DOI:10.3969/j.issn.1673-4785.201209054
网络出版地址:http://www.cnki.net/kcms/detail/23.1538.TP.20130125.1445.005.html

基于序列聚类的相似代码检测算法

于世英[1,2],袁雪梅[1],卢海涛[1],任家东[1],李硕[1]
(1.燕山大学 信息科学与工程学院,河北 秦皇岛 066004; 2.河北省科技管理信息中心,河北 石家庄 050021)

摘 要:为了提高源程序代码之间相似性的检测效率,提出一种基于序列聚类的相似代码检测算法.算法首先把源代码按照其自身的结构进行分段提取,然后对各个分段进行部分代码变换,再以带权重的编辑距离为相似度量标准对这些符号进行序列聚类,得到相似的程序代码片段,以达到对源程序进行相似功能检测的目的.使用多个真实和仿真程序对上述算法进行了实验,实验结果验证了算法的有效性和可伸缩性.

关键词:序列聚类;权重编辑距离;相似代码检测
中图分类号:TP311.131 **文献标志码**:A **文章编号**:1673-4785(2013)01-0052-06

Similar code detection algorithm based on sequence clustering

YU Shiying[1,2], YUAN Xuemei[1], LU Haitao[1], REN Jiadong[1], LI Shuo[1]
(1. College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China; 2. Hebei Provincial S&T Management Information Center, Shijiazhuang 050021, China)

Abstract: In order to improve efficiency of similar detecting between the codes of source programs, similar code detection algorithm based on sequence clustering was proposed in this paper. First, the algorithm extracts the source code by partitioning it into multi-segments according to its structure. Secondly, parts of the codes in each segment were transformed and the sequences were then clustered taking the weighted edit distance as the similar measure standard. Next, similar code fragments were obtained, achieving the objective of detecting similar functions among multi-codes of the source program. The experimental results based on a series of real and synthetic programs reveal the validity and scalability of the algorithm.

Keywords: sequence clustering; weighted edit distance; similar code detection

随着计算机软件的发展,程序代码在日常生活中越来越常见,代码能够实现各种各样的计算、匹配和查询.正是由于软件行业的快速发展,以编程为职业的工作人员层出不穷,因而代码的质量也因人而异.相似代码的检测应运而生,能在许多应用中发挥作用,比如可以通过检测源程序中的相似代码对源程序进行简化,也可以查找出多个程序之间的相似功能,还能用于抄袭检测.

国内外许多学者对检测相似代码进行了大量研究,K. Kontongiannis 等提出一种使用模式检测代码相似性的方法[1],模式被指定为概念语言中的源程序或者摘要描述的序列.该方法采用一系列代码对代码和摘要描述对代码的匹配技术,前者使用动态编程技术来局部化相似代码片段,能处理大型软件系统.后者使用马尔可夫模型,根据一个给定摘要描述生成一个给定代码段的概率,来计算一个摘要描述与代码段之间的不同距离. A. Ohno 提出一种使用相关向量来度量源代码相似性的算法[2].该算法直接使用相关向量而不是使用源程序代码,相关向量中的元素是从源代码中产生的 token-cooccurrence 矩阵的结构特征值,由于相似性仅仅定义为2个相关向量之间的距离,因此该方法节省了计算时间,同时也不用为存储源程序开辟大的存储空间. T.

收稿日期:2012-09-25. 网络出版日期:2013-01-25.
基金项目:国家自然科学基金资助项目(61170190).
通信作者:于世英. E-mail: wangqianysu@163.com.

Yamamoto 等提出一种基于源代码通信的大型软件系统相似性度量算法[3],根据这个相似度量开发出了一个软件相似性度量工具 SMAT,并且应用到了多个版本的操作系统中. J. H. Ji 等提出一种自适应的局部队列,把关键字的频率映射到相似度矩阵中,使用这个局部队列来自动检测程序中的相似代码片段[4]. 文献[5]提出了一种基于编译优化和反汇编的程序相似性检测方法. 该方法采用编译优化和反汇编技术将源程序代码转化为汇编指令集合,再删除和替换汇编指令中对程序本质影响不大的易变因素,然后使用一个与指令序列无关的决策函数计算程序相似度,最后给出一个简单有效的聚类算法,从程序集合中聚类出相似的程序子集. 文献[6]针对源程序代码相似度的度量问题,提出一种基于属性计数和结构度量的方法. 通过统计源程序代码的操作符和操作数个数,得到 Halstead 长度、Halstead 词汇和 Halstead 容量这 3 个源程序的特征向量,使用向量夹角的余弦公式计算属性相似度,利用最长公共子序列算法得到结构相似度,从而权衡程序之间的相似程度. 文献[7-9]中也分别提出了不同的相似代码检测算法,它们基于不同的结构,从不同的角度来对相似代码进行检测.

大部分判断代码相似的文献是通过不同的相似度度量来判断 2 个或多个源代码整体是否相似,这样增加了一些不必要的计算. 这是因为函数功能段在程序代码中占有主体地位,如果 2 个源程序中的功能段相似,那么就能确定这 2 个源程序基本相似. 因此,本文首先提取出源代码中的功能段,再以带权重的编辑距离为相似度量标准,通过聚类源程序代码序列的方法,查找出源程序中相似的代码功能段,以达到检测相似功能程序的目的.

1　问题定义

假设 $C_S=\{a_1,a_2,\cdots\}$ 作为字符集,大小为 $N=|C_S|$. $S\in C_S^*$ 表示一个字符序列,S 由 C_S 中的字符组成,其中的字符数为 $|S|$. 2 条序列 S_1 和 S_2 之间的编辑距离 $D_E(S_1,S_2)$ 是将 S_1 经过插入、删除、替代等操作变换成 S_2 所需要的最少操作次数.

如果有序列 $S=s_1\,s_2\cdots\,s_n$,n_i 是字母表中字母 a_i 在序列 S 中出现的次数,则向量 $\boldsymbol{S}_N(S)=(n_1,n_2,\cdots,n_N)$ 为序列 S 的签名. 若序列 S 和 S' 的签名分别为 $\boldsymbol{S}_N(S)=(n_1,n_2,\cdots,n_N)$ 和 $\boldsymbol{S}_N(S')=(n_1',n_2',\cdots,n_N')$,那么 $D_S(S,S')=\max\left[\sum_{i=1}^{N}I_i^P(n_i-n_i'),\sum_{j=1}^{N}I_j^Q(n_j'-n_j)\right]$[10] 为序列 S 和 S' 之间的签名距离,当 $n_i>n_i'$,$I_i^P=1$,否则 $I_i^P=0$;当 $n_j'>n_j$ 时,$I_j^Q=1$,否则 $I_j^Q=0$.

1.1　带权重的编辑距离

本文对序列中各种不同类型的字符赋予不同的权重,根据字符的可替代性大小规定各个字符的权重. 如果是代表关键字的字符,所得到的权重就大一些,代表变量的字符得到的权重就比较小,因为相比之下,变量的可替代性要比关键字的可替代性大. 本文中,把代表关键字的字符权重设为 2,代表函数或者变量的符号权重设为 1,代表操作符的符号权重设为 1,如表 1 所示.

表 1　不同符号的权重

Table 1　The weight of different symbol

符号类型	权　重
关键字	2
函数或变量	1
操作符	1

要对程序代码段进行聚类分析,首先要解决的问题就是怎样定义程序代码段之间的距离度量,以及怎样确定这 2 段代码是相似的. 本文定义一种带权重的编辑距离(weight edit distance, WED)来衡量 2 个序列之间的距离,度量它们的相似性.

定义 1　带权重的编辑距离(WED)　编辑距离是指将一个符号序列经插入、删除、替换等编辑操作变为另一个序列所需的操作次数,根据序列中各种不同类型的符号的不同权重,把进行编辑操作乘以相应符号操作的权重,就得到一个符号序列到另外一个符号序列的带权重的编辑距离.

1)插入、删除操作. 对于插入或删除操作,由于只涉及一个符号,所以其权重编辑距离设置为该符号规定的权重,如表 2.

表 2　符号插入、删除的权重编辑距离

Table 2　Weight edit distance of insert and delete symbol

符号类型	权重编辑距离(WED)
关键字	2
函数或变量	1
操作符	1

2)转换操作. 同一类型符号内部转换的权重在表3中给出了,不同类型符号之间的转换权重编辑距离则规定为这2种符号类型的权重之和. 表3列举出了本文使用到的符号转换时的不同权重编辑距离.

表3 符号转换的权重编辑距离

Table 3 The weight edit distance of transform symbol

符号类型	权重编辑距离(WED)		
	关键字	函数或变量	操作符
关键字	2	3	3
函数或变量	3	1	2
操作符	3	2	1

例1 假如一个符号序列 S_1 = asdfght,另一个符号序列 S_2 = abcfght. 可以看出这2个序列中只有2个符号不同,设定其中的b、c、d都为关键字类型的符号,而s为操作符类型的权重,根据上面定义的转换权重编辑距离可知,序列 S_1 和 S_2 之间的权重编辑距离是3+2=5.

1.2 序列间的相似度

定义2 序列间的相似度 2个序列 S_1 和 S_2 间的权重编辑距离为 $D_W(S_1,S_2)$,则序列 S_1 和 S_2 之间的相似度 $S_m(S_1,S_2)$ 表示为

$$S_m(S_1,S_2) = 1 - \frac{D_W(S_1,S_2)}{|S_1|+|S_2|}. \quad (1)$$

若有2个序列 S_1 和 S_2,它们之间的相似度为 $S_m(S_1,S_2)$,如果 $S_m(S_1,S_2) \geqslant S_{\min}$ 成立,就说这2个序列 S_1 和 S_2 相似,说明在聚类时这2个序列可以被归为同一个簇,$S_{\min}$ 被称为序列间的最小相似度阈值.

为了方便计算,在进行聚类时,可以直接以权重编辑距离为衡量相似度的标准. 根据相似度的定义式(1),可以得到满足 $S_m(S_1,S_2) \geqslant S_{\min}$ 的最大权重编辑距离边界值 $D_{\max}(S_1,S_2)$ 为

$$D_{\max}(S_1,S_2) = (|S_1|+|S_2|)(1-S_{\min}). \quad (2)$$

即如果满足 $D_W(S_1,S_2) \leqslant D_{\max}(S_1,S_2)$ 时,则这2个序列相似.

性质1 若有2个序列 S_1 和 S_2,其签名距离为 $D_S(S_1,S_2)$,如果满足 $D_S(S_1,S_2) \geqslant D_{\max}(S_1,S_2)$ 时,那么这2个序列不相似.

证明 序列 S_1 和 S_2 之间的签名距离 $D_S(S_1,S_2)$ 反映了序列字母组成上的差异,而权重编辑距离 $D_W(S_1,S_2)$ 反应了 S_1 和 S_2 之间插入、删除、替换操作不同权重的差异,由于给序列中不同类型符号的插入、删除、替换操作赋予了不同的权重(表1~3),那么 $D_W(S_1,S_2) \geqslant D_E(S_1,S_2)$ 成立,而签名距离是编辑距离的下界[10],即 $D_E(S_1,S_2) \geqslant D_S(S_1,S_2)$,因此可得到 $D_W(S_1,S_2) \geqslant D_S(S_1,S_2)$ 成立. 因此,$S_m(S_1,S_2) = 1 - \frac{D_W(S_1,S_2)}{|S_1|+|S_2|} \leqslant 1 - \frac{D_S(S_1,S_2)}{|S_1|+|S_2|}$,如果 $D_S(S_1,S_2) \geqslant D_{\max}(S_1,S_2)$,那么根据式(2),可以得到 $S_m(S_1,S_2) \geqslant S_{\min}$,说明这2个序列不相似.

由于计算签名距离的时间复杂度 $O(m+n)$ 远小于计算权重编辑距离的时间复杂度 $O(mn)$. 因此,根据性质1,在判断序列是否相似时,可以首先通过计算序列间的签名距离来进行初始过滤,再通过计算权重编辑距离来进行最后的判断.

2 基于带权重编辑距离的相似序列聚类算法

本节将详细介绍提出的基于权重编辑距离的相似序列聚类算法SSCW(similar sequence clustering based on weight edit distance).

2.1 源程序代码的分段提取

检测源程序的相似代码,首先需要对源程序代码进行分段,提取出函数功能段. 在对源程序代码进行分段时,采用一种多级分段方法,把源代码分为不同标准下的多种分段,分段的标准有类、函数、语句,然后再对同一个级别下的各个分段代码进行处理. 在查找相似代码时,由于函数功能段是整个源程序代码的主体,因此只需要使用第二级函数分段即可,对函数、语句和类的处理意义不大.

2.2 程序代码的部分转换

由于使用带权重编辑距离作为相似性度量的标准,并且关键字类型、变量类型与操作符类型的权重各不相同,因此需要对它们进行区分. 一般来说,操作符类型的代码能很好地与其他代码区分,但是关键字类型和变量类型的代码在读取和计算距离的时候很难区分. 本文首先把关键字类型的代码转换成一个个数字,再计算权重编辑距离,虽然变量代码里面也不可避免地会出现数字代码,但是这样的源代码很少,与关键字混淆的概率也不大. 这样处理既能与变量代码进行区分,保证可以得到带权重的编辑距离,又能减少计算距离时的操作次数.

2.3 符号序列的聚类

对同一个分段级别内的序列,根据定义1中的权重编辑距离作为相似性度量进行聚类,得到相似的代码段. 下面具体说明聚类过程中用到的几个主要算法.

2.3.1 判断序列与序列是否相似

首先根据式(2)计算出2个序列相似的最大距离阈值 $D_{\max}(S_1,S_2)$(SandSMaxDis);然后根据性质1,先计算2个序列的签名距离 $D_S(S_1,S_2)$(signDisS-

toS)来进行初始过滤,如果它们的签名距离大于最大距离阈值,就直接判定这 2 个序列不相似;如果签名距离小于最大距离阈值,再计算其权重编辑距离 $D_E(S_1,S_2)$(wEditDisStoS)进行进一步判断. 如算法 1 所述.

算法 1　判断序列与序列是否相似算法(isSandSSimilar).

输入:要判断相似性的 2 个序列 S_1 和 S_2.

输出:布尔值(isSimilar).

```
Begin
1) boolean isSimilar;
2) int SandSMaxDis = getMaxDistance(S1, S2);
3) int signDisStoS = calSignDisStoS(S1, S2);
4) If signDisStoS < = SandSMaxDis
5)       int wEditDisStoS = calWEditDisStoS(S1, S2);
6)       If wEditDisStoS < = SandSMaxDis
7)            isSimilar = true;
8)       Else
9)            isSimilar = false;
10)      End if
11) End if
End
```

2.3.2　判断序列与簇是否相似

有了判断序列与序列是否相似的算法作为支撑,判断序列与簇是否相似就很容易了. 只需要把簇中的所有序列依次与该序列进行相似性判断,如果簇中有任意一个序列与该序列相似,就判定为该序列与该簇相似. 如算法 2 所述.

算法 2　判断序列与序列是否相似算法(isSandCSimilar).

输入:要判断相似性的序列 ser 和簇 cluster.

输出:布尔值(isSimilar).

```
Begin
1) boolean isSimilar;
2) For each series s in cluster
3)       isSimilar = isSandCSimilar (ser, s);
4)       If isSimilar
5)            break;
6)       End if
7) End for
End
```

2.3.3　符号序列的聚类

本文采用改进的基于密度的聚类算法对序列聚类,首先创建一个簇列表来存放聚类结果簇,对于存放在 map 中任意一个序列 S_i,判断序列 S_i 与簇列表中的簇的相似关系,直到所有的序列 S_i 都被处理. 此时的相似关系可能为 3 种情况:1)如果簇列表中没有簇与序列 S_i 相似,那么就新建一个簇存放该序列,并且把新创建的簇添加到簇列表中;2)如果簇列表中有一个簇与该序列相似,就把它加入到这个簇中,更新簇的特征值;3)如果簇列表中有多个簇与该序列相似,就把这几个簇合成一个新簇,再把序列 S_i 加入到新簇中,并且在簇列表中移除合并了的那几个簇,添加新创建的簇. 聚类过程如算法 3 所述.

算法 3　聚类算法(clustering).

输入:存放序列的哈希表 seriesMap.

输出:簇列表 clusters.

```
Begin
1) List clusters;
2) For each series s in seriesMap
3)       List simCluNotoSList; // clusters in List clusters which are similar to series s
4)       List removeCluList;
5)       For each cluster c in clusters
6)            If isSandCSimilar(s,c)
7)                 simCluNotoSList.add(c);
8)            End if
9)       End for
10)      If simCluNotoSList.size() = = 0
11)           Create a new cluster c′ to put series s and put c′ into clusters;
12)      Else if simCluNotoSList.size() = = 1
13)           Put s in cluster c and update the feature of c;
14)      Else
15)           Merging clusters in simCluNotoSList into a new cluster c′ and put c′ into clusters;
16)           Remove clusters in simCluNotoSList from clusters;
17)      End if
18) End for
End
```

算法 3 给出了改进的基于密度的序列聚类过程. 其中,步骤 5) ~9)是判断簇列表中有几个簇与序列是相似的,可能出现上面给出的 3 种情况. 步骤 10) ~11)对应情况 1),即簇列表中没有簇与序列相似;步骤 12) ~13)对应情况 2),簇列表中有一个簇与该序列相似;步骤 14) ~16)对应情况 3),簇列表中有多个簇与该序列相似.

3 实验与分析

通过对4个文件对的相似性检测,来分析SSCW算法结果的正确性以及运行时间.测试中使用的代码采用人工合成的代码序列和从网上下载的真实代码.实验中文件对1(int00和int05)和文件对3(cross-ref00和cross-ref02)是从网站http://www.sei.buaa.edu.cn/buaasim下载的,实现统计整数和交叉引用生成器功能的相似代码.文件int05是在文件int00的基础上改变了函数顺序和修改了部分变量名得到的;文件cross-ref02是在文件cross-ref00的基础上改变了函数内部部分代码顺序,并在定义时添加了无用参数构成.文件对2(cluster00和cluster01)和文件对4(adjustCluster00和adjustCluster01)是把笔者编写的源程序中一些代码段稍作修改得到,文件对2中文件cluster01是在文件cluster00的基础上添加了一个功能函数段构成;文件对4中文件adjustCluster01是在文件adjustCluster00的基础上修改了函数内部部分代码得到的.

根据源文件中功能函数的个数以及聚类结果来评判它们是否相似.最终2个源文件的相似度由参数R_s来决定,其计算方法为

$$R_s = \frac{2N_s}{N}. \tag{3}$$

式中:N_s为聚类结果中包含了来自2个源文件的函数的簇个数,N为这2个源程序中总的功能函数个数.

实验所用的计算机配置如下:CPU为Pentium(R) Dual-Core 2.93GHz,内存为2GB,操作系统为Microsoft Windows XP Professional Edition 2002 Service Pack 3.所有算法用JAVA语言编写实现.

3.1 SSCW算法聚类结果

表4给出了使用提出的SSCW算法对相似代码序列的聚类结果,$R_{s,min}$为序列间的最小相似度阈值,R_s为2个源程序文件根据相似函数得到的相似度.

由于在源程序中功能函数所占的比例很大,而且对于程序的功能取决定性作用,因此如果功能函数相似,那么基本上可以确定其所在的源程序也是相似的.从表4中可以看出,算法能够正确判断代码的相似性.另外,还可以看出序列之间的最小相似度阈值$R_{s,min}$和结果文件相似度R_s的关系.由于判断序列相似是判断文件相似的基础工作,而在判断序列是否相似时,$R_{s,min}$是一个决定性参数,序列之间的最小相似度阈值$R_{s,min}$的确定会影响到结果文件相似度R_s.由表4中可以看出,$R_{s,min}$的增加会导致R_s不变或者减小,这是因为序列最小相似度阈值的增加,会使得满足相似条件的序列减少,从而使得最终代码相似性的减小,如表4中文件对1和文件对4中$R_{s,min}$设为0.9时的情况;如果函数代码序列本身的相似度高于设定的最小序列相似度,那么$R_{s,min}$的设定就不会影响到最终结果.

表4 SSCW聚类结果

Table 4 The clustering results of SSCW

测试代码	$R_{s,min}$	R_s
文件对1	0.7	1.000
	0.8	1.000
	0.9	0.500
文件对2	0.7	0.965
	0.8	0.965
	0.9	0.965
文件对3	0.7	1.000
	0.8	1.000
	0.9	1.000
文件对4	0.7	1.000
	0.8	1.000
	0.9	0.000

3.2 SSCW算法运行时间分析

表5中给出了SSCW算法的运行时间随着文件大小、函数个数以及最小序列相似度阈值$R_{s,min}$不同的变化情况,可以看出SSCW算法的运行时间受这3个因素的共同影响.在SSCW算法中,运行时间主要由计算权重编辑距离和聚类这2个过程的时间消耗组成,权重编辑距离的计算时间与序列的长度有关,因此当一个功能函数序列的长度增大时,算法的运行时间会大幅度增加;而聚类的运行时间与功能函数序列的个数以及其相似性有关,如果函数序列的个数增多,聚类所花费的时间也随着增加.由于在计算权重编辑距离时,通过先计算序列的签名距离来进行初始过滤,因此当最小序列相似度阈值$R_{s,min}$确定的序列距离小于序列之间的签名距离时,就不需要计算序列的权重编辑距离,此时需要的时间消耗较少,如表5中文件对2中$R_{s,min}$被设定为0.9的情况.

表5中的3个因素中,文件大小和函数个数决定了函数序列的长度,它们与最小相似度阈值一起决定了权重编辑距离的运行时间.而函数个数及其相似性决定了聚类过程的运行时间,因此聚类过程的时间消耗就与文件个数和最小相似度阈值有关.在文件大小一定的情况下,功能函数的个数越少,说明函数序列的长度越大,此时序列间权重编辑距离的计算时间消耗就越大;但是,功能函数的个数越少,聚类过程所需要的时间消耗就越小,因此,功能函数的个数对运行时间的影响不是很明确.一般来说,当功能函数个数一定时,文件的规模越大、最小相似性阈值越小,运行时间也越多.

表 5　运行时间随着不同因素改变的变化

Table 5　Running time changes with different factors

测试代码	文件大小/kB	函数个数	$R_{s,\min}$	运行时间/s
文件对 1	0.562	2	0.70	1.625
			0.75	1.515
	0.695	2	0.80	1.547
			0.90	1.516
文件对 2	1.370	14	0.70	6.360
			0.75	6.844
	1.460	15	0.80	6.360
			0.90	2.515
文件对 3	2.630	5	0.70	24.687
			0.75	24.953
	2.640	5	0.80	24.578
			0.90	24.875
文件对 4	15.200	3	0.70	1 581.573
			0.75	1 564.016
	17.000	3	0.80	1 568.953
			0.90	1 521.610

4　结束语

本文提出一种基于序列聚类的相似代码检测算法 SSCW,以得到相似功能的代码段. 该方法采用一种多级分段方法,把源代码分为不同标准下的多种分段,分段的标准有类、函数、语句. 将需要检测的代码段提取出来后,把不好区分权重的关键字代码转换为数字序列,以提出的权重编辑距离为距离度量标准,对同一个等级内的符号序列进行聚类分析,得到相似的代码段. 在实验时,使用了多个数据集对提出的算法进行了验证,实验结果证明了该算法的有效性.

参考文献:

[1] KONTOGIANNIS K, GALLER M, DEMORI R. Detecting code similarity using patterns[C]//Working Notes of Third Workshop on AI and Software Engineering: Breaking the Toy Mold (AISE). [S.l.], 1995: 68-73.

[2] OHNO A. Measure source code similarity using reference vectors[C]//Proceedings of the First International Conference on Innovative Computing, Information and Control. Washington, DC, USA: IEEE Computer Society, 2006, 2: 92-95.

[3] YAMAMOTO T, MATSUSHITA M, KAMIYA T, et al. Measuring similarity of large software systems based on source code correspondence[C]//Proceedings of the 6th International Conference on Product Focused Software Process Improvement. Berlin/Heidelberg: Springer-Verlag, 2005: 530-544.

[4] JI J H, PARK S H, WOO G, et al. Source code similarity detection using adaptive local alignment of keywords[C]//Proceedings of the Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies. Washington, DC, USA: IEEE Computer Society, 2007: 179-180.

[5] 赵长海,晏海华,金茂忠. 基于编译优化和反汇编的程序相似性检测方法[J]. 北京航空航天大学学报, 2008, 34(6): 711-715.
ZHAO Changhai, YAN Haihua, JIN Maozhong. Approach based on compiling optimization and disassembling to detect program similarity[J]. Journal of Beijing University of Aeronautics and Astronautics, 2008, 34(6): 711-715.

[6] 于海英. 程序代码相似度度量的研究与实现[J]. 计算机工程, 2010, 36(4): 45-49.
YU Haiying. Research and implementation of program code similarity measurement[J]. Computer Engineering, 2010, 36(4): 45-49.

[7] JIANG Linxiao. Scalable detection of similar code: techniques and applications[D]. Davis, CA, USA: University of California Davis, 2009: 12-45.

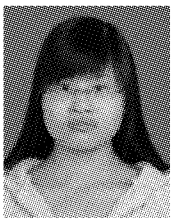
[8] 张丽萍,刘东升,李彦臣,等. 一种基于 AST 的代码抄袭检测方法[J]. 计算机应用研究, 2011, 28(12): 4616-4620.
ZHANG Liping, LIU Dongsheng, LI Yanchen, et al. AST-based code plagiarism detection method[J]. Application Research of Computers, 2011, 28(12): 4616-4620.

[9] 钟美,张丽萍,刘东升. 基于 XML 的 C 代码抄袭检测算法[J]. 计算机工程与应用, 2011, 47(8): 215-218.
ZHONG Mei, ZHANG Liping, LIU Dongsheng. Plagiarism detection algorithm based on XML for C code[J]. Computer Engineering and Applications, 2011, 47(8): 215-218.

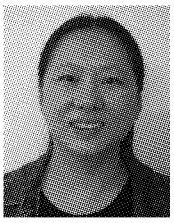
[10] 戴东波,汤春蕾,熊赟. 基于整体和局部相似性的序列聚类算法[J]. 软件学报, 2010, 21(4): 702-717.
DAI Dongbo, TANG Chunlei, XIONG Yun. Sequence clustering algorithms based on global and local similarity[J]. Journal of Software, 2010, 21(4): 702-717.

作者简介:

于世英,1973 年生,女,工程师,主要研究方向为数据挖掘.



袁雪梅,女,1989 年生,硕士研究生,主要研究方向为数据挖掘.



卢海涛,女,1975 年生,讲师,主要研究方向为数据挖掘、虚拟现实.