# 基于 Token 的结构化匹配同源性代码检测技术研究*

刘云龙
(工业和信息化部计算机与微电子发展研究中心, 北京 100048)

**摘 要**: 对于结构性代码变换,基于 Token 的同源性检测技术缺乏抽象提取手段,难以识别和定位结构化信息。为此,针对代码同源性检测结构化匹配进行了研究,在 LCS(longest common sequence)算法中融入了跳变信息保留、结构边界划分、窗口搜索、计数重置、有效序列界定等逻辑,用于 Token 摘要的结构化信息匹配,提出了一种结构化匹配同源性代码检测技术,并通过实际工程代码样本进行多种场景的实验测试。实验表明,该方法能够高效识别代码结构化信息,对于代码同源性检测是有效的。

**关键词**: 同源性检测; 公共子序列; 结构化信息; 代码变体
**中图分类号**: TP391 **文献标志码**: A **文章编号**: 1001-3695(2014)06-1841-05
doi:10.3969/j.issn.1001-3695.2014.06.057

# Token-based structured code matching homology detection technology

LIU Yun-long
(*Research Center for Computer & Microelectronics Industry Development*, *Beijing* 100048, *China*)

**Abstract**: For the structural code conversion, Token-based homology detection technology lacks of abstract means, which was difficult to identify and locate structured information. To solve such problems, this paper proposed a kind of Token-based structured code matching homology detection technology, which integrated LCS algorithm with hopping information retention, structural border demarcation, windows search, and effective sequence definition for Token abstract structured information matching. Experiment with different project code samples as test scenarios shows that the method is effective for structured information homology detection.

**Key words**: homology detection; common subsequence; structured information; code variants

## 0 引言

软件同源性检测通过分析软件的源代码或软件二进制程序,度量多款软件之间的相似性,从而判断软件之间是否存在互为变体的可能。代码变体主要表现为对代码的完全拷贝、函数名和变量名替换、语句顺序打乱、类型重定义等。软件同源性检测技术在软件重构性检测[1]、软件知识产权保护、计算机犯罪司法取证、研发者合法权益维护等方面发挥着积极的作用,已经成为研究热点。当前,软件同源性检测技术主要分为二进制程序和源代码。计算机语言成分少,针对每门具体的语言都只有有限多个关键字,结构简单,因此基于代码层次的软件同源性检测技术具有实现简单、实行效率高等优势。源代码同源性检测主要有基于文件属性、字符串、Token、树和语义的方法等。

基于文件属性的方法采用文本信息、文本内容散列处理的方法,通过散列值比较,评估源文件(待检测文件)与目标文件的相似性。虽然检测性能高,但对于字符串替换模式的检测有效性较低[2]。基于字符串的方法将源文件转换为字符串组,通常一个字符串对应于一行源代码,通过比对源代码程序段对应的字符串组,评估源代码间的相似性,但该方法只对于代码的完整拷贝或部分拷贝适用,对应代码拷贝修改的检测不鲁棒[3]。基于树的方法采用语法树或抽象语法树产生源代码描述,文件比对通过语法树或抽象语法树中子树的匹配来评估[4],子树的信息可以进一步生产指纹信息,通过指纹信息的比对,形成可能存在的克隆点[5]。基于语义的方法,通过程序依赖图(program dependence graphs, PDGs)中同构子图的匹配识别不同源代码中代码的克隆关系[6],该方法也被用于软件重构中的程序提取[7]。基于语法和语义在检测能力方面具有较好的表现,但其性能通常无法满足实际要求。

基于 Token 的方法通过源代码的词法化产生 Token(标号)序列,通过 Token 的比对实现源代码相似性度量。该方法相对于基于文件属性的方法和基于字符串的方法具有较好的鲁棒性,并且相对于基于树的方法与基于语义的方法具有较好的性能[8]。然而,对于代码块结构中内容的修改,该检测方法还缺乏相应的手段,如在代码块结构中插入新语句、在代码块结构中删除语句等。本文提出了一种基于 Token 的结构化匹配同源性代码检测算法,将跳变信息保留、结构边界划分、窗口搜索、计数重置、有效序列界定等逻辑融入到 LCS 算法逻辑中,并应用于 Token 摘要的比对,创新了 LCS 算法矩阵的匹配模式,解决代码块结构中内容增加、删节、修改的匹配问题。

## 1 基于 Token 的软件同源性检测原理与技术

基于 Token 比对的软件同源性检测通过将源代码中的标志符、关键字、运算符、分隔符、常量、注释符等进行预处理,并进行词法分析将源代码转换成 Token 序列,以多种规则进行源程序文件代码与目标程序文件代码比对,以实现代码同源性检

**收稿日期**: 2013-07-22; **修回日期**: 2013-09-10 **基金项目**: 电子发展基金资助项目

**作者简介**:刘云龙(1976-),男,辽宁大连人,高级工程师,博士,主要研究方向为云计算安全、信息系统设计、数字信号处理(welldome@163.com).

测的综合量化评价。该方法可以有效地鉴别程序代码替换问题,从而提高源代码同源性检测的准确率。总体上基于 Token 的软件同源性检测原理与技术可以归结为以下几个方面[1,2,9,10]:

a)预处理。这部分处理不影响程序的功能部分,对于注释、头文件导入语句、宏定义等进行处理。对于不影响语义的字符,预处理过程中设置为空,如宏定义、注释、TAB、回车、空格等。对于类型重定义情况,将源代码中重定义类型还原为原始类型,以降低同源性检测的误检率。

b)词法分析。进行源代码的词法分析。通过读入源代码信息,依据程序语言词法规则实现每行代码的参数(Token)化表示,例如返回与输入标志符、关键字等相匹配的符号序列(Token serial)。此项工作可以通过词法分析器 Lex(lexical compiler)完成。

c)建立代码行数据结构。对于参数化的源代码文件,以行为单位建立源文件和目标文件的行描述数据结构,记录行信息,并依此形成行链表。

d)计算行摘要值。行数据结构中要记录行的摘要值。实现代码行摘要值的高性能计算及摘要值的唯一性是实现高效源代码比对的关键,行摘要计算将不考虑空格和 TAB,通常通过 Token 自定义值与 Token 位置信息形成摘要值。

e)代码块比对。通过源文件和目标文件的相同行链表,比较源文件和目标文件的链表中的信息摘要值,可以依次遍历多个源文件和目标文件,实现文件的多对一对比,比对结果存入行链表的相关位置,通常采用 LCS 方法定位相应代码中的同源性模块。

f)相似度计算。根据行链表摘要信息进行代码比对评估,计算相似模块总计行数,以相似的行数与总行数的比值计算相似度。

基于 Token 的软件同源性检测的处理流程可以通过图 1 来描述。基于 Token 的软件同源性检测技术中的核心部分是基于行对比的相似性检测,以行为单位进行 Token 序列的比对,对于完全相同的代码块及行顺序打乱的代码块的拷贝都有较好的检测效果。其中,最长公共子序列(LCS)算法通常用于代码块相似性检测中。

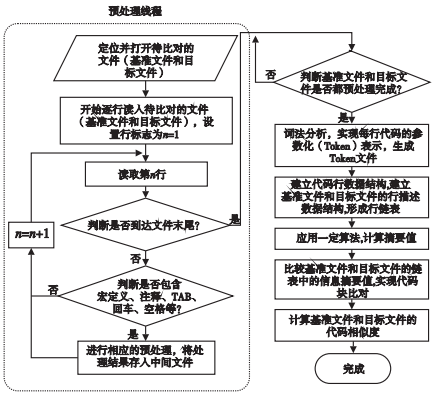



图1 基于Token的软件同源性检测的处理流程

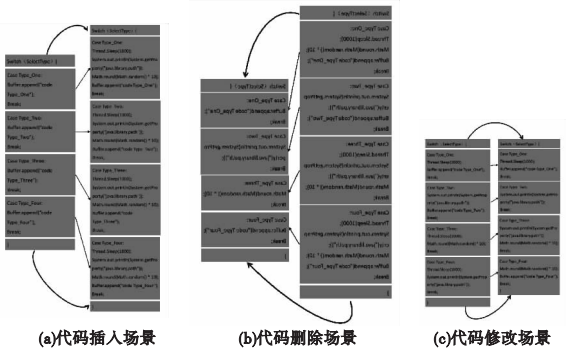
## 2 基于 Token 同源性代码检测问题分析

通常情况下,对于被克隆代码块本身应具备一定的结构化模式信息,这种信息通常便是用来克隆的主要代码框架,即代码中具有可利用的克隆结构。可以以被克隆文件源代码块为范本,在此基础上通过插入相应的定制性代码、删除无关代码及修改主题代码实施代码的克隆行为,以实现代码重用的目的,从而实现代码复用的定制性,完成代码级功能及非功能属性的重用。

对于这种窜改行为,传统的基于 Token 的源代码同源性检测技术很难模式化地定位和发现克隆代码块,相应检测结果的精确度及召回率都不同程度地有所损失。为说明问题,本文构建了几种简单场景,用以说明基于结构化信息利用的代码克隆现象,旨在说明本文算法实施的主要动机。图 2 所示为基于 switch 语句模板结构进行源代码克隆修改的行为。图 2(a)为在原有代码结构的基础上,根据逻辑需求新增执行代码的场景,该场景主要是利用源代码的结构化信息增加定制性信息,丰富应用的定制性功能或提升非功能属性。图 2(b)为在原有代码结构的基础上,根据逻辑需求删减执行代码的场景,该场景主要是利用已有的代码结构化信息,通过裁剪相应的代码内容保留可应用语句部分,达到预期的功能及非功能属性,从而实现结构化信息及部分代码内容的重用。图 2(c)为在原有代码结构的基础上,根据逻辑需求或业务需要替换执行代码的场景,该场景主要利用已有代码的结构化信息,替换或更改相应的代码内容,实现代码逻辑的功能及非功能属性的替代,从而有效地利用代码框架信息实现定制性业务或逻辑。

针对这些场景的设置,利用传统的源代码级同源性检测技术将通过基于行的检测方法进行定位,查找相似的克隆行,检测评估代码的克隆行为。特别地,对于基于 Token 的源代码检测方法,通过源代码的 Token 处理,逐行实现代码行的抽象,通过一定的相似性度量手段,量化比对代码的相似性。

传统的基于 Token 检测方法,没有考虑代码结构的整体特性,忽略了代码的结构化信息,难以定位和识别结构化对等代码块的克隆。因此,基于该类方法将难以扫描、定位和识别出代码内部的相似模式结构,过多地关注于基于连续行的比对结果,缺乏代码结构同等类别的模式识别能力,导致相似代码块结构的漏检,不利于高抽象级别代码相似性比对,从而影响整体代码相似性度量的精度及物理意义。图 2 用例子场景说明了几种典型的克隆行为,具有一定的代表性。该场景作为本文算法实施的应用场景,在代码克隆行为中普遍存在,同时,也是普遍被忽略的上层克隆行为。业界对于源代码级同源检测技术,缺乏对于基于结构化信息的代码利用的判断,因此,在相应的检测领域,对应的技术手段也相对缺乏或不完善。



(a)代码插入场景 (b)代码删除场景 (c)代码修改场景

图2 基于模板结构的源代码克隆

## 3 基于 Token 的结构化匹配同源性代码检测技术

### 3.1 传统基于 LCS 算法同源性检测技术分析

传统基于 Token 的软件同源性检测技术,以行为单位,以 Token 生成摘要值,进行摘要值序列的比对实现代码的相似性的度量。该级别的比对没有考虑代码上层结构信息,代码比对抽象层次还不足以实现智能化模式分析。因此,如果将代码抽

象级别提高,可以兼顾代码细节与结构化信息,从而实现更高级别的代码同源性检测。本文将在代码行摘要化的基础上进行同源性检测研究,通过代码结构化信息评估,运用结构化匹配算法增加代码结构信息智能模式识别功能,提升代码相似性度量和自动化评估水平。从代码内容、结构化模式信息方面检测代码样本的同源性,实现基于 Token 的高级别同源性检测,从而提升代码同源性检测的合理化水平与精度,将代码同源性检测技术推向一个新的级别与高度。

一般地,代码同源性匹配过程使用一定的相似性度量方法。LCS 是实现代码行相似性匹配的常用技术手段。其通过一个矩阵来记录两个序列中所有位置的元素的匹配情况[11],矩阵的行与列分别表示不同的字符串,如果位置字符匹配,则将对应的矩阵位置元素标记为符合一定规则的数字,然后找出符合算法约定的标记位置及匹配长度,确定两个序列中最长的公共子序列,从而判定匹配字符串的位置及长度。关于 LCS 算法,文献[12~14]给出了很多改进措施,但大多是算法时空复杂度方面的改进,相对于算法逻辑的改进较少。对于代码结构化信息的比对,简单基于 LCS 算法的比对已经难以评估和提取代码的结构化相似性。本文应用场景涉及到多个程序样本的公共子序列比较,并非最长公共子串,并且要基于同源性比对场景定义规则改进 LCS 算法逻辑。

### 3.2　典型 LCS 算法

字符串 $v$ 的子序列可以定义为:源于 $v$ 的有序的字符子序列,该序列在 $v$ 中不一定连续。例如,如果 $v$ = ATTGCTA,那么 AGCA 和 ATTA 为 $v$ 的子序列,而 TGTT 和 TCG 不是 $v$ 的有效子序列。最长公共子串问题可以定义为:在多个字符串中,找出字符外形的顺序一致的最长公共字符子串序列[8],即满足上述条件的公共子序列。通常通过以下方式定义 $v=A_0A_1\cdots A_{m-1}$ 和 $w=B_0B_1\cdots B_{n-1}$ 的公共子序列:

在 $v$ 中子序列的位置满足 $0\leqslant i_1<i_2<\cdots<i_k\leqslant m-1$。

在 $w$ 中子序列的位置满足 $0\leqslant j_1<j_2<\cdots<j_k\leqslant n-1$。

$v$ 和 $w$ 中相应的符号在以上相应的位置一致 $v_{i_t}=w_{i_t}$,$1\leqslant t\leqslant k$。

虽然在 $w$ 和 $v$ 中可能存在许多公共子串,但必定存在公共子串长度长于其他公共子串的情况。该子序列为 $w$ 和 $v$ 的公共子序列。对于求解字符串 $A_0A_1\cdots A_{m-1}$ 和 $B_0B_1\cdots B_{n-1}$ 最长公共子串问题,最著名的 LCS 实现之一由 Wahner 等人[15]研究开发,运行时间复杂度为 $O(mn)$,空间复杂度为 $O(\max(m,n))$。LCS 是实现序列相似性分析的最有效手段之一[16]。其中求解的字符串子序列是源于带比较字符串中的公共最长有序子序列,但不一定是连续的。通常使用式(1)描述 LCS 基本算法。

$$L(i,j)=\begin{cases}L(i+1,j+1)+1 & \text{if}\quad A_i=B_j\\ \max\begin{Bmatrix}L(i,j+1)\\L(i+1,j)\end{Bmatrix} & \text{if}\quad A_i\neq B_j\\ 0 & \text{if}\quad i=0\vee j=0\end{cases}\tag{1}$$

令 $v$ = ATCTGAT,$w$ = TGCATA,则 TCTA 是 ATCTGAT 和 TGCATA 的公共子序列。图 3 描述了计算 $v$ 和 $w$ 相似性评分的动态程序表。

### 3.3　基于结构化匹配的同源性检测中 LCS 算法的改进

如果要完成本文同源性检测场景,传统 LCS 算法在业务逻辑上需要修改。LCS 运用于源代码级同源性检测存在以下几个问题。首先,代码级同源性检测是要检测出所有满足一定约束条件的 Token 摘要值公共子串,并非一定是最长子串。同时,根据传统 LCS 方法,在匹配矩阵中每行字符的匹配,产生跳变的元素在与首个匹配位置,在之后的匹配处不产生跳变信息,因此很难提取除最长公共子序列之外的其他子序列,如图 4 所示,$w$ = ATCTGAATCTGA,$v$ = ATCTGA,$w$ 包含两个 $v$,而 $v$ 在 $w$ 的后半段没有跳变信息用于匹配。此外,由于 LCS 中公共子串位置允许存在一定的不连续性,并且没有限制不连续字符的尺度,因此,不能清晰度量代码逻辑结构内部的相关性边界,从而不能完全适应代码级同源性检测的业务场景。最后,LCS 缺乏匹配的停止条件,对于公共子序列的不连续性没有作规定,而对于代码同源性匹配,要求在一定距离不匹配,需要开启新的匹配子序列。

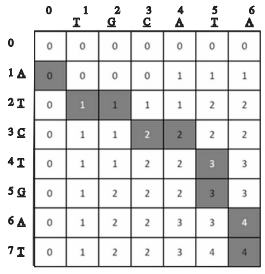


图3 计算$w$与$v$的相似性

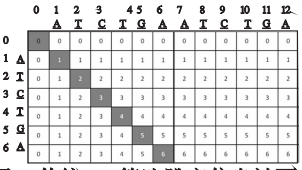


图4 传统LCS算法跳变信息被覆盖

为解决传统源代码级同源性检测及传统 LCS 算法在代码同源性检测应用中存在的问题,本文在传统 LCS 的基础上实现了改进。具体实现如下:

按照式(2)生成相似性矩阵。其中,$L(i,j)$ 为相似性矩阵中的元素,其根据摘要值匹配情况生成。矩阵中行方向为源文件摘要值方向,列方向为目标文件摘要值方向。首先初始化矩阵的第一行和第一列元素为 0。采用行优先的方式,逐行扫描列摘要和行摘要的匹配情况。为产生跳变信息,在摘要值不匹配的位置填入 0,在摘要值匹配的位置,以该位置为右下角,在向左上方向生长的正方形窗口中搜索最大的非零元素,并将其累计加 1,填入匹配成功的位置。这里需要说明的是,对于结构相似性的评估,不允许无限制地对于结构内部代码进行增、删、改,即不能将结构内部尺度无限制地增加或减少。因此,需要设定一定的阈值,规范代码结构匹配范围。如果结构内部代码的改动超出了一定范围,则需要重新计数匹配情况,将对应匹配位置元素重新置 1,重新开始计数匹配情况,这样可以将摘要匹配序列,依据匹配情况,清晰划分结构化边界。为实现规定代码尺度范围内的结构及代码匹配,矩阵中匹配成功摘要值的元素计算,是通过可扩展正方形窗口实现有效范围内摘要值最大匹配累计计数的搜索,正方形窗口具有扩展性,依据搜索结果判断是否增加窗口尺寸或尺寸重置,如式(3)所示。为避免累计匹配计数的重复使用,利用未标记集合来剔除已匹配摘要值的累计计数,减少重复匹配情况,降低虚警率,如式(4)所示。其次,为记录待比较摘要值的同源性情况,需记录每一个摘要值的匹配情况。这里,需要计数摘要值的匹配频度和定位匹配位置。此外,由于 LCS 缺乏匹配的停止条件,这里使用变量 $th_{\text{win}}$ 标注不匹配距离的下限。最后,为界定有效匹配子序列长度,这里使用变量 $th_{\text{seq}}$ 标注有效匹配子序列最小长度。

$$L(i,j)=\begin{cases}\max\{L(h,k)\mid(h,k)\in w(i,j)\}+1 & \text{if } A_i=B_j\wedge\max\{L(h,k)\}>0\wedge\\ & (h,k)\in\Omega\wedge\text{diagonal}(S_{w(i,j)})\leqslant th_{\text{win}}\\ 0 & \text{if } A_i\neq B_j\\ 0 & \text{if } i=0\vee j=0\\ 1 & \text{if } A_i=B_j\wedge\max\{L(h,k)\}=0\ \wedge\\ & \{L(h,k)\}\notin\Omega\wedge\text{diagonal}(S_{w(i,j)})>th_{\text{win}}\end{cases}\tag{2}$$

$$S_{w(i,j)}=\begin{cases}S_{w(i,j)}+1 & \text{if } \max\{L(h,k)\}=0 \vee (h,k)_{L(h,k)\neq 0}\notin \Omega \vee \\ & \text{diagonal}(S_{w(i,j)})<th_{\text{win}} \\ 1 & \text{else}\end{cases} \tag{3}$$

$$\Omega=\begin{cases}\Omega\backslash(h,k) & \text{if } A_i=B_j \wedge \max\{L(h,k)\}>0 \wedge \\ & (h,k)\in\Omega \\ \{(h,k)\mid 1\leqslant h\leqslant m, 1\leqslant k\leqslant n\} & \text{if } i=0\wedge j=0\end{cases} \tag{4}$$

总体上,算法的实施步骤如下:

a)初始化矩阵的第 1 行与第 1 列为 0。初始化未标记集合 $\Omega$ 为所有矩阵坐标位置,如式(4)所示。当前行列号为:行号赋值为 1,列号赋值为 1。设置匹配计数变量 $N_{\text{match}}=0$。初始化阈值 $th_{\text{win}}$ 值。

b)初始化可扩展正方形窗口尺寸 $S_{w(i,j)}=1$。

c)在当前行列号处,匹配源文件摘要值和目标文件摘要值。

d)如果匹配失败,转 e);如果匹配成功,转 f)。

e)在矩阵相应位置填入 0,转 m)。

f)根据式(3)改变窗口尺寸。

g)判断窗口尺寸是否超出了阈值范围,如果否,转 h);否则转 l)。

h)搜索窗口中最大非零元素。如果存在非零最大元素,转 i);否则转 f)。

i)判断该矩阵位置索引$(h,k)$是否在标记集合 $\Omega$,如果是,转 j);否则转 f)。

j)根据式(2),将非零最大元素加 1,填入矩阵相应位置,记录矩阵的位置到匹配位置集合链表。根据式(4)将矩阵匹配位置索引从 $\Omega$ 中剔除。转 m)。

k)如果不在未标记集合,转 f)。

l)矩阵当前位置元素置为 1,根据式(3)将窗口尺寸置为 1。

m)当前列号加 1。如果超出矩阵列数,将行号加 1,列号赋值为 1。如果超出矩阵行数,转 n);如果未超出矩阵行数,转 b)。

n)统计匹配位置集合链表中长度大于阈值 $th_{\text{seq}}$ 的成功匹配子序列中不同元素的个数,即相似性行个数,记为 $\text{line}_{\text{similar}}$。其中 $th_{\text{seq}}$ 标记有效匹配子序列的最小长度。

o)根据式(5)计算源文件代码与目标文件代码的相似性。

$$\text{similar}=\text{line}_{\text{similar}}/\text{line}_{\text{waitingcodes}} \tag{5}$$

以上基于改进 LCS 算法的结构化信息匹配逻辑,将替换第 1 章中的代码块比对和相似度计算部分,形成本文算法。图 5 为算法实施检测的结果示意图,其中,$th_{\text{win}}=2$,$th_{\text{seq}}=2$。

| | 0 | 1 A | 2 T | 3 T | 4 X | 5 C | 6 J | 7 W | 8 T | 9 A | 10 H | 11 G | 12 K | 13 T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 A | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 2 T | 0 | 0 | 2 | 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 3 C | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 T | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 5 G | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 A | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 |
| 7 T | 0 | 0 | 2 | 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 |

图5 本文算法检测示意图

## 4 实验结果

本文算法应用于电子发展基金“云计算关键支撑软件(平台安全软件)研发与产业化”项目。代码级同源性检测软件开发环境为 Visual Studio 2008,基本业务逻辑以 C 语言实现,并打包为动态链接库以备其他软件环境以组件的方式调用。基本用户图形界面采用 eclipse-jee-helios-SR2-win32 开发环境,使用 Java 语言实现,并通过用户界面接口调用动态链接库. dll 文件实现互操作、可视化同源性检测功能。同源性软/硬件开发及运行环境为 Lenovo T420 笔记本,其中 CPU 为 IntelCore i5-2450 M,主频 2.50 GHz,内存为 4 GB。

实验中,软件参数设置为 $th_{\text{win}}=4$,$th_{\text{seq}}=2$。实验代码为用于生产环境电流监控的真实项目软件源代码样本。同时,对于代码模块进行了内部结构内容的增、删、改操作,产生符合本文检测业务场景的三类代码样本。运用基于本文算法实现的代码级同源性检测软件进行了实验测试,实验结果如图 6 所示。其中,代码插入场景的源文件代码行数为 1147,检测定位具有同源性的代码行数为 143,代码插入场景代码行数为 36,相似性度量结果为 12.471%;代码删除场景的源文件代码行数为 1136,检测定位具有同源性的代码行数为 132,代码删除场景代码行数为 25,相似性度量结果为 11.62%;代码修改场景的源文件代码行数为 1143,检测定位具有同源性的代码行数为 139,代码插入场景代码行数为 32,相似性度量结果为 12.16%。经过多次实验,本文算法都给出了较为鲁棒的实验检测结果,验证了算法的有效性。

a.代码插入场景

b.代码删除场景

c.代码修改场景

图6 本文算法检测结果

## 5 结束语

变体代码是软件同源性检测实施的主要对象,同源性检测在软件重构性检测、软件知识产权保护、计算机犯罪司法取证、维护研发者合法权益等方面发挥着积极的作用。相对于其他同源性检测机理,基于 Token 的源代码级检测方法具有较好的鲁棒性,并且相对于基于树的方法与基于语义的方法具有较好的性能。本文针对传统基于 Token 的代码级同源性检测技术中存在的结构化内容检测乏力的问题,给出了一种有效的解决方案,提出了一种基于 Token 的结构化匹配同源性代码检测算法,通过创新 LCS 算法矩阵的匹配模式,解决了代码内容窜改的场景下,结构化信息的匹配和识别问题。实验证明该方法针对问题代码能够有效、鲁棒地进行检测、定位。本文工作内容被成功地运用于电子发展基金“云计算关键支撑软件(平台安

全软件)研发与产业化"项目,并期待具有一定的产业化价值。

**参考文献:**


[1] JIANG Ling-xiao, MISHERGHI G, SU Zhen-dong, *et al*. DECKARD: scalable and accurate tree-based detection of code clones[C]//Proc of the 29th International Conference on Software Engineering. 2007: 96-105.

[2] CUI Bao-jiang, GUAN Jun, GUO Tao, *et al*. Code syntax-comparison algorithm based on type-redefinition-preprocessing and rehash classification[J]. Journal of Multimedia, 2011, 6(4): 320-328.

[3] BAKER B S. On finding duplication and near-duplication in large software systems[C]//Proc of the 2nd Working Conference on Reverse Engineering. 1995: 86-95.

[4] BAXTER I D, PIDGEON C, MEHLICH M. DMS: program transformations for practical scalable software evolution[C]//Proc of the 26th International Conference on Software Engineering. 2004: 625-634.

[5] KONTOGIANNIS K, MORI D R, MERLO E, *et al*. Pattern matching for clone and concept detection[J]. Automated Software Engineering, 1996, 3(1/2): 77-108.

[6] WEISER M. Program slicing[J]. IEEE Trans on Software Engineering, 1984, 10(4): 352-357.

[7] KOMONDOOR R, HORWITZ S. Semantics-preserving procedure extraction[C]//Proc of POPL. 2000: 155-169.

[8] CAMPOS R A C, MARTINEZ F J Z. Batch source-code plagiarism detection using an algorithm for the bounded longest common subsequence problem[C]//Proc of the 9th International Conference on Electrical Engineering, Computing Science and Automatic Control. 2012: 1-4.

[9] KAMIYA T, KUSUMOTO S, INOUE K. CCFinder: a multilinguistic token-based code clone detection system for large scale source code[J]. IEEE Trans on Software Engineering, 2002, 28(7): 645-670.

[10] TOOMEY W. Ctcompare: code clone detection using hashed token sequences[C]//Proc of the 6th International Workshop on Software Clones. 2012: 92-93.

[11] RUBI R D, AROCKI L. Positional_LCS: a position based algorithm to find longest common subsequence (LCS) in sequence database (SDB)[C]//Proc of IEEE International Conference on Computational Intelligence & Computing Research. 2012: 1-4.

[12] PARVINNIA E, TAHERI M, ZIARATI K. An improved longest common subsequence algorithm for reducing memory complexity in global alignment of DNA sequences[C]//Proc of International Conference on BioMedical Engineering and Informatics. 2008: 57-61.

[13] LIN Jin-xian, LV Yi-qing. Computing all longest common subsequences on MPI cluster[C]//Proc of the 2nd International Conference on Computational Intelligence and Natural Computing Proceedings. 2010: 386-389.

[14] YANG Jiao-yun, XU Yun, SUN Guang-zhong, *et al*. A new progressive algorithm for a multiple longest common subsequences problem and its efficient parallelization[J]. IEEE Trans on Parallel and Distributed Systems, 2013, 24(5): 862-870.

[15] WAHNER R A, FISCHER M J. The string-to-string correction problem[J]. Journal of ACM, 1974, 21(1): 168-173.

[16] JONES N C, PEVZNER P A. An introduction to bioinformatics algorithms[M]. Cambridge: MIT Press, 2004.


(上接第 1835 页)

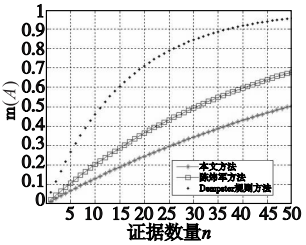



**图1　假设{A}的mass分布变化曲线**

## 5　结束语

由于人为或自然环境等因素,信息融合系统的证据常常有较大的冲突,这时传统的 Dempster 组合规则无法有效地处理这些冲突证据。本文针对证据合成中的悖论消除问题,在分析现有证据理论改进方法的基础上,提出了一种综合考虑合成规则和数据模型的联合改进方法。实验结果表明,该方法可以有效地处理冲突证据,合成结果与人们日常信息融合结果相一致,能够较好地解决证据融合中的合成悖论问题。

**参考文献:**


[1] DEMPSTER A P. Upper and lower probabilities induced by a multivalued mapping[J]. Annals Mathematical Statistics, 1967, 38(2): 325-339.

[2] SHAFER G. A mathematical theory of evidence[M]. Princeton: Princeton University Press, 1976: 10-40.

[3] SMETS P. The combination of evidence in the transferable belief model[J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 1990, 12(5): 447-458.

[4] SMARANDACHE F, DEZERT J. Advances and application of DSmT for information fusion[M]. [S. l.]: American Research Press, 2004.

[5] YAGER R. On the Dempster Shafer framework and new combination rules[J]. Information Sciences, 1987, 41(2): 93-137.

[6] 孙全,叶秀清,顾伟康. 一种新的基于证据理论的合成公式[J]. 电子学报,2000,28(8):117-119.

[7] 李弼程,王波,魏俊,等. 一种有效的证据理论合成公式[J]. 数据采集与处理,2002,17(1):33-36.

[8] 张山鹰,潘泉,张洪才. 一种新的证据推理组合规则[J]. 控制与决策,2000,15(5):540-545.

[9] 向阳,史习智. 证据理论合成规则的一点修正[J]. 上海交通大学学报,1999,33(3):357-360.

[10] LEFEVRE E, COLOT O, VANNOORENBERGHE P. Belief function combination and conflict management[J]. Information Fusion, 2002, 3(3): 149-162.

[11] 邢清华,雷英杰,刘付显. 一种按比例分配冲突度的证据推理组合规则[J]. 控制与决策,2004,19(12):1387-1390.

[12] MURPHY C K. Combining belief functions when evidence conflicts[J]. Decision Support Systems, 2000, 29(1): 1-9.

[13] 邓勇,施文康,朱振福. 一种有效处理冲突证据的组合方法[J]. 红外与毫米波学报,2004,23(1):27-32.

[14] 梁昌勇,陈增明,黄永青,等. Dempster-Shafer 合成法则悖论的一种消除方法[J]. 系统工程理论与实践,2005,25(3):7-12.

[15] 陈炜军,景占荣,袁芳菲,等. D-S 证据理论的不足及其数学修正[J]. 中北大学学报:自然科学版,2010,31(2):161-168.

[16] 蒋雯,张安,邓勇. 基于新的证据冲突表示的信息融合方法研究[J]. 西北工业大学学报,2010,28(1):27-32.

[17] 韩德强,韩崇昭,邓勇,等. 基于证据方差的加权证据组合[J]. 电子学报,2011,39(3A):153-157.

[18] 周哲,徐晓滨,文成林,等. 冲突证据融合的优化方法[J]. 自动化学报,2012,38(6):976-985.

[19] 杨风暴, 王肖霞. D-S 证据理论的冲突证据合成方法[M]. 北京: 国防工业出版社, 2010.