第 33 卷 第 9 期 计 算 机 工 程 2007 年 5 月
Vol.33 No.9 Computer Engineering May 2007
·博士论文· 文章编号：1000—3428(2007)09—0043—03 文献标识码：A 中图分类号：TP311

基于双轨迹差异分析法的软件故障定位

刘彦斌，朱小冬
（军械工程学院装备指挥与管理系，石家庄 050003）

摘 要：软件运行监控器监测出故障之后，软件故障定位非常困难。该文提出了双轨迹差异分析法，根据成功的运行（run）和含有故障的运行之间的差异来进行故障定位。它采用程序谱来抽象表达程序执行轨迹，按照编辑距离度量来选取和含有故障的运行最近邻居的成功运行。通过序列间的最长共同子序列和最大稳定子序列集的计算，最终得到导致成功运行和失效运行之间差异的可疑故障语句集，并把它作为故障原因。经实验验证，该方法大大减少了故障定位中代码审查的范围。

关键词：故障定位；轨迹；编辑距离；最大共同子序列；运行监控

Fault Localization with Difference Analysis of Dual-traces

LIU Yanbin, ZHU Xiaodong
(Dept. of Command & Technology of Equipment and Management, Ordnance Engineering College, Shijiazhuang 050003)

【Abstract】 Software fault localization is always very difficult, when software fault is detected by runtime monitor. A method to fault localization with difference analysis of dual-traces is proposed. It locates software fault by analyzing the differences between successful run and faulty run. It abstracts program execution with spectrum and selects nearest neighbor successful run with distance metric. By computing longest common subsequence and maximal common adjacent subsequence set, suspicious fault location set is got finally, which is treated as fault cause. This method will reduce code scope of checking for fault localization dramatically.

【Key words】 Fault localization; Trace; Edit distance; Longest common subsequence; Runtime monitor

在软件运行监控技术中，插装后的程序在运行中将不断发送程序运行信息到监控器。监控器接收信息、观察并报告目标系统行为是否和系统需求一致。当监控器发现软件故障之后，故障定位器进行软件故障定位。对于于大部分程序而言，由于缺陷位置和程序失效之间的复杂因果关系，判定那一块代码引起失效是很困难的事情，即使通过监控技术得到了软件的含有故障的执行轨迹，也很难诊断出精确的故障位置，主要原因在于：(1)需要审查的程序规模常常非常庞大；(2)避免同样的错误后果，可能存在多种程序修改方案；(3)错误可能是分散的代码片。

目前，利用人工方式通过调试器进行故障定位仍然占据主导地位，故障定位能力常常依赖于人员的知识和经验。然而，故障定位不能仅仅依赖于那些不能转化为算法的纯粹的心理概念，而应当从程序中发现因果关系。本文采用双轨迹差异分析法，根据含有故障的运行和与其相似的成功的运行之间的差异来缩小可疑故障位置空间，辅助进行故障定位。

轨迹是能够收集的关于运行的所有信息，双轨迹差异分析法需要构建成功运行的轨迹库。它可以从两种途径获得：(1)根据白盒测试中测试用例和测试模拟环境中获得的数据，经过格式转换后获得，称为实验成功轨迹。(2)从实际运行监控器获得的，称为历史成功轨迹。

故障定位器常常不依赖于输入结构知识，而依赖于程序中能够和源码某个位置关联的程序执行特征，这些特征的整体可被称为程序谱(spectrum)。程序谱特征化程序行为、或者提供了程序行为的信号，如路径谱、分枝谱等，它可为故障定位提供重要知识。本文采用的程序谱包括程序基本块轨迹谱、距离谱等。

双轨迹差异分析法采用程序谱来抽象表达执行轨迹，按照一定的距离准则(Distance Criterion)，选取和故障的运行相似的正确的运行。通过比较两个程序谱的差异，产生程序中的可疑部分。并认为不产生差异的部分不是故障原因，从而减少了故障定位中代码审查的范围。其中如何根据程序谱来度量轨迹之间的距离并选取最近距离轨迹是该方法的关键。

1 程序运行的抽象表达

监控器从程序执行中收集到的数据充当程序谱，从谱中很容易得到每次运行的函数或者基本块之间的调用序列。

基本块定义为：一个语句序列中，控制流从开始进入，到结束时离开，期间没有停顿或者其他可能分支，直到结束。如果把基本块作为节点，基本块之间的转移作为边(函数调用或者分支，分别对应基本块为函数或者分支的情况)，就构成了基本块控制依赖图。

程序的一次运行可以抽象为通过该图的一条路径，节点报告了经历过的节点，边报告了遍历过的边。从而，可把程序执行看作基本块之间的组合，一次运行抽象为基本块构成的一个字符串序列。令程序P具有K个基本块，每个基本块是字母表Σ的一个元素，则含有故障的运行可以表达为$A=a_1a_2\ldots a_m$；成功的运行是$B=b_1\,b_2\ldots\,b_n$；其中，A、B的字都来自大小为K的子母表Σ。从而程序运行的相似性，可以转换为字符串序列间的相似性度量问题。

基金项目：国防预研基金资助项目
作者简介：刘彦斌(1978－)，男，博士生，主研方向：软件保障；朱小冬，教授
收稿日期：2006-06-21 **E-mail：**liuyan0203@sina.com

基于基本块的运行抽象表达既满足了精确性，又具有相对简单的优势，还能导致很清晰的距离度量。

2 程序运行的距离度量

为了选取最相似的成功运行，可以采用距离谱对程序运行进行距离度量。距离谱是一种提供距离操作的谱，通过距离操作实现相似性度量。该相似性测度的根本思想是：两序的相似性反映为把其中一个序列转换成另一序列的度量，也就是说，如果两序列很相似，则其一序列应该很容易转换成另一序列。同时，采用的距离定义不同，得到的程序相似性也有很大差异。

汉明距离(Hamming Distance)定义为两个字符串中位不同的数目，一般用来比较同样长度字符串的差异。但是，它通常仅仅能够把 X 中的第 i 个字符和 Y 中的第 i 个子母进行比较，不能很好地用来表达程序运行之间的形似性。例如，对于 X=DSDSSDAM，Y= MDSDSSDA，汉明距离 D(X,Y)=8，说明两个串之间相似性很小。实际上，这两个字符串之间仅仅调换了一个子母位置，具有很大的相似性，而编辑距离(edit distance)定义随机长度的字符串之间的不同，能够有效地克服汉明距离的某些缺陷。

Levenshtein 引入的编辑距离，根据两个字符串之间把一个字符串转换为另一个的最小基本操作数(插入、删除、置换)来度量相似性，它可以用来比较 X 中第 i 个字符和 Y 中的第 j 个字符，对于同样的 X=DSDSSDAM；Y= MDSDSSDA，编辑距离 D(X,Y)＝2，能够很好地体现两个字符串之间的相似性，本文选用编辑距离进行相似度度量，从而程序运行的距离度量(ditance metric)转换为编辑距离问题。

定义 1 任意给定两个串 X 和 Y，那么串 X 和 Y 之间的编辑距离 D(X,Y)定义为使用如下 3 种编辑操作将串 A 转换成串 B 所需的最小成本：(1)从串 X(或者 Y)中删除一个符号(字符)；(2)向串 X(或者 Y)插入一个符号；(3)用另一个符号替换串 X(或者 Y)中指定的某个符号。

性质 两个串 X 和 Y 之间的编辑距离 D(X,Y)具有如下性质：(1)非负性：D(X,Y)≥0；(2)零性：D(X,Y)=0 当且仅当 X=Y；(3)对称性：D(X,Y)=D(Y,X)；(4)三角不等式：设 Z 为另一个串，则 D(X,Y)≤D(X,Z)+D(Z,Y)。

令 $\wedge$ 为空字符，编辑操作是 $a \to b$，则如果 $a \neq \wedge$ 且 $b \neq \wedge$，$a \to b$ 是替换操作；如果 $b \neq \wedge$，$a \to b$ 是删除操作；如果 $a \neq \wedge$，$a \to b$ 是插入操作。通常，如果 a=b，令从 a 转换为 b 成本值为 0；如果 $a \neq b$，令插入或删除操作的成本值为 1，替换的成本值为 2。

令A(i)是字符串A的第i个字符，｜A｜表示字符串A的长度(字符个数)，γ 是对于每个编辑操作的非负的任意费用函数，则A_i到B_j的最短编辑距离可以分解为 3 个子题：求A_{i-1} 到B_{j-1}，A_i 到B_{j-1}，A_{i-1} 到B_j的最短编辑距离，通过修改，增加，删除一个字符得到；从这 3 种方式中选择编辑距离最短的一个就是A_i到B_j的最短编辑距离。即

D(i，j)＝min{

$m_1 := D[i-1, j-1] + \gamma(A(i) \to B(j))$，　//对应替换操作

$m_2 := D[i-1, j] + \gamma(A(i) \to \wedge)$，　//对应删除操作

$m_3 := D[i, j-1] + \gamma(\wedge \to B(j))$　//对应添加操作

}

通过递归计算编辑距离的算法如算法 1 所示，该时间复杂性是 O(|A|*|B|)。根据该算法，得到的成功轨迹库中和故障轨迹距离最小的运行，可以作为最相近的用来进行比较的成功运行，也有文献称之为最近邻居模型(nearest neighbor model)，思路和本文是一致的。

算法 1 编辑距离计算

输入 含有故障的轨迹 A，和成功运行轨迹库中提取的轨迹 B。

输出 序列 AB 之间的编辑距离 D(A，B)。

(1)D[0, 0] := 0;

(2) for i := 1 to |A| do D[i, 0] := D[i-1, 0] + $\gamma(A(i) \to \wedge)$;

(3) for j := 1 to |B| do D[0, j] := D[0,j-1] + $\gamma(\wedge \to B(j))$;

(4) for i := 1 to |A| do

(5) for j := 1 to |B| do begin

(6) $m_1 := D[i-1, j-1] + \gamma(A(i) \to B(j))$;

(7) $m_2 := D[i-1, j] + \gamma(A(i) \to \wedge)$;

(8) $m_3 := D[i, j-1] + \gamma(\wedge \to B(j))$;

(9) D[i, j] :=min(m_1, m_2, m_3);

(10) end;

3 最长共同子序列和最大稳定连续子序列集

程序运行的距离度量从序列的差异性方面进行了度量，也可以根据序列间的交叉程度(即相同性)作为其相似性的测度标准，通过求解两条路径序列中的最长共同子序列(Longest Common Subsequence, LCS)的方法来研究序列的距离。LCS 问题在遗传工程、数据压缩、编辑错误纠正等方面广泛应用，本文采用 LCS 主要用来计算和故障定位密切相关的最大稳定连续子序列集。

令字符串 $C = c_1 c_2 ... c_q$ 是A=$a_1a_2…a_m$的子序列，所谓子序列就是将一个序列中的一些(可能是零个)字符去掉所得到的序列，例如：abcd、den、enk等都是“ abcdefnk ”的子序列。如果C既是A的子虚列，同时也是B的子序列，则称C是A和B的共同子序列。 如果C在A和B的共同子序列中具有最大长度，则称字符串C是A和B的最长共同子序列。在LCS中，子序列不必连续出现于 A 或 B，只要出现的顺序和 A 及B 内出现的顺序相同即可。如在两个程序运行序列中，序列A为acagccatgagtcagtaaat，序列B为agagccgtgaggtcagtaa，则LCS为aagcctgagtcagtaa。

LCS求解可以描述为：给定长度为S的字母表 Σ_s，及其字符串A=$a_1a_2…a_m$和B=b_1 $b_2…b_n$，求解字符串 $C = c_1 c_2 ... c_q$，使得C是A和B的LCS。LCS问题已经通过采用递归的方式来解决，对于输入长度为m和n的字符串，这些算法通常需要执行时间是 O(mn)。利用LCS问题和编辑距离问题的相关性，从算法 1 中数列D中存储的信息取得LCS的算法如下(算法 2)，该算法的输出为一个四元组，OutPut＝{Key,Value,I,j}，其中q是LCS的字符长度，Key表示LCS中每个字符的序号，$Key \in \{1...q\}$；Value表示LCS每个字符的数值；i，j分别表示当前Value在字符串A和B中的位置序号。如某个输出变换后如表 1。

表 1 算法 1 转换得到的某个 LCS 输出结构

key	Value	i	j
1	b	2	4
2	d	4	7
3	a	5	8
4	d	6	9
5	e	7	10
6	f	9	11
7	g	12	14
8	f	13	15
9	b	14	16

从表 1 中 Value 列显然可以看出，LCS=bdadefgfb。

对于双轨迹差异分析而言，计算 LCS 的目的是为了缩小

维护人员检查程序的可疑位置空间。由于对于程序而言，最可能的故障位置是双轨迹之间的差异及其产生差异的函数调用或者条件转移等原因，为了尽可能防止在双轨迹差异分析中去除故障位置，需要排除的仅仅是那些最不可能含有故障的位置，即本文定义的稳定连续子序列含有的位置集合。

定义 2 最大共同连续子序列集(Maximal common adjacent subsequence set，MCDSS)指由 LCS 中那些字符连续出现于 A 和 B 的顺序出现的共同子序列构成的集合。即如果 A=a1a2…am 和 B=b1b2…bn，则 MCDSS=$\{S_1,S_2,...,S_t\}$,MCDS 的每个元素 $S^p_i \subseteq LCS$，且 $S^p_i = a_j a_{j+1}...a_{j+p-1} = b_k b_{k+1}...b_{k+p-1}$，$1 \le i \le t$，p 是 S_i 的长度。

定义 3 最大稳定连续子序列集(Maximal Stable Adjacent Subsequence Set, MSASS)：指去除最大共同连续子序列中每个元素首尾所得到的非空序列段。若 MCDSS=$\{S_1,S_2,...,S_t\}$，则 MSASS＝$\{V_1,V_2,...,V_t\}$，每个元素 $V^{p-2}_i = a_{j+1}...a_{j+p-2} = b_{k+1}...b_{k+p-2}$，$1 \le i \le t$，p－2 是 V_i 的长度。

对于上文的例子，序列 A 为 acagccatgagtcagtaaat，序列 B 为 agagccgtgaggtcagtaa。根据 LCS：aagcctgagtcagtaa，很容易得到 MCDSS：agcc，tgag，tcagtaa；从每个 MCDSS 分别去除首尾得到 MSASS 为 gc、ga、cagta。这些 MSASS 就是在双轨迹差异分析法中可从可疑位置集中排除的轨迹段。

算法 2 LCS 计算

输入 含有故障的轨迹 A，和与之最近似的具有最小编辑距离的轨迹 B。

输出 序列 A 和 B 的 LCS。

(1)i := |A|; j := |B|;q=1;
(2)while($i \neq 0$ & $j \neq 0$) do
(3) if D[i, j] := D[i-1, j] + $\gamma(A(i) \to \wedge)$ then i:= i-1;
(4) else if D[i, j] := D[i,j-1] + $\gamma(\wedge \to B(j))$ then j :=j-1;
(5) else begin
(6) if (A(i)=B(j)) then print(q,A(i),i, j); q:＝q+1;
(7) i : = i -1 ; j : = j - 1 ;
(8) end;

一般而言，MSASS 可以通过对输出矩阵进行矩阵变换得到，如对于表 1 输出可以得到输出矩阵如下：

$$\begin{bmatrix} 1 & b & 2 & 4 \\ 2 & d & 4 & 7 \\ 3 & a & 5 & 8 \\ 4 & d & 6 & 9 \\ 5 & e & 7 & 10 \\ 6 & f & 9 & 11 \\ 7 & g & 12 & 14 \\ 8 & f & 13 & 15 \\ 9 & b & 14 & 16 \end{bmatrix}$$

对该矩阵进行如下操作：从第 2 行开始依次减去上一行数值，如果得到的结果中 3、4 列均为 1，则保留上一行数值到 MCDSS 元素矩阵中。如第 2 行减去第 1 行得到[1 d-b 2　3] 3、4 列不均为 1,则不保留。第 3 行减去第 2 行得到[1 a-b 1 1] 3、4 列均为 1，则保留第 2 行，同样可以保留 3、4、5 行，从而得到 MCDSS 的第 1 个元素：

$$S_1 = \begin{bmatrix} 2 & d & 4 & 7 \\ 3 & a & 5 & 8 \\ 4 & d & 6 & 9 \\ 5 & e & 7 & 10 \end{bmatrix}$$

相应地得到 MSASS 的第 1 个元素：

$$V_1 = \begin{bmatrix} 3 & a & 5 & 8 \\ 4 & d & 6 & 9 \end{bmatrix}$$

依次进行循环，可以得到 MCDSS 和 MSASS 的所有元素值。

把上述计算过程，加以推广和改进，也可以直接输出 MSASS 的元素值。令序列 A 和 B 的 LCS 输出四元组为 $q \times 4$ 元矩阵 Matrix：

$$\begin{bmatrix} 1 & value_1 & i_1 & j_1 \\ 2 & value_2 & i_2 & j_2 \\ . & . & . & . \\ q & value_q & i_q & j_q \end{bmatrix}$$

其中，每行为 ROW_k，$1 \le k \le q$；MSASS＝$\{V_1,V_2,...,V_t\}$，每个元素为矩阵 V_x，$x \in [1...t]$；则算法 3 伪代码如下：

算法 3 MSASS 计算过程

输入 序列 A 和 B 的 LCS 输出 4 元组。

输出 序列 A 和 B 的 MSASS。

(1)IsDeleted=TRUE; x=0;
(2) for k=1 to q-1 do
(3) if ($i_{k+1} - i_k = j_{k+1} - j_k = 1$)
(4) if (IsDeleted=FALSE) Add ROW_k to V_x;
(5) else x=x+1; IsDeleted=FALSE;
(6) else IsDeleted=TRUE;
(7) end;

根据算法 3，可以直接得到 MSASS 的所有元素值。

4 双轨迹差异分析算法

定义 4 可疑轨迹段：从故障序列 A 中去除最大稳定连续子序列后并经过源码匹配后得到的可疑语句集。

故障定位器中的双轨迹差异分析法包括轨迹抽象、距离度量、LCS 求解、MSASS 求解、源码匹配、可疑轨迹段输出等几个过程，可以参见双轨迹差异分析算法 4。

算法 4 双轨迹差异分析法

输入 从运行验证算法中获得的含有故障的轨迹，和成功运行轨迹库。

输出 故障轨迹中的可疑轨迹段。

步骤如下：

(1)根据相似度，搜索轨迹库，采用最近邻模型 选取相似轨迹。

(2)求故障轨迹谱和最近邻居谱的最长共同子序列。

(3)从最长共同子序列中求出最大稳定连续子序列 D。

(4)从故障轨迹谱中去除最大稳定连续子序列集中的所有元素，得到可疑轨迹谱 K＝A－D。

(5)把可疑轨迹谱转换为程序源码位置的集合，即差异语句集，输出可疑轨迹段。

5 结论

本文提出的双轨迹差异分析法，根据含有故障的运行和与其相似的成功的运行之间的差异来缩小可疑故障位置空间。该算法不需要任何关于程序输入和故障的知识，仅仅需要监控器隔离出含有故障的运行，并且能够很好地重用软件测试阶段的信息，从而充分利用了被诊断对象的各种相关知识进行故障定位。经实验验证，该方法对于故障后的代码审查规模平均缩小 45%以上，大大减少了故障定位的效率。

参考文献

1 Reps T, Das, Larus J. The Use of Program Profiling for Softw are Maintenance with Applications to the Year 2000 Problem[C] //Proceedings of the 6th European Software Engineering Conference. 1997-09: 432-449.

（下转第 48 页）