权　利　要　求　书

1、基于频繁子图挖掘的错误定位方法，其特征是将程序的所有实体共同作为一个整体，通过图挖掘方法获取失败测试用例执行中的特征模式，进而实现更加准确且包含运行时上下文的自动化错误定位结果，包括以下步骤：

1）对程序执行频谱进行建模，基于测试用例对程序实体的执行顺序，构建出各测试用例的执行序列图，得到执行序列图集合；

2）根据执行序列图中每个边在所有测试用例的执行序列图中的比例，对执行序列图进行剪枝，移除可疑度较小的边，以提升后续挖掘算法的效率和精度；对剪枝后得到的测试用例的执行序列图集合，使用区分性子图挖掘方法，首先通过频繁子图挖掘算法从失败的测试用例的执行序列图中挖掘出频繁子图，再通过频繁子图的熵值从中确定出对成功和失败的测试用例执行序列图最有区分性的子图，也即最有可能导致程序行为失败的执行模式，由此定位程序中最可能导致测试用例失败的位置。

2、根据权利要求 1 所述的基于频繁子图挖掘的错误定位方法，其特征是步骤 1）对程序执行频谱的建模为：首先对程序插桩，记录程序运行时程序实体的执行顺序信息，执行测试用例池中的成功/失败的测试用例，基于测试用例执行时的程序实体的顺序，构建出成功/失败测试用例的执行序列图集合；最后采用最小深度优先算法来唯一标示一个执行序列图，完成程序执行频谱建模，为下一步处理做准备。

3、根据权利要求 1 所述的基于频繁子图挖掘的错误定位方法，其特征是所述区分性子图挖掘方法为：对于一个包含了执行成功和执行失败的测试用例的测试用例池，测试用例池的熵值 Hc 取决于池中成功的测试用例所占的比例 pSuccess 以及失败测试用例所占比例 pFail，如公式（1）所示：

$$Hc = - pSuccess * log(pSuccess) - pFail * log(pFail) \quad (1)$$

设子图 g 为所有失败的测试用例执行序列图中挖掘出的频繁子图之一，首先依据测试用例是否包含子图 g 将所有执行序列图分为两类：包含子图 g 的执行序列图的集合 S1，以及不包含子图 g 的执行序列图的集合 S2；然后依据公式（1），根据 S1 和 S2 中包含的成功/失败测试用例的比例分别计算出 S1 和 S2 的熵值 Hg1 和 Hg2，再根据 S1 的元素个数占所有测试用例池规模的比例 pS1、S2 的元素个数占所有用例池规模比例 pS2 加权，求出依据子图 g 划分下的测试用例池的的熵值 Hg，如公式（2）：

$$Hg = pS1 * Hg1 + pS2 * Hg2 \quad (2)$$

子图 g 在所有测试用例执行序列图上的划分的信息增益 IG 如下所示：

$$IG = Hc – Hg$$

区分性越强的频繁子图具有越高的信息增益，信息增益越高的子图在程序中的错误执行模式的可疑度越高，在枚举频繁子图的过程中计算信息增益并排列可疑度最高的 K 个频繁子图，提供给软件开发者和测试者错误定位使用。

4、根据权利要求 3 所述的基于频繁子图挖掘的错误定位方法，其特征是如果存在两个可疑度相同的子图 g1 和 g2，即计算得到的信息增益相同，而子图 g1 包含了子图 g2，则在可疑度排序中保留 g2，移除子图 g1。

5、根据权利要求 1 所述的基于频繁子图挖掘的错误定位方法，其特征是步骤 2）中的剪枝为：对于任意一条边 e，e 在失败的执行序列图中出现的频率记为 failed(e)，e 在成功的执行序列图中出现的频率记为 passed(e)；将所有执行过的失败的测试用例总数记为 totalFailed，将所有执行过的成功的测试用例总数记为 totalPassed，则满足以下条件的边将被视为可疑边：

failed(e)/passed(e) > totalFailed / totalPassed

不满足上述公式的边将被从所有的执行序列图中移除。

说　　明　　书

基于频繁子图挖掘的错误定位方法

技术领域

本发明属于计算机技术领域，涉及软件测试技术，其是软件测试中自动化测试技术领域，为一种基于频繁子图挖掘的错误定位方法。

背景技术

自动化错误定位技术主要基于对程序源文件、覆盖信息、执行序列信息、测试结果进行分析，它能够有效辅助开发人员快速定位程序出错位置，提高调试效率。

自动化错误定位技术经过多年在学术界的发展，已经出现了多种不同类别的错误定位技术。该方法主要涉及基于程序动态执行的错误定位技术，该种类型的定位技术主要结合程序执行信息、测试用例执行信息已经以及一部分源程序分析结果，给出程序中可能出错的位置的排序。传统的自动化错误定位技术使用统计方法计算单个程序实体（语句、分支、语句块、方法等）为程序错误的可疑度，进而按照程序实体可疑度由高到低的顺序排序，提供给开发人员或者测试人员作为排查程序错误的优先顺序。

传统的自动化错误定位技术中最主要的一类技术被称为基于程序频谱的错误定位技术。频谱信息，当被插桩的程序执行包含成功测试用例和失败测试用例的用例池时，每个程序实体分别被成功（或失败）测试用例执行到（或未执行到）的次数。以频谱信息为基础，一系列的排序度量被提出用以计算每个程序实体的可疑度，进而排序筛选出可疑度较高的程序实体。一般来说，一个程序实体被失败的测试用例执行的次数越多，被成功的测试用例执行的次数越少，此程序实体为程序失败的可能性越高。

图挖掘程序错误定位方法结合机器学习领域已有技术，将最新的机器学习方法应用于错误定位领域，将程序的所有实体共同作为一个整体，通过图挖掘方法获取失败用例执行中的特征模式，进而实现更加准确且包含运行时上下文的自动化错误定位结果。

发明内容

本发明要解决的问题是：在现有自动化错误定位技术上，结合图挖掘程序错误定位方法，提出一种以频繁子图挖掘方法为基础框架的区分性子图挖掘方法，以更准确得定位程序中的错误。

本发明的技术方案为：基于频繁子图挖掘的错误定位方法，将程序的所有实体共同作为一个整体，通过图挖掘方法获取失败测试用例执行中的特征模式，进而实现更加准

确且包含运行时上下文的自动化错误定位结果，包括以下步骤：

1）对程序执行频谱进行建模，基于测试用例对程序实体的执行顺序，构建出各测试用例的执行序列图，得到执行序列图集合；

2）根据执行序列图中每个边在所有测试用例的执行序列图中的比例，对执行序列图进行剪枝，移除可疑度较小的边，以提升后续挖掘算法的效率和精度；对剪枝后得到的测试用例的执行序列图集合，使用区分性子图挖掘方法，首先通过频繁子图挖掘算法从失败的测试用例的执行序列图中挖掘出频繁子图，再通过频繁子图的熵值从中确定出对成功和失败的测试用例执行序列图最有区分性的子图，也即最有可能导致程序行为失败的执行模式，由此定位程序中最可能导致测试用例失败的位置。

步骤 1）对程序执行频谱的建模为：首先对程序插桩，记录程序运行时程序实体的执行顺序信息，执行测试用例池中的成功/失败的测试用例，基于测试用例执行时的程序实体的顺序，构建出成功/失败测试用例的执行序列图集合；最后采用最小深度优先算法来唯一标示一个执行序列图，完成程序执行频谱建模，为下一步处理做准备。

步骤 2）所述区分性子图挖掘方法为：对于一个包含了执行成功和执行失败的测试用例的测试用例池，测试用例池的熵值 Hc 取决于池中成功的测试用例所占的比例 pSuccess 以及失败测试用例所占比例 pFail，如公式（1）所示：

$$Hc = - pSuccess * log(pSuccess) - pFail * log(pFail) \quad (1)$$

设子图 g 为所有失败的测试用例执行序列图中挖掘出的频繁子图之一，首先依据测试用例是否包含子图 g 将所有执行序列图分为两类：包含子图 g 的执行序列图的集合 S1，以及不包含子图 g 的执行序列图的集合 S2；然后依据公式（1），根据 S1 和 S2 中包含的成功/失败测试用例的比例分别计算出 S1 和 S2 的熵值 Hg1 和 Hg2，再根据 S1 的元素个数占所有测试用例池规模的比例 pS1、S2 的元素个数占所有用例池规模比例 pS2 加权，求出依据子图 g 划分下的测试用例池的的熵值 Hg，如公式（2）：

$$Hg = pS1 * Hg1 + pS2 * Hg2 \quad (2)$$

子图 g 在所有测试用例执行序列图上的划分的信息增益 IG 如下所示：

$$IG = Hc – Hg$$

区分性越强的频繁子图具有越高的信息增益，信息增益越高的子图在程序中的错误执行模式的可疑度越高，在枚举频繁子图的过程中计算信息增益并排列可疑度最高的 K 个频繁子图，提供给软件开发者和测试者错误定位使用。

本发明方法是一种区分性子图挖掘方法，以频繁子图挖掘为基础框架，基于信息增

益方法计算子图的可疑度，通过优化的频繁子图挖掘方法获取失败用例和成功用例执行中的特征模式，进而实现更加准确且包含运行时上下文的自动化错误定位结果。

本发明的有益效果是：改进原频繁子图挖掘算法，原频繁子图挖掘方法定位错误比较粗糙，仅能输出可疑的程序实体，不能对其进行排序，本发明改进后的区分性子图挖掘方法可以获得子图可疑度的排序，进而提供给测试人员可疑度最高的 K 个子图，从而提高了开发人员定位错误的准确性和效率。

附图说明

图 1 为本发明的执行流程图。

图 2 为本发明实施实例的每个测试用例执行的执行序列图，（a）（b）为成功的测试用例，（c）（d）为失败的测试用例。

图 3 为对应图 2 的剪枝之后的执行序列图，（a）（b）为成功的测试用例，（c）（d）为失败的测试用例。

图 4 为本发明实施例中，A 程序的 F-Measure。

图 5 为本发明实施例中，B 程序的 F-Measure。

图 6 为本发明实施例中，C 程序的 F-Measure。

具体实施方式

本发明对成功及失败的测试用例执行序列图集合进行图挖掘，以找到最能区分成功失败测试用例的子图，进而定位错误。

本发明方法以频繁子图挖掘算法为框架，先实现频繁子图挖掘算法，再在其上进行优化，通过在频繁子图挖掘算法基础上加入对成功执行测试用例的分析，比较成功失败测试用例执行图集合，找出最有可能有错误的子图。本发明方法的具体执行流程图见图 1。图 1 中 oracle 类似程序断言，可以标识程序的是 pass 还是 fail。

基于一个测试用例的程序实体的执行顺序，可以构建出测试用例执行的程序实体的执行序列图，本发明由此得到各测试用例的执行序列图，也即图 1 中的调用序列图。具体为：首先对程序插桩，记录程序运行时程序实体的执行顺序信息，执行测试用例池中的成功/失败的测试用例，基于测试用例执行时的程序实体的顺序，构建出成功/失败测试用例的执行序列图集合；最后采用最小深度优先算法来唯一标示一个执行序列图，完成程序执行频谱建模，为下一步处理做准备。

频繁子图挖掘算法将所有支持度大于某个值的子图全部加入结果集合，但其并没有使用到相应的成功的测试用例的图信息，以频繁子图挖掘为基础框架，我们实现了区分性子图挖掘方法。区分性子图挖掘方法的最终目标是在失败的频繁子图的范围内找寻最能区分失败的测试用例图数据集和成功的测试用例图数据集的子图，这些子图被认为是最有可能导致程序失败的原因。

为了定义一个子图的区分性，本发明使用了信息学中熵（entropy）的概念。熵通常被用来度量信息的无序度或多样性。一个集合中元素取值的无序度或多样性越高，通常也就具有越高的熵值。

对于一个包含了执行成功和执行失败的测试用例的测试用例池来说，池中的元素具有成功和失败两种不同的取值，而这个集合的熵值 Hc 取决于池中成功的用例所占的比例 pSuccess 以及失败用例所占比例 pFail，具体计算方法如下所示：

$$Hc = - pSuccess * log(pSuccess) - pFail * log(pFail) \quad (1)$$

每个测试用例都有一个程序执行序列图，设子图 g 为所有失败的测试用例执行序列图中挖掘出的频繁子图之一。为了度量子图 g 对测试用例池中成功和失败两类用例的区分能力，这里首先依据测试用例是否包含子图 g 将所有执行序列图分为两类：包含子图 g 的执行序列图的集合 S1，以及不包含子图 g 的执行序列图的集合 S2。然后根据 S1 和 S2 中包含的成功/失败测试用例的比例分别计算出 S1 和 S2 的熵值 Hg1 和 Hg2，再根据 S1 的元素个数占所有测试用例池规模的比例 pS1、S2 的元素个数占所有用例池规模比例 pS2 加权，求出依据子图 g 划分下的测试用例池的熵值：

$$Hg = pS1 * Hg1 + pS2 * Hg2 \quad (2)$$

最终，*Hc* 和 *Hg* 二者的差异即为子图 g 在所有用例执行序列图上的划分的信息增益 IG（information gain），计算方法如下所示。

$$IG = Hc – Hg$$

由上面的分析可知，子图 g 对成功和失败的用例的划分越精确，则划分之后的集合的熵值 Hg 越小。所以区分性更强的子图倾向于具有更高的信息增益。因此信息增益越大的子图，在程序中的错误执行模式的可疑度越高。本发明使用此度量方法，在枚举频繁子图的过程中计算并排列可疑度最高的 K 个子图提供给软件开发者和测试者错误定位使用。

下面的伪代码算法给出了在频繁子图挖掘算法框架基础上的改进以完成最可疑的子图挖掘：

```
Input: the base graph s together with its container graph set s.GS,
    the threshold of supports minSup
Output: the result discriminative subgraphs S
if s is not minimum DFS code do
   return;
end if
if support(s) < minSup do
   return;
end if
insertIntoTopK(s, S) /* no longer simply "S = S ∪ s" */
extend s with 1 edge growth
for each extended graph ss do
   enumerate s.GS to find ss.GS
   subgraphMining(ss, minSup, S)
end for
```

在本发明上述算法中，每个被确认为频繁子图的子图不是简单地插入结果集合，而是使用一个子过程 insertIntoTopK(s, S)来度量此频繁子图的可疑度，进而确定对前 K 个最可疑的子图列表的添加、删除等操作。

对频繁子图进行筛选，更新前 K 个最可疑的频繁子图的伪代码如下所示：

```
Input: a new frequent subgraph freqGraph,
    existing top K most discriminative subgraphs topK
Output: adding and/or removing elements into/from topK
```

```
for each subgraph s with the same informatin gain as freqGraph do
   if s is subgraph of freqGraph then
      return
   else if freqGraph is subgraph of s then
      topK.remove(s)
   end if
end for
topK.insert(freqGraph)
if topK.size() > capacity then
   topK.removeLastElement()
end if
```

总体来讲，此流程根据每个频繁子图的可疑度由高到低进行排序。由于可能存在某些子图与新增候选的频繁子图可疑值相同并且互为父子图的关系。如果存在两个可疑度相同的子图 g1 和 g2，而 g1 包含了 g2，那么仅需要在排序中保留 g2，因为它提供了更为精准的错误位置和上下文信息，而 g1 将被从结果中移除。上面的排序流程伪代码对此类父子图关系的频繁图进行了筛选，并且在算法最后移除过多的元素，以防列表长度超出最大容量。

下面用一个具体的实施例说明本方法的步骤，直观展示本发明的动机和效果，我们构造如下被测程序片段作为示例。下面的程序片段将返回数组 arr 中第一次出现 target1 或者 target2 的下标。但是程序第 6 行和第 10 行分别包含一个 bug：程序本应该在第 5 行或者第 9 行出跳出循环。这样，当程序在同时执行到第 5 行和第 9 行时会产生失败结果。

```
1:  int findFirstOccurance(int arr[], int size, int target1, int target2) {
2:      int index = -1;
3:      for(int i = 0; i < size; i++) {
4:          if(arr[i] == target1) {
5:              index = i;
6:              //Bug: should be a break here.
7:          }
```

```
8:              if (arr[i] == target2) {
9:                  index = i;
10:                 //Bug: should be a break here
11:             }
12:         }
13:         return index;
14: }
```

假设我们拥有以下测试数据（表 1），表格中列出了每个用例的参数列表，以及每个用例在此程序片段上执行的结果：成功还是失败。

表 1

编号	arr	target1	target2	结果
1	{1, 2}	1	3	成功
2	{1, 2}	2	3	成功
3	{1, 2}	1	2	失败
4	{1, 2}	2	1	失败

对每个用例，得到测试用例语句执行序列，如表 2。

表 2

编号	语句执行序列	结果
1	{1, 2, 3, 4, 5, 7, 8, 11, 12, 3, 4, 7, 8, 11, 12, 3, 13}	成功
2	{1, 2, 3, 4, 7, 8, 9, 11, 12, 3, 4, 7, 8, 11, 12, 3, 13}	成功
3	{1, 2, 3, 4, 5, 7, 8, 11, 12, 3, 4, 7, 8, 9, 11, 12, 3, 13}	失败
4	{1, 2, 3, 4, 7, 8, 9, 11, 12, 3, 4, 5, 7, 8, 11, 12, 3, 13}	失败

我们将每个用例的序列构建为语句的执行序列图，如图 2(a)-(d)。

得到用例的执行序列图之后，根据图中每条边在成功/失败的执行序列图中出现的频率评估边是否可疑，并基于此进行图的剪枝，移除可疑度较小的边，这里剪枝的实施根据实际情况来设定。例如。对于任意一条边 e，e 在失败的执行序列图中出现的频率记为 failed(e)，e 在成功的执行序列图中出现的频率记为 passed(e)；同时我们将所有执行过的失败的测试用例总数记为 totalFailed，将所有执行过的成功的测试用例总数记为 totalPassed，那么满足以下条件的边将被视为可疑边：

failed(e)/passed(e) > totalFailed / totalPassed

不满足上述公式的边将被从所有的执行序列图中移除，以缩小图挖掘算法的搜索空间，同时提高挖掘结果的精度。剪枝之后的执行序列图如图 3(a)-(d)所示。

最终，通过区分性子图挖掘方法找出最能区分成功和失败的执行序列图的子图，也

即在失败的执行序列图中频度较高而在成功的执行序列图中频率较低的子图。根据上述算法可以计算出在此例子中前 K 个最可疑的子图，由于数据规模较小，我们不难发现经过执行模式：7->8 的子图是被所有失败的执行序列图包含，而不被任何成功的执行序列图包含的最小子图。因此这个频繁子图将会被作为可疑的子图列表中可疑值最高的一个提供给开发者帮助进行错误定位和修正。

下面用一些真实的测试程序来展示基于信息增益的图挖掘技术的实际效果，验证本发明的方法具有更好的效果。

实验程序

测试对象程序	版本数	代码行数	基本块数目	测试用例数目
A	5	284	24	2650
B	5	255	29	2710
C	6	512	57	5542

本次实验使用 3 个程序：A、B、C，每个程序包含多个版本，每个版本包含一个 bug。每个程序包含的代码行数、基本块的数目以及相应的测试用例数据都罗列在表中。同时为了度量算法的效果，本发明实施例使用 F-Measure 来评定准确度和召回率的总体优劣情况，计算 F-Measure 的公式为：

*F-Measure=2*precision*recall/(precision+recall)*

其中 precision 为精度，指的是算法返回的排名最高的 K 个子图的列表中包含的能够指示 bug 的子图的比例。recall 是召回率，指的是程序多个 bug 版本，能够通过本算法输出的排名最高的 K 个子图的列表检测出的 bug 所占的比例。

由于以上三个度量会随着排名最高的列表长度 K 的不同而变化，因此本发明在实验中对 K 取 1、3、5、7、10、15 这几个不同取值，得出 F-Measure 随着 K 变化的曲线。结果如图 4-图 6 所示。其中方格图标的线（F-Measure w/IG）表示使用本发明信息增益的挖掘方法的结果，圆点图标的线（F-Measure w/o IG）表示未使用信息增益挖掘方法的结果。这三张图展示了这两种方法在三个程序上的综合表现 F-Measure 的对比。通过实验结果的对比可以发现，本发明方法比之未使用信息增益方法的错误挖掘的综合效果要有明显的提升。

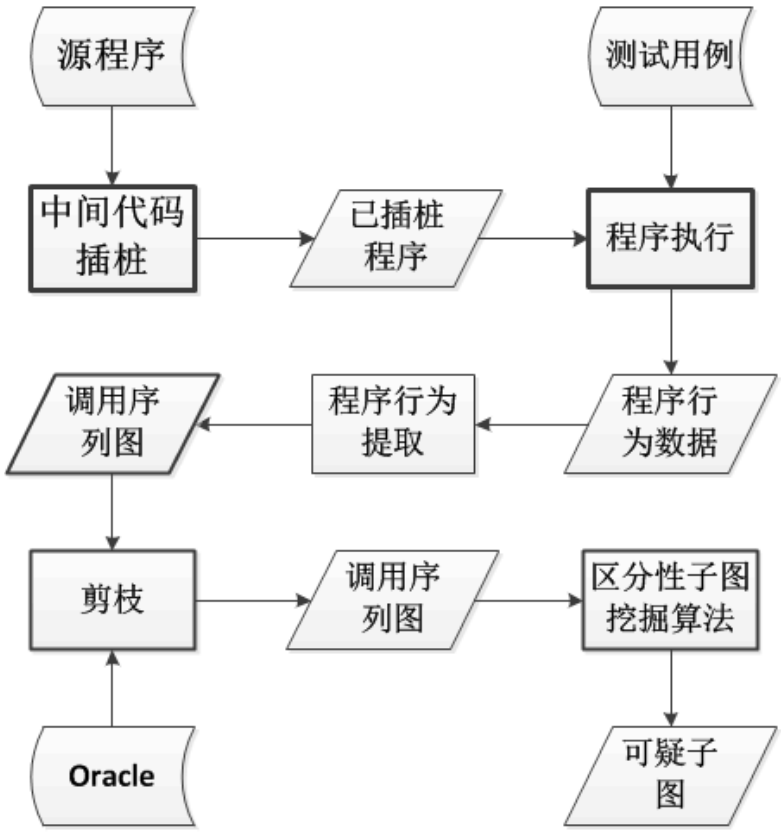
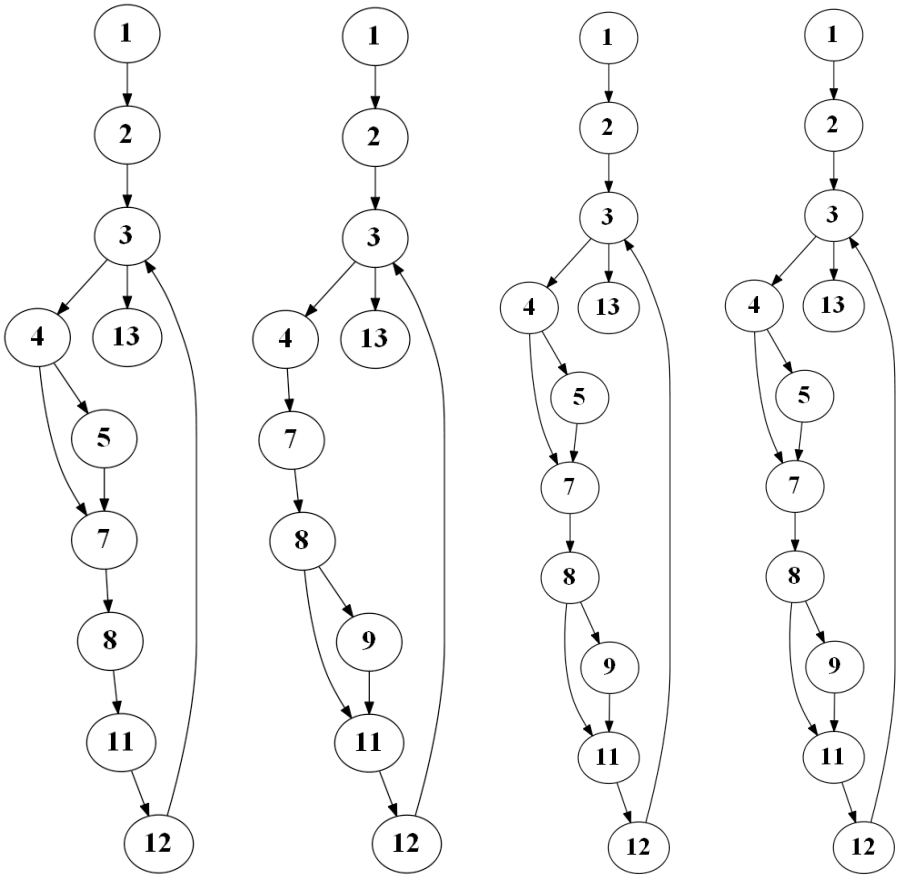

源程序
测试用例
中间代码
插桩
已插桩
程序
程序执行
调用序
列图
程序行为
提取
程序行
为数据
剪枝
调用序
列图
区分性子图
挖掘算法
Oracle
可疑子
图

图 1


1
2
3
4
13
5
7
8
11
12
1
2
3
4
13
7
8
9
11
12
1
2
3
4
13
5
7
8
9
11
12
1
2
3
4
13
5
7
8
9
11
12

（a）　（b）　（c）　（d）

图 2

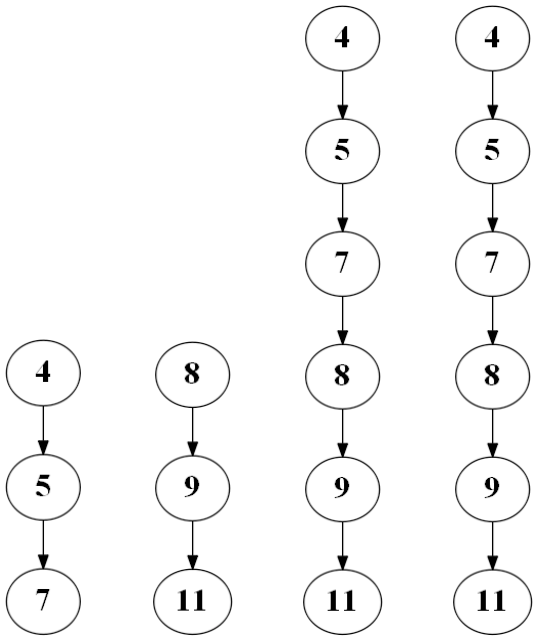

4
5
7
4
8
9
11
4
5
7
8
9
11
4
5
7
8
9
11
（a）
（b）
（c）
（d）

图 3

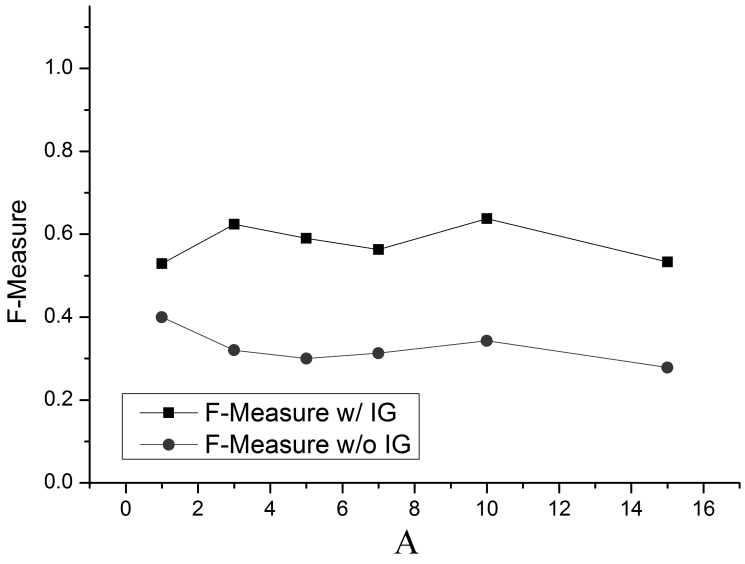

F-Measure
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14
16
A
F-Measure w/ IG
F-Measure w/o IG

图 4

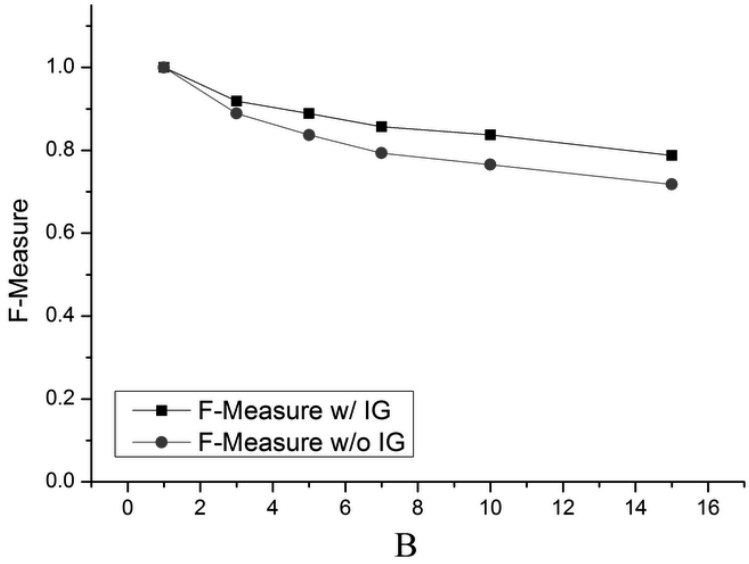

F-Measure
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14
16
F-Measure w/ IG
F-Measure w/o IG
B

图 5

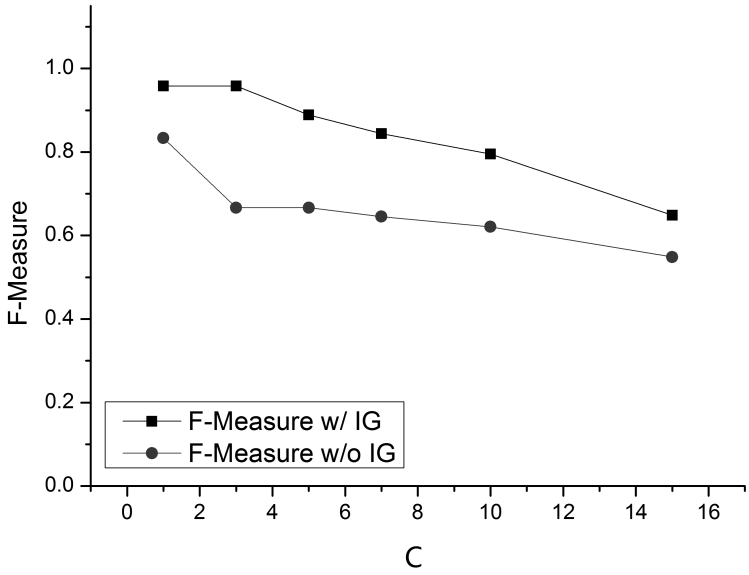
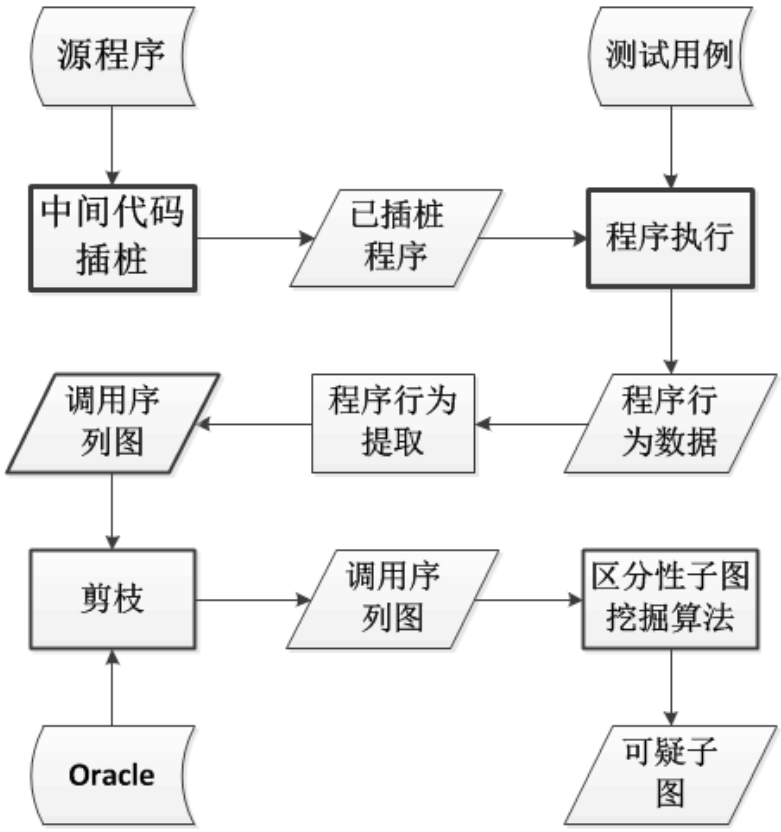

F-Measure
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14
16
F-Measure w/ IG
F-Measure w/o IG
C

图 6

说　明　书　摘　要

基于频繁子图挖掘的错误定位方法，将程序的所有实体共同作为一个整体，通过图挖掘方法获取失败用例执行中的特征模式，进而实现更加准确且包含运行时上下文的自动化错误定位结果。本发明方法是一种区分性子图挖掘方法，以频繁子图挖掘为基础框架，基于信息增益方法计算子图的可疑度，通过优化的频繁子图挖掘方法获取失败用例和成功用例执行中的特征模式，进而实现更加准确且包含运行时上下文的自动化错误定位结果。本发明可以获得子图可疑度的排序，进而提供给测试人员可疑度最高的 K 个子图，从而提高了开发人员定位错误的准确性和效率。

摘　　要　　附　　图


源程序
测试用例
中间代码
插桩
已插桩
程序
程序执行
调用序
列图
程序行为
提取
程序行
为数据
剪枝
调用序
列图
区分性子图
挖掘算法
Oracle
可疑子
图