.875	0.265	0.957

策略使用的 RBF-FL 尽管平均定准率从 0.786 到 0.518,但是其对每个故障不能很好地分辨,如对实验对象 Flex2.5.4,通过 RBF-FL 方法对其故障 f_4 的定准率达到了 0.794,而对其他故障定准率却较低,这主要是因为其在故障定位中对多故障不具有很好的分辨能力,使得包含故障 f_4 可能性大的位置在其他 4 个故障定位位置有序序列中也排的比较靠前所导致的.

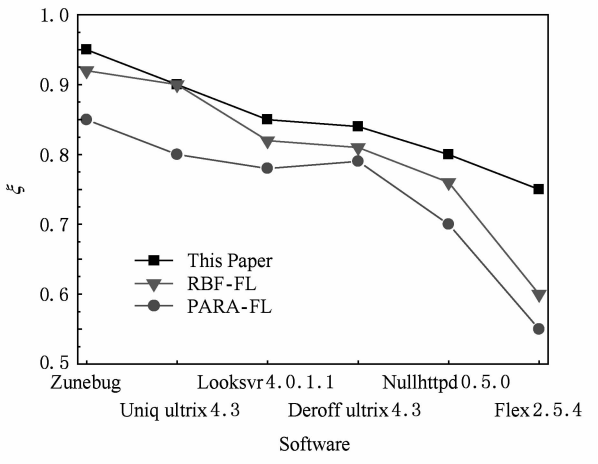

Fig. 4 ξ of single-fault.

图 4 single-fault 类型的定准率 ξ

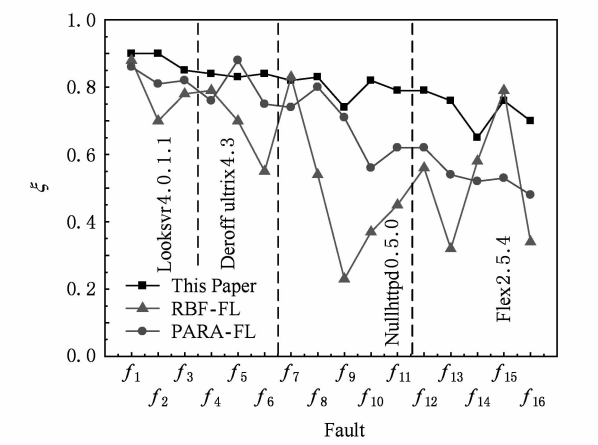

Fig. 5 ξ of multi-faults.

图 5 multi-faults 类型的定准率 ξ

为更好地说明方法对多故障的分辨能力,引入分辨率 $\beta=\sum_{i=1}^{m}(\xi_{f_i}-\delta)\Big/m$, m 为故障数,δ 为有效定准率阈值,因为定准率过低不仅不会帮助程序员定位故障,反而会降低程序员调试效率,原始的 Adhoc 即随机对位置进行检查以确定故障位置的方法的定准率的数学期望 E_δ 是 0.5,因此在统计意义上对任何故障定位方法而言,定准率低于 E_δ 的都会降低程序员调试效率,不具有实用意义. 在实验中对 δ 值的确定并没有严格要求,原则上高于 E_δ 认为是有效的;对应图 5 中实验对象各方法分辨率比较如表 4. 从表中可以看出本文方法对实验对象中故障分辨率均高于其他两种方法,而 RBF-FL 是相对低的.

Table 4 Comparison for $\beta(\delta=0.618)$

表 4 各方法分辨率 β 比较($\delta=0.618$)

Software	m	β-This Paper	β-RBF-FL	β-PARA-FL
Looksvr4.0.1.1	3	0.2653	0.1687	0.2120
Deroff ultrix4.3	3	0.2187	0.0620	0.1787
Nullhttpd0.5.0	5	0.1820	−0.1340	0.0680
Flex2.5.4	5	0.114	−0.1000	−0.0800

图 6 是在时间开销方面的比较,对于使用 one-bug-at-a-time 策略的 RBF-FL 时间开销计算方法是将每个故障所需要开销时间累加所得.

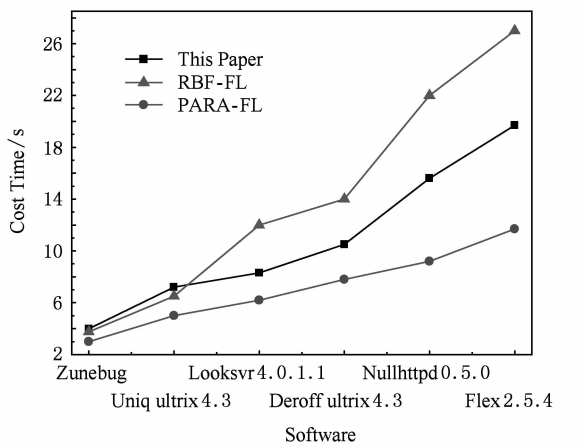

Fig. 6 Cost time comparison.

图 6 开销时间比较

从图 6 中可以看出,对于 single-fault 而言,PARA-FL 相对较低,这主要是由于神经网络的学习需要较长的时间;但本文方法和 RBF-FL 差异不是十分明显,这主要是因为本文方法在第 1 层神经网络选用的 PNN 网络具有收敛速度快得特点;而对于 multi-faults 而言,PARA-FL 所需要时间最少,本文方法次之,但 RBF-FL 方法所需要时间比较高,这主要是由于在故障定位中 PARA-FL 和本文方法都是针对多故障定位问题的,因而具有一定优势. 另外从图 6 中来看,随着程序规模的增大和故障数的增加,时间开销都有所增加,但本文方法和 RBF-FL 增长较快,这主要是由于神经网络所需要的计算量所决定的,因此如何进一步降低本文方法时间开销值得进一步的研究.

5 结 论

虽然现存在许多软件故障定位方法,但是这些定位方法都很少涉及多故障定位情况.本文针对该问题,深入研究了故障间复杂关系,通过分析故障间的相互关系,将输入按不同权值分解为支持各故障的输入分量,充分利用了每个输入提供的故障定位信息;结合神经网络特点构建模型完成相应的计算任务.实验结果表明,本文方法显著提高了故障定位的定准率,对多故障具有很强的分辨能力,表现出较大的优越性,因而具有很高的实用价值.

参 考 文 献

[1] Jones J A, Harrold M J. Empirical evaluation of the Tarantula automatic fault-localization technique [C] //Proc of the 20th IEEE/ACM Int Conf on Automated Software Engineering. New York: ACM, 2005: 273-282

[2] 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

[3] Liblit B, Naik M, Zheng A X, et al. Scalable statistical bug isolation [C] //Proc of the ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2005: 15-26

[4] Liu C, Fei L, Yan X, et al. Statistical debugging: A hypothesis testing-based approach [J]. IEEE Trans on Software Engineering, 2006, 32(10): 831-848

[5] Zhang X, Gupta N, Gupta R. Locating faults through automated predicate switching [C] //Proc of the 28th Int Conf on Software Engineering. New York: ACM, 2006: 272-281

[6] Wong W E, Qi Y. BP neural network-based effective fault localization [J]. Int Journal of Software Engineering and Knowledge Engineering, 2009, 19(4): 573-597

[7] Wong W E, Vidroha D, Bhavani T, et al. RBF neural network-based fault localization, UTDCS-20-10 [R]. Dallas: Golden Department of Computer Science, University of Texas at Dallas, 2010

[8] Jones J A, Bowring J, Harrold M J. Debugging in parallel [C] //Proc of the Int Symp on Software Testing and Analysis. New York: ACM, 2007: 16-26

[9] Zheng A X, Jordan M I, Liblit B, et al. Statistical debugging: Simultaneous isolation of multiple bugs [C] //Proc of the 23rd Int Conf on Machine Learning. New York: ACM, 2006: 26-29

[10] Bryan B, Jeffrey K H. An API for runtime code patching [J]. International Journal of High Performance Computing Applications, 2000, 14(4): 317-329

[11] Liu Yongpo, Wu Ji, Jin Maozhong, et al. Experimentation study of BBN-based fault localization [J]. Journal of Computer Research and Development, 2010, 47(4): 707-715 (in Chinese)
(柳永坡, 吴际, 金茂忠, 等. 基于贝叶斯统计推理的故障定位实验研究[J]. 计算机研究与发展, 2010, 47(4): 707-715)

He Jialang, born in 1984. PhD. His current research interests include software fault location and software repair.

Zhang Hong, born in 1956. Professor and PhD supervisor in Nanjing University of Science and Technology. His main research interests include network performance and information security (zhhong@mail.njust.edu.cn).

附录 A

引理 A1. 若 $\varphi_A^{X^*} \leqslant \varphi_A^{Y^*}$,则 $\xi_A^X \leqslant \xi_A^Y$.

证明. 由正文定义 6 $L_A^* = \{l_{n_1}^*, l_{n_2}^*, \cdots, l_{n_k}^*\}$,设 $L_A^X = \{l_{x_1}^X, l_{x_2}^X, \cdots, l_{x_k}^X\}$, $L_A^Y = \{l_{y_1}^Y, l_{y_2}^Y, \cdots, l_{y_k}^Y\}$,其中 $(n_1, n_2, \cdots, n_k)$, $(x_1, x_2, \cdots, x_k)$, $(y_1, y_2, \cdots, y_k)$ 均为自然数组 $(1, 2, \cdots, k)$ 的重排列.

1) 若 $\varphi_A^{X^*} = \varphi_A^{Y^*}$,则 $(n_1, n_2, \cdots, n_k) = (x_1, x_2, \cdots, x_k) = (y_1, y_2, \cdots, y_k)$,显然 $\xi_A^X = \xi_A^Y$;

2) 否则,就必定存在 x_i, y_j,满足 $(n_1, n_2, \cdots, n_{i-1}) = (x_1, x_2, \cdots, x_{i-1})$, $(n_1, n_2, \cdots, n_{j-1}) = (y_1, y_2, \cdots, y_{j-1})$,且 $x_i \neq n_i$, $y_j \neq n_j$,此时 $\varphi_A^{X^*} = \left(\sum_{p=1}^{i-1} \frac{1}{2^p}\right) + \frac{1}{2^i}\text{bool}[l_{x_i}^X == l_{n_i}^*] + \sum_{p=i+1}^{k} \frac{1}{2^p}\text{bool}[l_{x_p}^X == l_{n_p}^*]$,由于 $x_i \neq n_i$,故 $\text{bool}[l_{x_i}^X == l_{n_i}^*] = 0$;所以 $\varphi_A^{X^*} = \left(\sum_{p=1}^{i-1} \frac{1}{2^p}\right) + \sum_{p=i+1}^{k} \frac{1}{2^p}\text{bool}[l_{x_p}^X == l_{n_p}^*]$,同理 $\varphi_A^{Y^*} = \left(\sum_{q=1}^{j-1} \frac{1}{2^q}\right) + \sum_{q=j+1}^{k} \frac{1}{2^q}\text{bool}[l_{y_q}^Y == l_{n_q}^*]$,显然若 $\varphi_A^{X^*} < \varphi_A^{Y^*}$,则 $i < j$,根据定义 8 易知 $n_C^X > n_C^Y$,所以 $\xi_A^X \leqslant \xi_A^Y$. 证毕.