(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 105893256 A
(43)申请公布日 2016.08.24

(21)申请号 201610191313.9

(22)申请日 2016.03.30

(71)申请人 西北工业大学
地址 710072 陕西省西安市友谊西路127号

(72)发明人 郑炜 柏晗 刘文兴 王文鹏 谭海斌

(74)专利代理机构 西北工业大学专利中心 61204
代理人 王鲜凯

(51)Int.Cl.
G06F 11/36(2006.01)

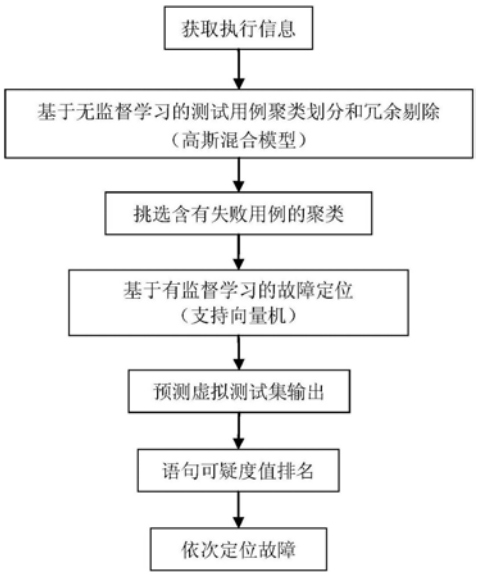
权利要求书3页 说明书7页 附图4页

(54)发明名称

基于机器学习算法的软件故障定位方法

(57)摘要

本发明公开了一种基于机器学习算法的软件故障定位方法，用于解决现有软件故障定位方法定位效率低的技术问题。技术方案是首先利用高斯混合分布描述现实程序中可能存在的故障分布，使得程序中的故障分布更为明确；再借助基于高斯混合模型的聚类分析方法，对冗余测试样本进行剔除，找到针对特定故障的专用测试集，从而减少了冗余用例对定位精度的不良影响；再修改支持向量机模型来适应不平衡的数据样本，并结合并行调试理论找到用例覆盖信息和执行结果之间的非线性映射关系，使得机器学习算法不会因样本不均而陷于局部最优解问题。最后，设计虚拟测试套件，放到训练好的模型中预测，得出语句可疑度值排名，进行故障定位，提高了软件故障定位效率。



CN 105893256 A

1.一种基于机器学习算法的软件故障定位方法,其特征在于包括以下步骤:

步骤一、获取程序的执行信息;采用GNU标准编译器GCC和Gcov工具对待测文件进行编译,编译命令为:

gcc-O2test.c-I.-fprofile-arcs-ftest-coverage-o test.exe

所述命令在编译的同时生成gcov所需的test.gcno文件;然后运行可执行文件test.exe,生成test.gcda文件,用以记录插桩信息;最后,用gcov test.c命令得到test.c.gcov文件;其中在每行代码的开头,-表示此行代码不是可执行语句,数字表示此行代码在运行过程中被执行的次数,#####表示此行代码虽为可执行语句,但在本次执行中并没有被覆盖到;

借助Gcov工具,编写C语言代码,在每执行一次测试用例之后,都对生成的gcov文件进行分析,得到错误版本程序的语句被测试用例覆盖的信息;收集测试用例的执行结果的步骤如下:

a)编译原版本程序代码,执行测试用例,将输出结果放到测试套件的outputs文件夹中;

b)运行错误版本程序程序,将输出结果放到newoutputs文件夹下;

c)将newoutputs文件夹中的测试输出同outputs文件夹中的输出相比较,如果输出结果一致,说明测试用例执行成功,如果输出结果不一致,说明测试用例执行失败;

步骤二、通过对故障版本的测试用例覆盖信息和执行结果的采集,生成用以进行故障定位的实验样本;构建高斯混合模型来无限逼近故障分布,使测试集分成属于各自分布的用例聚类;高斯混合模型本质上是单一高斯分布的概率密度函数的加权求和,且各项的计算结果即为样本属于各类的概率大小,即:

$$\varpi_i(k)=\frac{w_k N(x_i\mid\lambda_k)}{N(x_i\mid\lambda)} \qquad (1)$$

其中,w_k是第k个分布$N(x_i|\lambda_k)$的权重,λ为模型参数,$\varpi_i(k)$为样本x_i属于k的概率;假设程序代码P中含有m条语句,其中m_f条语句含有故障,m_p条语句正确,且满足$\begin{cases}0\le m_f\ll m_p<m\\ m_f+m_p=m\end{cases}$,则针对P的一个测试集T含有t个测试用例,其中包含t_f个失败用例以及t_p个成功用例,且满足$\begin{cases}0\le t_f\ll t_p<t\\ t_f+t_p=t\end{cases}$;由于错误的测试用例覆盖了故障语句,正确执行的测试用例可能覆盖也可能没有覆盖故障语句;因此,程序故障在代码中的分布会直接导致测试用例的覆盖信息也服从该分布;假设测试集T的第i个测试用例为t_i,程序P的第j条语句为s_j,令$C_{i,j}=1$表示t_i执行时覆盖了语句s_j,$C_{i,j}$值为0时表示未覆盖;那么,测试用例t_i对程序P的覆盖信息表示为$C_i=(C_{i,1},C_{i,2},\ldots,C_{i,m})$;用这一向量表示测试用例的特征信息,则对于测试集T来说,其中的每一个测试用例都是多维特征空间中的一个点;由于故障语句大多被失败用例覆盖,则这t_f个数据点在特征空间中将聚在一起,因而服从某种单一分布;并且,在这t_f个数据点的附近,还会聚集一些覆盖了故障语句的成功用例,和一些未覆盖该故障语句但是执行信息与这t_f个点很相似的成功用例,因此这些用例也将服从这t_f个点的分布;

步骤三、对冗余的测试用例进行剔除;将步骤二获得的测试集T作为输入,剔除其中的

失败用例，将剩下的正确用例放到高斯混合模型中训练；用EM算法求解模型参数之后，将属于同一分布的测试用例聚成一类；然后，在这几类中，按照类平均距离法则，寻找到离剔除出去的失败用例集最近的那一类，将这两类合并，组成最终用来做故障定位的专用测试集；

测试用例聚类划分和冗余剔除这两步都运行于Windows环境下，采用Matlab数学分析软件中的Voicebox工具箱；Voicebox收纳了包括GMM在内的多种概率密度函数；采用Voicebox工具箱中的gaussmix和gaussmixp函数来进行模型训练和预测；其中，gaussmix函数的使用方法如下：

function[m,v,w,g,f,pp,gg]＝gaussmix(x,c,l,m0,v0,w0)

gaussmixp函数的使用方法如下：

function[lp,rp,kh,kp]＝gaussmixp(y,m,v,w)

gaussmix和gaussmixp函数能帮助解决用例划分和冗余剔除的问题，从而找到最针对特定故障的测试套件子集；

步骤四、利用基于支持向量机模型的监督学习算法进行故障定位；由于向量$C_i=(C_{i,1},C_{i,2},...,C_{i,m})$作为测试用例覆盖信息的同时，还能够表示为特征空间中的一个数据点；而测试用例的输出r_i即表示为每个样本点所属的类别；因此，把测试用例的覆盖信息C_i当做支持向量机的训练输入，把测试用例的执行结果r_i当做训练输出，以此来训练支持向量机；训练好的模型反映了测试用例的覆盖信息与执行结果之间的非线性映射关系，利用这种关系，间接通过如下的虚拟测试集来找到故障语句；

$$\begin{bmatrix} C_{t_1} \\ C_{t_2} \\ \vdots \\ C_{t_m} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

所述虚拟测试集共有m条测试用例，对应m条程序语句；其中，第i条测试用例只覆盖第i条程序语句，使得整个覆盖信息表是一个对角矩阵，对角线的值为1，其他为0；将虚拟测试集放在训练好的支持向量机模型中，预测哪些用例会被分为失败的一类；

考虑到惩罚因子的作用是允许支持向量机错分某些离群点的程度，因此，修改分类超平面的优化目标函数，使之成为：

$$\min \frac{1}{2}\|\omega\|^2 + C_+ \sum_{i=1}^{p} \varepsilon_i + C_- \sum_{i=p+1}^{n} \varepsilon_i \qquad (2)$$
$$s.t., y_i(\omega \bullet x_i + b) \geq 1 - \varepsilon_i, i = 1,...,n$$

这里，i＝1,...,p是分类为执行成功的样本，i＝p+1,...,n是分类为执行失败样本，ε_i为松弛变量，n为样本总数；C_+与C_-的取值选择根据这两类样本数量的反比来确定；采用LibSVM工具箱，使用用于训练支持向量机模型的svm-train.exe程序以及用于预测的svm-predict.exe程序；其中训练方法如下：

svm-train[options]training_set_file[model_file]

之后，构建虚拟矩阵来测试该模型：

svm-predict[options]test_file model_file output_file

这里取三个输出:[predicted_label,accuracy,decision_values],分别表示的预测结果标签、预测准确度和决策值;由这三个值得到一条语句含有故障的可疑度值,并按大小关系排列成表;

步骤五、根据可疑度值列表,从上往下依次检测语句,直到故障被定位。

基于机器学习算法的软件故障定位方法

技术领域

[0001] 本发明涉及一种软件故障定位方法,特别涉及一种基于机器学习算法的软件故障定位方法。

背景技术

[0002] 软件测试在软件开发过程中占据了大量的人力物力,而故障定位又是测试中付出代价最高的行为之一。因此,人们提出自动化故障定位技术来分析程序源代码和测试过程中的程序行为及结果,从而定位出软件的故障。目前,国内外专家已经在自动故障定位领域里提出了多种理论和方法,并取得了一定的成绩。

[0003] Wong等人在文献“BP Neural Network-based Effective Fault Localization.International Journal of Software Engineering and Knowledge Engineering,2009,19(4):573-597”中提出了一种基于反向传播(BP)神经网络的故障定位技术。他们用语句的覆盖信息和测试用例的执行结果来训练BP神经网络模型,并且设计一个虚拟测试集作为训练好的神经网络模型的输入,由于这个虚拟测试集的每一个测试用例只覆盖一条语句,所以可以认为输出不仅是测试用例成功或失败的可能性,也是覆盖的语句含有故障的可疑度值。

[0004] 之后,他们针对BP神经网络的局部最优解问题,又在文献“Using an RBF Neural Network to Locate Program Bugs.Proc.of the 19th IEEE International Symposium on Software Reliability Engineering.Seattle,Washington,USA,2008:27-38”中提出了一种基于径向基函数(RBF)神经网络模型的故障定位技术,来降低局部最优解的影响。然而无论是基于反向传播神经网络的故障定位,还是基于径向基函数神经网络模型的故障定位,都存在着各自的局限性和不足。尤其是在面对诸如故障分布不明、测试用例冗余、测试样本倾斜等问题时,这些故障定位技术的有效性都大为降低。

发明内容

[0005] 为了克服现有软件故障定位方法定位效率低的不足,本发明提供一种基于机器学习算法的软件故障定位方法。该方法首先利用高斯混合分布描述现实程序中可能存在的故障分布,使得程序中的故障分布更为明确;再借助基于高斯混合模型的聚类分析方法,对冗余测试样本进行剔除,找到针对特定故障的专用测试集,从而减少了冗余用例对定位精度的不良影响;然后,修改支持向量机模型来适应不平衡的数据样本,并结合并行调试理论来找到用例覆盖信息和执行结果之间的非线性映射关系,使得机器学习算法不会因样本不均而陷于局部最优解问题。最后,设计虚拟测试套件,放到训练好的模型中预测,得出语句可疑度值排名,进行故障定位,提高了软件故障定位效率。

[0006] 本发明解决其技术问题所采用的技术方案:一种基于机器学习算法的软件故障定位方法,其特点是包括以下步骤:

[0007] 步骤一、获取程序的执行信息。采用GNU标准编译器GCC和Gcov工具对待测文件进

行编译，编译命令为：

[0008] gcc-O2test.c-I.-fprofile-arcs-ftest-coverage-o test.exe

[0009] 所述命令在编译的同时生成gcov所需的test.gcno文件。然后运行可执行文件test.exe，生成test.gcda文件，用以记录插桩信息。最后，用gcov test.c命令得到test.c.gcov文件。其中在每行代码的开头，-表示此行代码不是可执行语句，数字表示此行代码在运行过程中被执行的次数，#####表示此行代码虽为可执行语句，但在本次执行中并没有被覆盖到。

[0010] 借助Gcov工具，编写C语言代码，在每执行一次测试用例之后，都对生成的gcov文件进行分析，得到错误版本程序的语句被测试用例覆盖的信息。收集测试用例的执行结果的步骤如下：

[0011] a)编译原版本程序代码，执行测试用例，将输出结果放到测试套件的outputs文件夹中；

[0012] b)运行错误版本程序程序，将输出结果放到newoutputs文件夹下；

[0013] c)将newoutputs文件夹中的测试输出同outputs文件夹中的输出相比较，如果输出结果一致，说明测试用例执行成功，如果输出结果不一致，说明测试用例执行失败。

[0014] 步骤二、通过对故障版本的测试用例覆盖信息和执行结果的采集，生成用以进行故障定位的实验样本。构建高斯混合模型来无限逼近故障分布，使测试集分成属于各自分布的用例聚类。高斯混合模型本质上是单一高斯分布的概率密度函数的加权求和，且各项的计算结果即为样本属于各类的概率大小，即：

[0015] $$\varpi_i(k)=\frac{w_k N(x_i\mid\lambda_k)}{N(x_i\mid\lambda)} \qquad (1)$$

[0016] 其中，w_k是第k个分布$N(x_i|\lambda_k)$的权重，λ为模型参数，$\varpi_i(k)$为样本x_i属于k的概率。假设程序代码P中含有m条语句，其中m_f条语句含有故障，m_p条语句正确，且满足$\begin{cases}0\le m_f \ll m_p < m\\ m_f+m_p=m\end{cases}$，则针对P的一个测试集T含有t个测试用例，其中包含t_f个失败用例以及t_p个成功用例，且满足$\begin{cases}0\le t_f \ll t_p < t\\ t_f+t_p=t\end{cases}$。由于错误的测试用例覆盖了故障语句，正确执行的测试用例可能覆盖也可能没有覆盖故障语句。因此，程序故障在代码中的分布会直接导致测试用例的覆盖信息也服从该分布。假设测试集T的第i个测试用例为t_i，程序P的第j条语句为s_j，令$C_{i,j}=1$表示t_i执行时覆盖了语句s_j，$C_{i,j}$值为0时表示未覆盖。那么，测试用例t_i对程序P的覆盖信息表示为$C_i=(C_{i,1},C_{i,2},\ldots,C_{i,m})$。用这一向量表示测试用例的特征信息，则对于测试集T来说，其中的每一个测试用例都是多维特征空间中的一个点。由于故障语句大多被失败用例覆盖，则这t_f个数据点在特征空间中将聚在一起，因而服从某种单一分布。并且，在这t_f个数据点的附近，还会聚集一些覆盖了故障语句的成功用例，和一些未覆盖该故障语句但是执行信息与这t_f个点很相似的成功用例，因此这些用例也将服从这t_f个点的分布。

[0017] 步骤三、对冗余的测试用例进行剔除。将步骤二获得的测试集T作为输入，剔除其中的失败用例，将剩下的正确用例放到高斯混合模型中训练。用EM算法求解模型参数之后，将属于同一分布的测试用例聚成一类。然后，在这几类中，按照类平均距离法则，寻找到离

剔除出去的失败用例集最近的那一类，将这两类合并，组成最终用来做故障定位的专用测试集。

[0018] 测试用例聚类划分和冗余剔除这两步都运行于Windows环境下，采用Matlab数学分析软件中的Voicebox工具箱。Voicebox收纳了包括GMM在内的多种概率密度函数。采用Voicebox工具箱中的gaussmix和gaussmixp函数来进行模型训练和预测。其中，gaussmix函数的使用方法如下：

[0019] function[m,v,w,g,f,pp,gg]＝gaussmix(x,c,l,m0,v0,w0)

[0020] gaussmixp函数的使用方法如下：

[0021] function[lp,rp,kh,kp]＝gaussmixp(y,m,v,w)

[0022] gaussmix和gaussmixp函数能帮助解决用例划分和冗余剔除的问题，从而找到最针对特定故障的测试套件子集。

[0023] 步骤四、利用基于支持向量机模型的监督学习算法进行故障定位。由于向量$C_i=(C_{i,1},C_{i,2},...,C_{i,m})$作为测试用例覆盖信息的同时，还能够表示为特征空间中的一个数据点。而测试用例的输出r_i即表示为每个样本点所属的类别。因此，把测试用例的覆盖信息C_i当做支持向量机的训练输入，把测试用例的执行结果r_i当做训练输出，以此来训练支持向量机。训练好的模型反映了测试用例的覆盖信息与执行结果之间的非线性映射关系，利用这种关系，间接通过如下的虚拟测试集来找到故障语句。

[0024] $$\begin{bmatrix} C_{t_1} \\ C_{t_2} \\ \vdots \\ C_{t_m} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

[0025] 所述虚拟测试集共有m条测试用例，对应m条程序语句。其中，第i条测试用例只覆盖第i条程序语句，使得整个覆盖信息表是一个对角矩阵，对角线的值为1，其他为0。将虚拟测试集放在训练好的支持向量机模型中，预测哪些用例会被分为失败的一类。

[0026] 考虑到惩罚因子的作用是允许支持向量机错分某些离群点的程度，因此，修改分类超平面的优化目标函数，使之成为：

[0027] $$\min \frac{1}{2}\|\omega\|^2 + C_+ \sum_{i=1}^{p} \varepsilon_i + C_- \sum_{i=p+1}^{n} \varepsilon_i \qquad (2)$$
$$s.t., y_i(\omega \bullet x_i + b) \geq 1 - \varepsilon_i, i = 1,...,n$$

[0028] 这里，i＝1,...,p是分类为执行成功的样本，i＝p+1,...,n是分类为执行失败样本，ε_i为松弛变量，n为样本总数。C_+与C_的取值选择根据这两类样本数量的反比来确定。采用LibSVM工具箱，使用用于训练支持向量机模型的svm-train.exe程序以及用于预测的svm-predict.exe程序。其中训练方法如下：

[0029] svm-train[options]training_set_file[model_file]

[0030] 之后，构建虚拟矩阵来测试该模型：

[0031] svm-predict[options]test_file model_file output_file

[0032] 这里取三个输出:[predicted_label,accuracy,decision_values]，分别表示的

预测结果标签、预测准确度和决策值。由这三个值得到一条语句含有故障的可疑度值，并按大小关系排列成表。

[0033] 步骤五、根据可疑度值列表，从上往下依次检测语句，直到故障被定位。

[0034] 本发明的有益效果是：该方法首先利用高斯混合分布描述现实程序中可能存在的故障分布，使得程序中的故障分布更为明确；再借助基于高斯混合模型的聚类分析方法，对冗余测试样本进行剔除，找到针对特定故障的专用测试集，从而减少了冗余用例对定位精度的不良影响；然后，修改支持向量机模型来适应不平衡的数据样本，并结合并行调试理论来找到用例覆盖信息和执行结果之间的非线性映射关系，使得机器学习算法不会因样本不均而陷于局部最优解问题。最后，设计虚拟测试套件，放到训练好的模型中预测，得出语句可疑度值排名，进行故障定位，提高了软件故障定位效率。

[0035] 为了验证本发明方法的效果，采用SIR中最为经典的Siemens测试套件和Space测试套件，来对本发明方法的有效性进行评估。Siemens测试套件包含有7组测试程序，自2003年被应用到NNQ技术的有效性评估上之后，该测试套件已成为故障定位领域内使用最多的测试数据集。Space是由欧洲航天局开发的一款解释器程序。作为一个大型测试集，它包含有38个错误版本，9126行代码，3657条可执行语句以及13585条测试用例。在这两个标准测试套件之上，应用Wong等人提出的EXAM得分体系，来测试本发明的实际定位效果。实验结果表明，由于采用了基于高斯混合模型的无监督学习算法，以及基于支持向量机的监督学习算法，故障定位技术所面临的故障分布不明、测试样本冗余和样本分布不均的问题，得到了有效的解决。从图2～8中可以看出，本发明(图中GVM曲线)比背景技术方法能更块地完成故障定位工作。并且，在0％到10％的分值区间内，故障定位的效率提升了20％以上，这验证了本发明方法的确能够有效地提高软件故障定位效率，降低定位所需的工作量。

[0036] 下面结合附图和具体实施方式对本发明作详细说明。

附图说明

[0037] 图1是本发明基于机器学习算法的软件故障定位方法的流程图。

[0038] 图2是本发明基于机器学习算法的软件故障定位方法在print_tokens套件上的实际测试结果。

[0039] 图3是本发明基于机器学习算法的软件故障定位方法在replace套件上的实际测试结果。

[0040] 图4是本发明基于机器学习算法的软件故障定位方法在schedule套件上的实际测试结果。

[0041] 图5是本发明基于机器学习算法的软件故障定位方法在tcas套件上的实际测试结果。

[0042] 图6是本发明基于机器学习算法的软件故障定位方法在tot_info套件上的实际测试结果。

[0043] 图7是本发明基于机器学习算法的软件故障定位方法在整个Siemens套件上的实际测试结果。

[0044] 图8是本发明基于机器学习算法的软件故障定位方法在Space套件上的实际测试结果。

具体实施方式

[0045] 参照图1-8。本发明基于机器学习算法的软件故障定位方法具体步骤如下：

[0046] (1)首先，获取程序的执行信息。由于本发明中程序执行信息的采集均运行于GNU/Linux环境下，且测试套件均使用标准C语言编写，因此在本发明的研究工作中，所有程序均采用GNU标准编译器GCC(GNU Compiler Collection)进行编译。在测试样本采集这一步里，主要用到的是Gcov工具。Gcov是一种命令行形式的控制台程序。它和GCC相配合，能对C/C++文件进行程序插桩和覆盖分析。首先对待测文件进行编译，编译命令为：

[0047] gcc-O2test.c-I.-fprofile-arcs-ftest-coverage-o test.exe

[0048] 该命令在编译的同时生成gcov所需的test.gcno文件。然后运行可执行文件test.exe，生成test.gcda文件，用以记录插桩信息。最后，用gcov test.c命令就可以得到test.c.gcov文件。其中在每行代码的开头，“-”表示此行代码不是可执行语句，数字表示此行代码在运行过程中被执行的次数，“#####”表示此行代码虽为可执行语句，但在本次执行中并没有被覆盖到。

[0049] 借助Gcov工具，手工编写C语言代码，在每执行一次测试用例之后，都对生成的gcov文件进行分析，从而得到错误版本程序的语句被测试用例覆盖的信息。此外，本发明还需要收集测试用例的执行结果。主要步骤如下：

[0050] d)编译原版本程序代码，执行测试用例，将输出结果放到测试套件的outputs文件夹中；

[0051] e)运行错误版本程序程序，将输出结果放到newoutputs文件夹下；

[0052] f)将newoutputs文件夹中的测试输出同outputs文件夹中的输出相比较，如果输出结果一致，就说明测试用例执行成功，否则执行失败。

[0053] 这些步骤的自动化过程同样也是本发明手工编写的C语言代码来实现的。

[0054] (2)通过对故障版本的测试用例覆盖信息和执行结果的采集，生成了用以进行故障定位的实验样本。接下来，构建高斯混合模型来无限逼近故障分布，使测试集分成属于各自分布的用例聚类。高斯混合模型本质上是单一高斯分布的概率密度函数的加权求和，且各项的计算结果即为样本属于各类的概率大小，即：

[0055] $$\varpi_i(k)=\frac{w_k N(x_i\mid\lambda_k)}{N(x_i\mid\lambda)} \qquad (1)$$

[0056] 其中，第k个分布$N(x_i|\lambda_k)$的权重为w_k，λ为模型参数，$\varpi_i(k)$为样本x_i属于k的概率。在这一原理的帮助下，可以近似地模拟软件故障的分布。假设程序代码P中含有m条语句，其中m_f条语句含有故障，m_p条语句正确，且满足$\begin{cases}0\le m_f\ll m_p<m\\ m_f+m_p=m\end{cases}$，则针对P的一个测试集T含有t个测试用例，其中包含t_f个失败用例以及t_p个成功用例，且满足$\begin{cases}0\le t_f\ll t_p<t\\ t_f+t_p=t\end{cases}$。由于错误的测试用例大多覆盖了故障语句，正确执行的测试用例可能覆盖也可能没有覆盖故障语句。因此，程序故障在代码中的分布会直接导致测试用例的覆盖信息也服从该分布。假设测试集T的第i个测试用例为t_i，程序P的第j条语句为s_j，令$C_{i,j}=1$表示t_i执行时覆盖了语句

s_j，$C_{i,j}$值为0时表示未覆盖。那么，测试用例t_i对程序P的覆盖信息可以表示为$C_i=(C_{i,1},C_{i,2},...,C_{i,m})$。用这一向量表示测试用例的特征信息，则对于测试集T来说，其中的每一个测试用例都是多维特征空间中的一个点。由于故障语句大多被失败用例覆盖，则这t_f个数据点在特征空间中将聚在一起，因而服从某种单一分布。并且，在这t_f个数据点的附近，还会聚集一些覆盖了故障语句的成功用例，和一些未覆盖该故障语句但是执行信息与这t_f个点很相似的成功用例，因此这些用例也将服从这t_f个点的分布。传统的软件故障定位技术把整个程序故障及其测试集看作符合某种单一分布，这种方式使得程序故障并不能够很好地被暴露出来，这是因为故障会随着程序依赖关系被传递到其他语句中去。而本发明提出基于高斯混合模型的测试用例聚类划分的方法，能从测试集中寻找最能直观展现故障存在位置的子测试集，并以此作为故障定位技术的输入数据。

[0057] 然后，对冗余的测试用例进行剔除。在用高斯混合模型模拟现实中软件故障分布的同时，还需要用这种无监督学习算法来消除冗余测试用例对故障定位精度的不良影响。首先将上一步中获得的子测试集作为输入，剔除其中的失败用例，将剩下的正确用例放到高斯混合模型中训练。用EM算法求解模型参数之后，将属于同一分布的测试用例聚成一类。然后，在这几类中，按照类平均距离法则，寻找到离刚才剔除出去的失败用例集最近的那一类，将这两类合并，组成最终用来做故障定位的专用测试集。

[0058] 测试用例聚类划分和冗余剔除这两步都运行于Windows环境下，采用了Matlab数学分析软件中的Voicebox工具箱。Voicebox收纳了包括GMM在内的多种概率密度函数。本发明主要采用该工具箱中的gaussmix和gaussmixp函数来进行模型训练和预测。其中，gaussmix函数的使用方法如下：

[0059] function[m,v,w,g,f,pp,gg]=gaussmix(x,c,l,m0,v0,w0)

[0060] gaussmixp函数的使用方法如下：

[0061] function[lp,rp,kh,kp]=gaussmixp(y,m,v,w)

[0062] gaussmix和gaussmixp函数能帮助解决用例划分和冗余剔除的问题，从而找到最针对特定故障的测试套件子集。

[0063] (3)然后，利用基于支持向量机模型的监督学习算法来进行故障定位。由于向量$C_i=(C_{i,1},C_{i,2},...,C_{i,m})$作为测试用例覆盖信息的同时，也能表示为特征空间中的一个数据点。而测试用例的输出r_i即表示为每个样本点所属的类别。因此，把测试用例的覆盖信息C_i当做支持向量机的训练输入，把测试用例的执行结果r_i当做训练输出，以此来训练支持向量机。训练好的模型反映了测试用例的覆盖信息与执行结果之间的非线性映射关系，利用这种关系，间接通过如下的虚拟测试集来找到故障语句。

[0064] $$\begin{bmatrix} C_{t_1} \\ C_{t_2} \\ \vdots \\ C_{t_m} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

[0065] 可以看出，该测试集共有m条测试用例，恰好对应了m条程序语句。其中，第i条测试用例只覆盖第i条程序语句，使得整个覆盖信息表是一个对角矩阵，对角线的值为1，其他为

0。将其放在训练好的支持向量机模型中，以此来预测哪些用例会被分为失败的一类。Vapnik在SVM里提出了最优超平面、软间隔和内积核函数的思想。他实现了让两类样本点，都能够尽量正确地被一个高维超平面分开。且样本点离分类超平面越远，其分为某一类的置信度也就越高。因此，将这些用例在多维空间中距离分类超平面的远近，描述为该语句含有故障的可疑度值，即分类为失败的用例，离分类超平面越远，它覆盖的语句越可能含有故障；分类为成功的用例，离分类超平面越远，它覆盖的语句越不可能含有故障。

[0066] 由于在实际的测试集中，成功用例的数量往往远大于失败用例，因此数据倾斜问题一直困扰着故障定位技术的有效性。考虑到惩罚因子的作用是允许支持向量机错分某些离群点的程度，因此，修改分类超平面的优化目标函数，使之成为：

[0067] $$\min \frac{1}{2}\|\omega\|^2 + C_+ \sum_{i=1}^{p} \varepsilon_i + C_- \sum_{i=p+1}^{n} \varepsilon_i$$
$$s.t., y_i(\omega \bullet x_i + b) \geq 1 - \varepsilon_i, i = 1,...,n \qquad (2)$$

[0068] 这里，i＝1,...,p都是分类为执行成功的样本，i＝p+1,...,n都是分类为执行失败样本，ε_i为松弛变量，n为样本总数。C_+与C_-的取值选择，就可以根据这两类样本数量的反比来确定，也就是说，失败类样本受重视的程度决定于两类间样本数量的差异。改善后的支持向量机模型能有效进行故障定位工作。这里，采用由台湾大学林智仁博士开发的LibSVM工具箱，主要使用的是用于训练支持向量机模型的svm-train.exe程序以及用于预测的svm-predict.exe程序。其中训练方法如下：

[0069] svm-train[options]training_set_file[model_file]

[0070] 之后，构建虚拟矩阵来测试该模型：

[0071] svm-predict[options]test_file model_file output_file

[0072] 这里主要取3个输出:[predicted_label,accuracy,decision_values]，分别表示的预测结果标签、预测准确度和决策值。由这三个值，能够得到一条语句含有故障的可疑度值，并按大小关系排列成表。

[0073] (4)最后，根据可疑度值列表，从上往下依次检测语句，直到故障被定位出来。至此，完成了基于机器学习算法的故障定位方法的全部步骤。

[0074] 为了验证算法的效果，本发明采用了SIR中最为经典的Siemens测试套件和Space测试套件，来对本算法的有效性进行评估。Siemens测试套件包含有7组测试程序，自2003年被应用到NNQ技术的有效性评估上之后，该测试套件已成为故障定位领域内使用最多的测试数据集。Space是由欧洲航天局开发的一款解释器程序。作为一个大型测试集，它包含有38个错误版本，9126行代码，3657条可执行语句以及13585条测试用例。在这两个标准测试套件之上，应用Wong等人提出的EXAM得分体系，来测试本发明的实际定位效果。实验结果表明，由于采用了基于高斯混合模型的无监督学习算法，以及基于支持向量机的监督学习算法，故障定位技术所面临的故障分布不明、测试样本冗余和样本分布不均的问题，得到了有效的解决。从附图2至附图8中可以看出，本发明(图中GVM曲线)比其他方法能更早地完成故障定位工作。并且，在0％到10％的分值区间内，故障定位的效率提升了20％以上，这验证了本发明的确能够有效地提高软件故障定位的精度，降低定位所需的工作量。

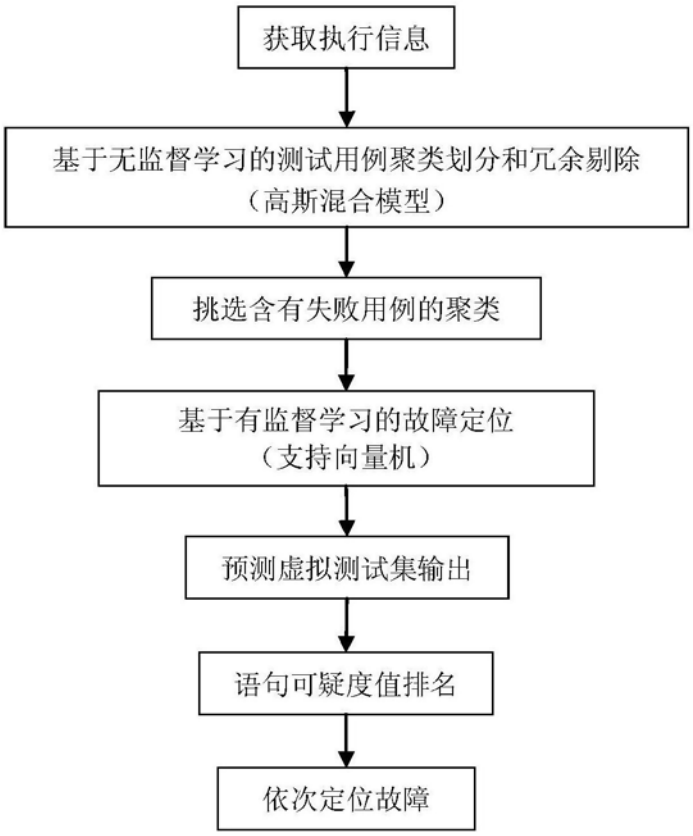

获取执行信息
基于无监督学习的测试用例聚类划分和冗余剔除
（高斯混合模型）
挑选含有失败用例的聚类
基于有监督学习的故障定位
（支持向量机）
预测虚拟测试集输出
语句可疑度值排名
依次定位故障

图1

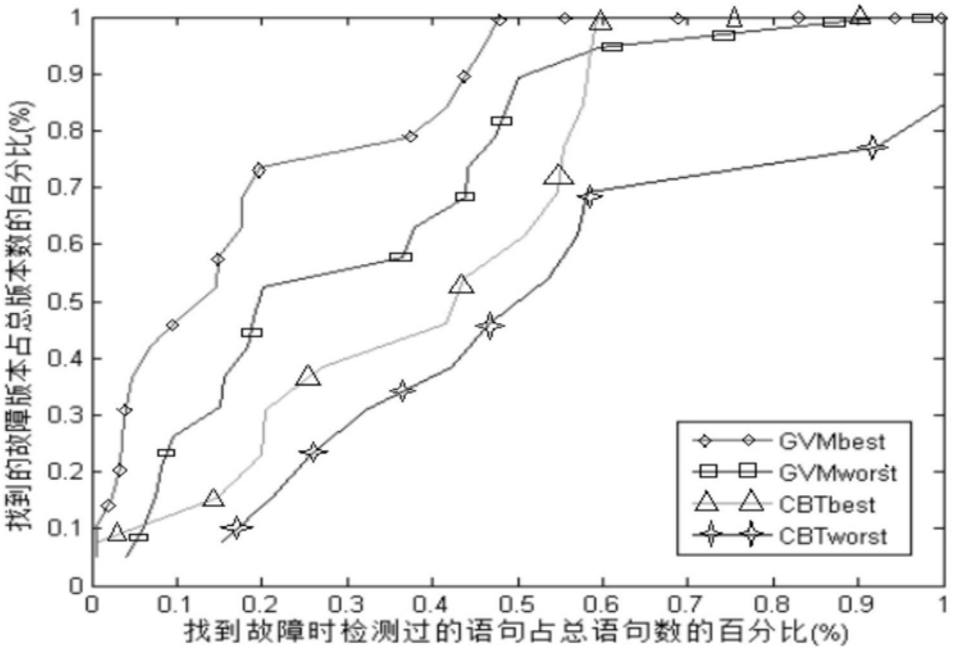

找到的故障版本占总版本数的百分比(%)
找到故障时检测过的语句占总语句数的百分比(%)
GVMbest
GVMworst
CBTbest
CBTworst
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

图2

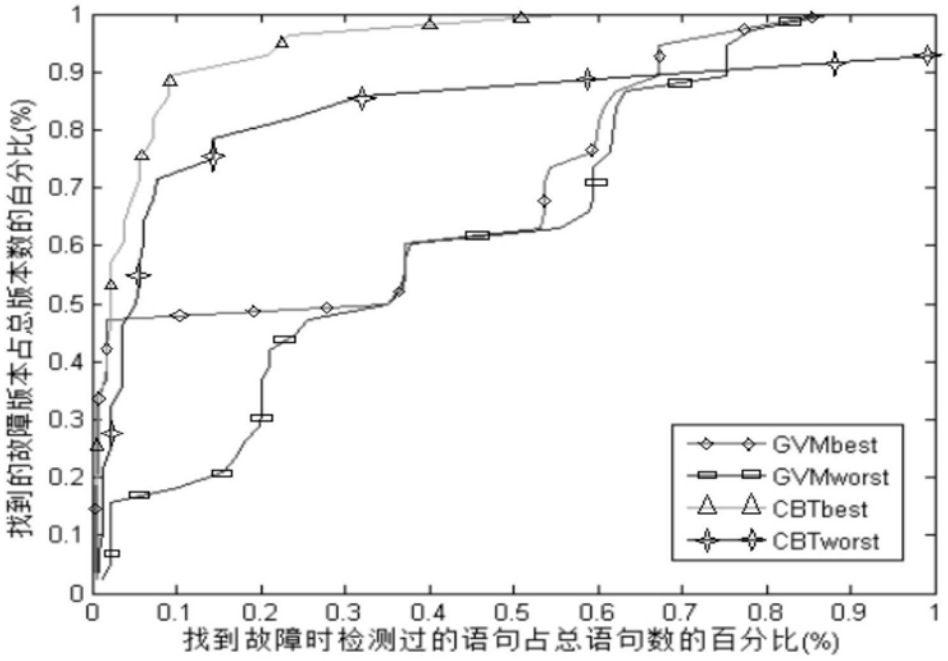

找到的故障版本占总版本数的百分比(%)
找到故障时检测过的语句占总语句数的百分比(%)
GVMbest
GVMworst
CBTbest
CBTworst

图3

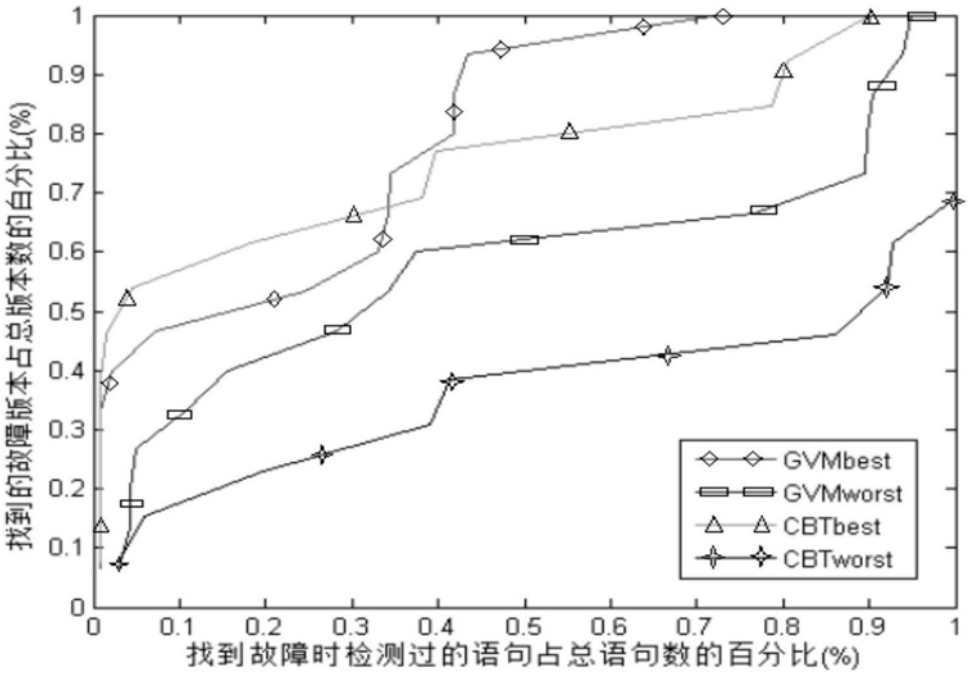

找到的故障版本占总版本数的百分比(%)
找到故障时检测过的语句占总语句数的百分比(%)
GVMbest
GVMworst
CBTbest
CBTworst

图4

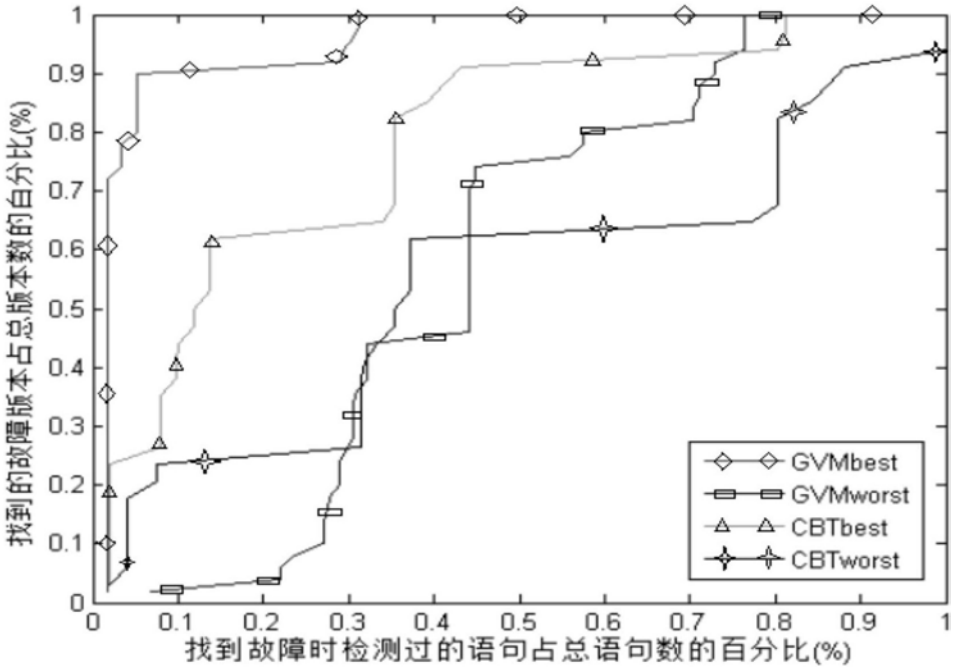

找到的故障版本占总版本数的百分比(%)
找到故障时检测过的语句占总语句数的百分比(%)
GVMbest
GVMworst
CBTbest
CBTworst

图5

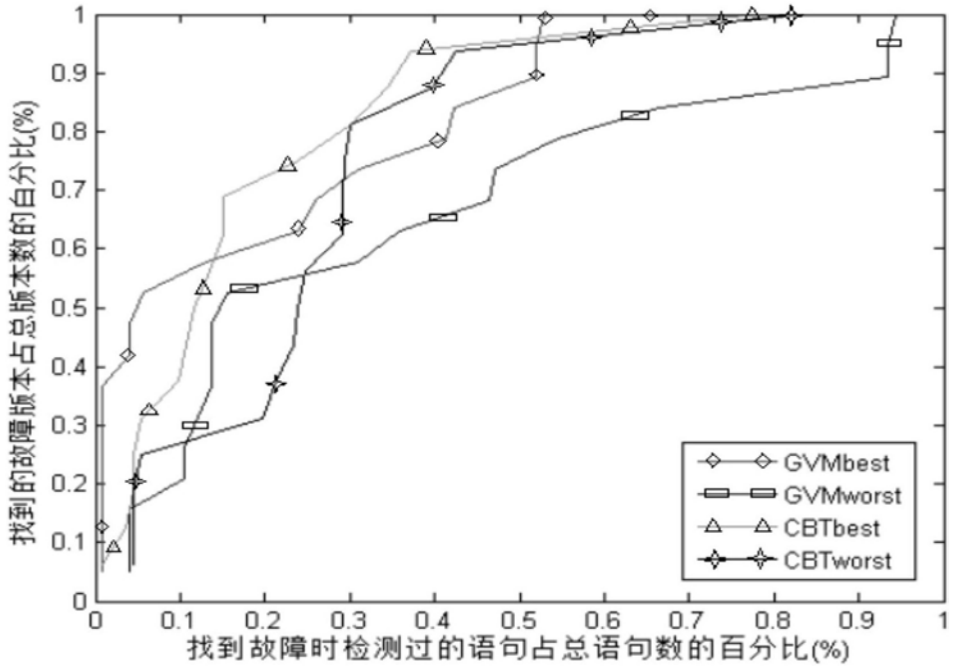

找到的故障版本占总版本数的百分比(%)
找到故障时检测过的语句占总语句数的百分比(%)
GVMbest
GVMworst
CBTbest
CBTworst

图6

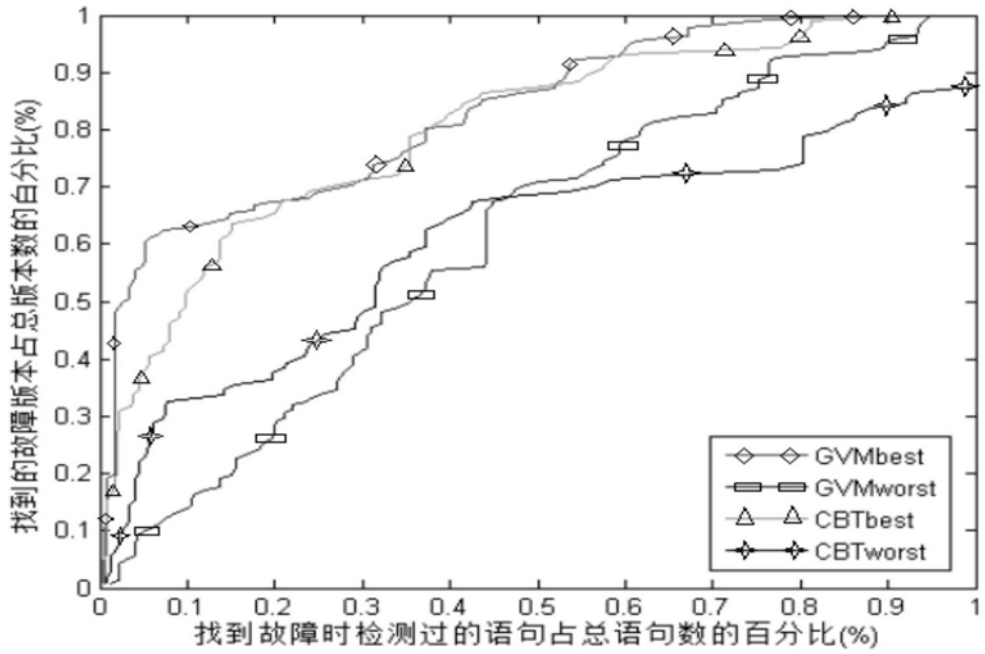

找到的故障版本占总版本数的百分比(%)
找到故障时检测过的语句占总语句数的百分比(%)
GVMbest
GVMworst
CBTbest
CBTworst

图7

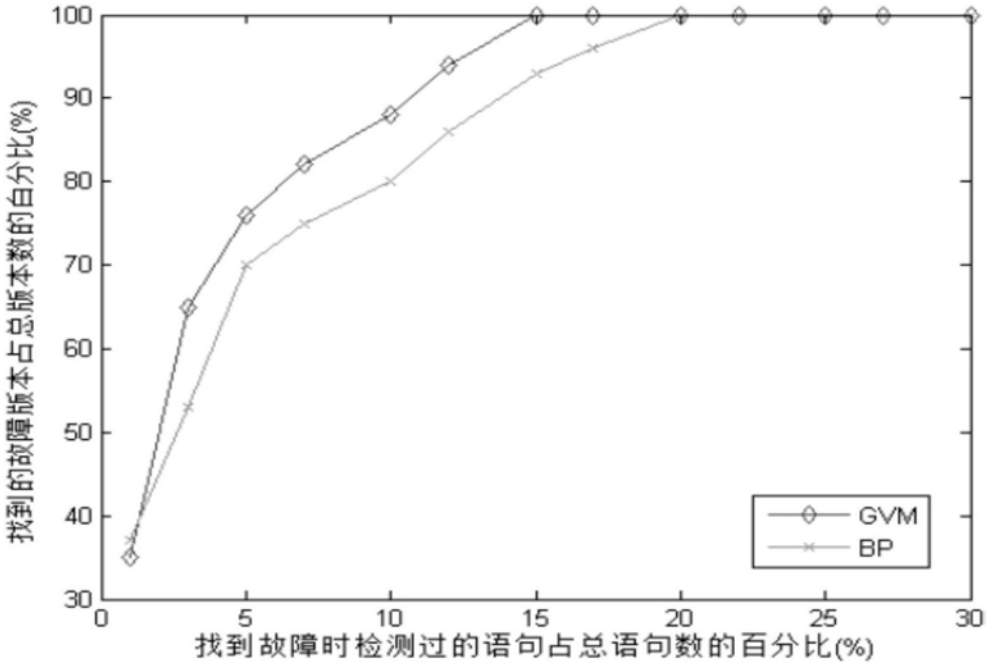

找到的故障版本占总版本数的百分比(%)
找到故障时检测过的语句占总语句数的百分比(%)
GVM
BP

图8