doi 10.3969/j.issn.1671-1122.2014.01.009

一种改进的基于抽象语法树的软件源代码比对算法

刘楠[1]，韩丽芳[1]，夏坤峰[2]，曲通[2]
（1. 中国电力科学研究院，北京 100192 ；2. 北京邮电大学计算机学院 , 北京 100876）

摘 要 ：在软件同源性检测方法中，基于抽象语法树的比对方法能够有效地检测出基于代码全文拷贝、修改变量名、调整代码顺序等的抄袭手段，被广泛用于抄袭检测工具中。但基于抽象语法树的比对方法对于修改变量类型和添加无意义变量的抄袭手段束手无策。针对这种情况，提出了一种基于抽象语法树的改进思想，该思想通过剪去语法树中影响判断的叶子节点的手段来还原检测原文抄袭，能够达到有效检测修改变量类型和添加无意义变量等抄袭的目的。

关键词 ：抽象语法树 ；AST ；改进算法
中图分类号 ：TP309 文献标识码 ：A 文章编号 ：1671-1122（2014）01-0038-05

An Improved Algorithm based on Abstract Syntax Tree for Source Code Plagiarism Detection

LIU Nan[1],HAN Li-fang[1],XIA Kun-feng[2],QU Tong[2]
(1.*China Electric Power Research Institute,Beijing* 100192*, China;* 2.*School of Computer Since, Beijing University of Posts and Telecommunications, Beijing* 100876*, China*)

Abstract: Among the source code plagiarism detection algorithms used in software engineering, the algorithm based on abstract syntax tree (AST) can effectively detect those plagiarized cases of copying with no modification, modifying variable names and changing the source code sequence, but the algorithm can not detect the cases of modifying the variable type, adding no useful variables and so on. In this paper, we propose an improved algorithm based on abstract syntax tree, which is implemented by cutting out the syntax tree leaf nodes that may affect the judgment. This improved algorithm can positively detect the plagiarism cases described in the previous.

Key words: abstract syntax tree; AST;improved algorithm

0 引言

随着科技的发展，软件产业的环境不断改善，新技术的快速涌现，促进了软件产业的飞速发展，每天出现在市场上的新软件越来越多。但是在这众多的新软件中，也不可避免的出现了软件质量参差不齐的情况，对软件源代码的克隆、抄袭等现象也日益涌现，软件的抄袭主要表现为完全拷贝代码，或者在此基础上再进行一些不影响代码功能的修改等。常见的对于代码的修改有修改方法的访问类型、改变变量属性和添加或删除类等[1]。因此，针对软件源代码抄袭的检测技术在软件的剽窃检测和评估工作中起到非常重要的作用，并且在这样的背景下，出现了很多针对软件抄袭检测技术的研究成果。现在比较常用的软件源代码抄袭检测工具主要是基于文本的、基于 Token 的和基于代码语法结构的[2]。

基于文本的抄袭检测技术一般是将源代码转化为字符串，通过对字符串的查找匹配检测克隆代码。Johnson 做过相关研究，提出了将源代码划分为一系列字符串并使用字符串的指纹信息进行字符串匹配的思想。Ducasse 等人更进一步提出先对源代码进行过滤，除去代码中的“ 噪音 ”（注释、空格、换行等），再计算代码单行 Hash 值，通过匹配代码行 Hash 值实现抄袭检测。

收稿日期 ：2013-10-10
基金项目 ：国家自然科学基金 [61170268, 61100047,61272493]、国家国际科技合作专项 [2013DFG72850]、973 计划 [2012CB724400]
作者简介 ：刘楠（1979-），男，北京，工程师，硕士研究生，主要研究方向 ：信息安全 ；韩丽芳（1984-），女，内蒙古，工程师，硕士，主要研究方向 ：信息安全 ；夏坤峰（1989-），男，内蒙古，硕士研究生，主要研究方向 ：信息安全 ；曲通（1992-），男，山东，本科，主要研究方向 ：信息安全。

基于 Token 比对的源代码抄袭检测技术就是将源代码的每个词转化为一个标记，通过比对标记来判定抄袭 [3-7]，其算法研究较多，CP-Miner、CCFinder 等众多比对工具都采用了这种比对技术。在基于 Token 的源代码检测技术基础上，Muddu 等学者最近提出了一种 Robust 技术，这种技术基于一种 language aware token 序列，同时又融合了抽象语法树一些技术。它首先遍历由 JDT 解析获得代码的语法树来获取代码的 token 序列，然后通过 K-gram 技术把得到的 token 序列分割成一系列的 K-gram 并将其存储在索引中，最后通过比对 K-gram 来定位源代码抄袭。这种技术在一定程度上弥补了 LCS 算法的缺陷。王伟等在基于 token 的技术研究方面提出了 Token 流分析和序列挖掘相结合的方法，将克隆代码和克隆代码引起的缺陷进行联合检测，降低克隆代码误检和漏检对标识符重命名不一致缺陷检测的影响。于世英把聚类引入到代码同源性检测之中，提出了一种基于序列聚类的检测算法。该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。

在基于 token 检测的技术上再深入一步就是基于软件源代码语法结构的检测技术。现在已经有一些从源代码语法结构的层次进行代码抄袭检测和比对的研究成果，比如利用欧式向量空间记录语法树信息进行比对的 DECKARD，由 Wahler 等人提到的用 Xml 记录语法树信息来比对的方法，同时还有一些在线比对系统，比如 Moss，以及 Prcchelt L 等人开发的 JPlag。Baxter 等人也提出了一种利用源代码语法树进行代码抄袭检测的算法，并且有名为 CloneDr 的软件克隆检测软件。但是他的算法在整个比对过程中都保持了树形结构，需要进行多次树的遍历。基于抽象语法树的同源比对是语法树比对方法的一种改进，它对语法树进行了两次存储格式的变换，将树形结构转换为线性链表，并且按照子节点数进行了语法树子树的分组，并按照 Hash 值对语法树子树进行了排序，从而提高了比对的效率，该算法还对交换位置后语义发生改变的运算（例如减法和除法）进行特殊处理，降低了误报率。然而，基于抽象语法树比对方法仍然具有局限性，它对那些修改底层数据结构的抄袭无法检测或检测误差较大。

随着技术的发展，一些新的抄袭检测技术被提出来，包含有基于语义、编码风格和数据挖掘方面的方法。其中基于语义的抄袭检测技术能够从语义方面判断代码结构，而不受代码形式的影响，具有很高的准确度。目前已经形成了一些实际应用的算法，大都是基于程序依赖图（PDG），它将源代码变换为图形化的 PDG，再基于子图同构的概念对代码克隆进行检测。2012 年，Chan 等提出了一种基于堆图（Heap Graph）的抄袭检测方法，它通过获取软件的 birthmark 来判断抄袭。这种算法能够有效检测出针对 JavaScript 程序的抄袭。Cosma 等使用 Latent Semantic Analysis（LSA）方法提高了抄袭检测的准确度。基于编码风格的抄袭检测方法是由 Arabyarmohamady 提出的，这种抄袭检测方法不仅适用于程序设计语言，同时也适用于自然语言，它主要分两个阶段进行：在第一阶段，提取文件的主要编码风格特征；在第二阶段，用在三种不同的模块来检测抄袭的代码以确定抄袭的位置。WANG 针对学生程序抄袭现象提出了一种基于数据挖掘的算法。算法使用 CloSpan 算法挖掘程序中相似的代码，然后计算相似度，最后输出相似列表。在国内，这是数据挖掘首次应用到同源性检测之中。这些技术融合了多门学科知识，极大地提高了抄袭检测的准确性，不过这些技术还停留在学术研究上，并没有推广到实际的大规模应用中。

在本文中，我们提出了一种基于抽象语法树的改进思想，该思想通过剪去语法树中影响判断的叶子节点的手段来还原检测原文抄袭，能够达到有效检测修改变量类型和添加无意义变量等抄袭的目的。

1 基于 AST 的改进算法

1.1基于抽象语法树算法的介绍

基于抽象语法树（AST）的抄袭检测算法是一种基于语法结构的检测方法，它依靠源代码的语法树节点信息来判断代码的抄袭情况。该算法的步骤主要如下：

首先，通过调用词法和语法分析程序来获得源代码文件的语法树结构；获取源代码的语法结构是进行抄袭检测的基础，图 1 为一个程序源代码中的一个函数，图 2 为该函数经过词法和语法分析后得到的语法树结构。

```
01  void fun1
02  {
03  int a=0;
04  int b=0;
05  fun2(a,b);
06  }
```

图1 函数fun1源代码

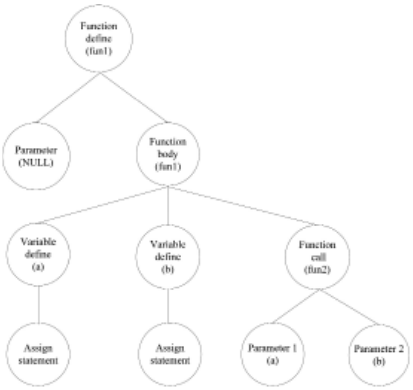

图2 函数fun1语法树

其次，调用抽象语法树的 Hash 生成算法，遍历语法树，分别得到文件语法树各个节点的 Hash 值等相关信息；在遍历过程中对于语法树的所有节点计算其 Hash 值及子节点数，还有对应于源代码中的起止行号。

最后，调用 Hash 比对算法将参与比对文件语法树中各个节点 Hash 值进行比较，记录下 Hash 值相同的节点和其对应于源代码中的位置，并计算出相似度。

1.2 基于AST算法的改进描述

在软件工程中，对于大部分抄袭程序来说，如果抄袭者只是对程序的函数名称及底层数据进行相应的修改，并没有触动程序的高层设计，在基于抽象语法树的同源比对算法中，由于一个语法树节点 Hash 值为其子节点 Hash 值之和，所以函数名称的改变并不影响函数节点的 Hash 值，不会带来误判。然而大多数的抄袭者会修改抄袭代码的底层数据，比如：增加无意义的变量、修改变量的类型；底层数据的修改（尤其是类型、数量等修改）很可能会改变树种节点的 Hash 数值，造成程序的错误判断。例如：如图 3 所示，树 A 为源程序部分代码对应的语法树，树 B 为抄袭程序相应代码部分对应的语法树，假设树 A 中 1 节点与 B 中 1 节点、A 中 2、3 节点与 B 中 2、3 节点类型相同，树 A 中 4,5,6 节点及树 B 中 4,5,6,7 节点均为同类型数据，因此其 Hash 值均相同，但是由于抄袭程序在程序中增加了变量使得树 B 中 2 号节点的 Hash 值相对源程序 2 号节点的 Hash 值发生改变，进而影响根节点的 Hash 值，造成检测程序误判。

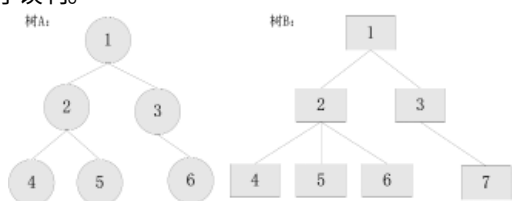

图3 语法分析程序生成的抽象语法树

这种对底层数据修改的抄袭行为从一定程度上躲避了工具的检测，降低了被检测出的概率。然而，我们通过分析上面语法树的结构不难发现，如果删除底层的叶子节点，那么对树节点的 Hash 值的影响就会消除。因此，我们对基于抽象语法树的同源比对方法进行了如下改进：

1）删除语法树的底层叶子节点，然后比对两棵树的根节点。这里，我们认为，如果两棵树顶层 Hash 值不相同，那么该树对应的源程序一定不同源。对于上面的例子，删除叶子节点后，如图 4 所示，由于树 A 中节点 2、3 与树 B 中对应节点类型相同，故 Hash 值也相同，因此树 A 与树 B 根节点对应 Hash 值相同，同时也说明了两者具有同源性。

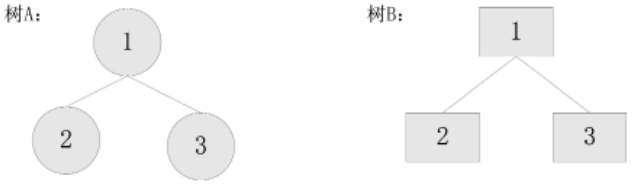

图4 去除叶子节点后的语法树

2）对叶子节点及去除叶子节点的语法树按照 Hash 值进行排序。排序的目的是减少下面进行叶子节点插入的复杂性，找出源树与目标树节点的对应关系。由于类型相同的节点其 Hash 值也相同，所以需要特别注意如图 5 中一对多情况，即两棵树相应类型节点如何对应，这里采用尝试的方法：让树 A 中 2 号节点分别与树 B 中 2、3 号节点相对应，算出每种状态下的相似程度。

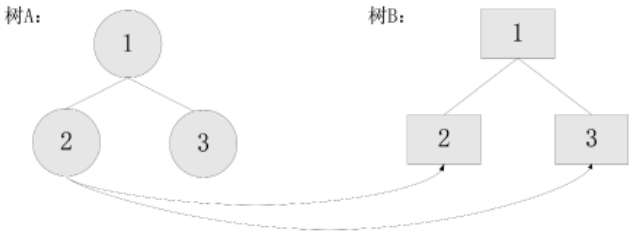

图5 抽象树中1对多的关系

3）把排序后叶子节点按照 Hash 值大小相对应的加入到树中，同时比较两棵树根节点的 Hash 值变化情况，如图 6 所示。如果根节点 Hash 值发生变化，则说明刚刚加入的叶节点不同，同时去除这些异同节点，继续向树中增加叶节点并观察树的变化情况。

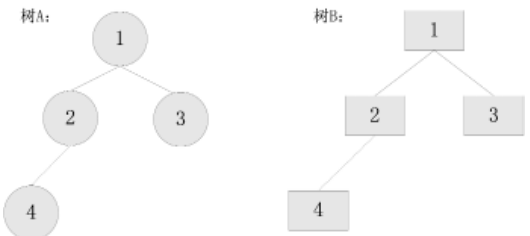

图6 叶子节点逐个添加到已比对的树中

1.3 算法流程

通过词法及语法分析程序，我们能够获得源代码带有节点 Hash 值的语法树，遍历该语法树可以获得语法树的各个叶子节点，为了避免这些叶子节点对语法树比对判断的影响，我们删除这些节点并修改其父节点的 Hash 值，这里删除节点并不是完全抛弃这些节点，而是把它们存放到一个专门的数组中，接着，对数组按照 Hash 值大小进行排序。完成上面所有的准备后，就进入了算法的核心步骤 ：把叶子节点逐个加入到语法树中（如图 6 中所示），加入节点时要时刻监视两棵树的根节点的 Hash 值的变化，如果 Hash 值依然相同，则刚刚加入的节点必相似，那么把这些相似的节点记录到相似节点的表中 ；如果两树的根节点 Hash 值不相同，则说明刚刚加入的节点不相似，把这些不相似的节点从树中取下并记录到另一张表中。按照上面的方法，直到所有的叶子节点都被测试（即 ：叶子节点数组为空）。其算法流程如图 7 所示。

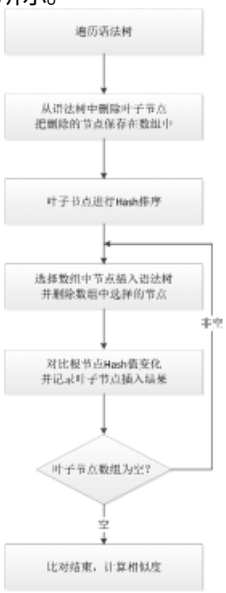

图7 算法流程图

1.4 相似度计算

相似度是评判一个程序代码抄袭的程度的重要指标，基于抽象语法树的比对方法的相似度计算依靠语法树的节点数目，它的计算公式如下所示 ：

$$sim = \frac{num_{sim}}{num_{total}}$$

其中 ：sim 为相似度，num_{sim} 为相似节点个数，num_{total} 为源文件节点总数。

从计算公式中，我们不难看出，“ 树中所有节点的个数 ” 是固定不变的，而“ 相同节点的个数 ” 跟我们所选择的算法有很大关联，针对抽象语法树的算法由于无法检测出基于底层数据修改的抄袭代码，那么它计算所得到的“ 相同节点的个数 ” 相对偏低，因而相似度的计算值相对偏低，使得误差增大 ；基于改进思想的算法能够准确检测出对底层数据修改的抄袭，相对更加准确地计算出程序中“ 相同节点个数 ”，因此，其相似度计算值更加贴近实际。

2 算法验证及实验结果分析

根据算法的内容，我们通过大量的代码检测对算法进行了验证，结果证明该算法能够有效检测出针对基本数据修改的抄袭行为，表 1 为针对三种常见的底层数据修改抄袭行为的实验结果统计，表 2 为测试代码比对结果的准确度 ：

表1 三种常见的基本数据修改抄袭检测结果

检测算法 / 抄袭方式	未改进的抽象语法树算法	改进的抽象语法树算法
变量类型改变	不可检测	可以检测
增加无效变量	可检测	可以检测
改变参数顺序	可检测	可以检测

表2 三种常见的基本数据修改抄袭比对结果准确度

检测算法 / 抄袭方式	未改进的抽象语法树算法	改进的抽象语法树算法
变量类型改变	0	98.6%
增加无效变量	83%	100%
改变参数顺序	95%	100%

下面选取一个实例进行分析 ：图 8 为一个程序源代码中的一个函数 ；图 9 是对图 8 的修改，在函数中增加了一个 int 类型的无意义变量 z。

```
01  void printResult(int x, int y)
02  {
03  int m;
04  m = x — y;
05  printf("%d",m);
06  }
```

图8 代码1：原始文件代码

```
01  void printResult(int x, int y)
02  {
03  int z, m;
04  m = x - y;
05  printf("%d",m);
06  }
```

图9 代码2：抄袭文件代码

图 10 为图 8 中代码的语法树，图 11 为图 9 中代码的语法树。从语法树中我们可以看到，图 11 中“ 变量声明 ” 节点比图 10 中的同一节点多了一个叶子节点，其他节点均

相同。因此，程序对节点进行 Hash 赋值时，同类型节点 Hash 值相同，图 10 中“ 赋值运算 ”、“ 输出指令 ” 节点对应的子树 Hash 值与图 11 中同类节点对应的子树 Hash 值完全相同。由于图 10 中“ 变量声明 ” 节点有 3 个 int 类型的叶子节点，其 Hash 值就等于 3*(hash_int)，而图 11 中“ 变量声明 ” 节点含有 4 个 int 类型叶子节点，其 Hash 值等于 4*(hash_int)，进而通过计算可得，两树的“ 函数定义 ” 节点 Hash 值也不相同。如果采用未改进的算法，结果会判定代码 2 部分抄袭代码 1，相似度为 ：83.3%。

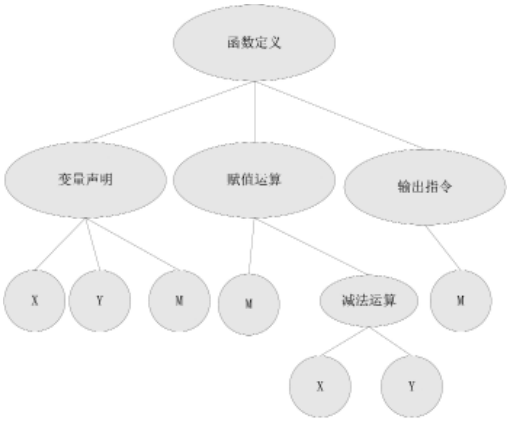

图10 原始文件源代码语法树

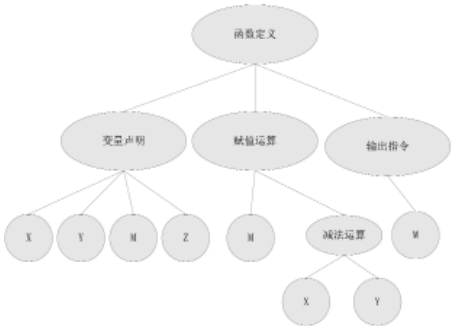

图11 抄袭文件源代码语法树

现在采用改进算法进行分析，由于两语法树只有“ 变量声明 ” 节点所在子树不同，所以，只需对该节点使用剪枝算法，图 12 为两树剪枝后的语法树，此时的语法树的所有节点的 Hash 值均对应相等，因此剪枝后的语法树完全相似。然后，对剪枝后的语法树逐个进行叶子节点插入操作，同时比对两树“ 变量声明 ” 节点的 Hash 值，当两棵树均增加了 3 个叶子节点后，“ 变量声明 ” 节点的 Hash 值保持相同，此时语法树即为代码 1 的语法树，如图 10 所示，当代码 2 继续进行最后一个叶子节点插入后，其“ 变量声明 ” 节点与“ 函数定义 ” 节点的 Hash 值将相对增加而与代码 1 的不同，如图 13 所示，可以判断该叶子节点是影响判断的节点，从语法树中取下并记录到程序列表中，那么，此时两树的叶子节点均进行了插入操作，因此可以判定代码 2 中与代码 1 中不同节点个数为 1，因此通过 1.4 中相似度计算公式可得，代码 2 与代码 1 的相似度为 ：100%。

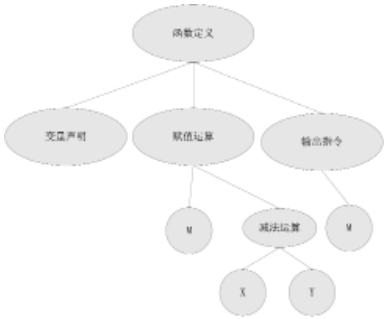

图12 “ 变量声明 ” 节点剪枝后的语法树

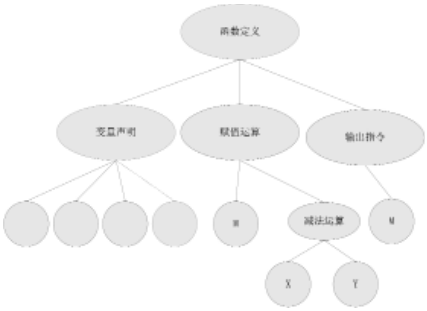

图13 插入的节点与hash值发生变化的节点

3 结束语

本文在对提出了一种基于抽象语法树的改进算法，该算法在比对过程中对可能影响判断的叶子节点进行剪枝比较，即 :先从树中删去叶子节点，然后逐个添加比较。最后，通过大量源代码抄袭检测实验对算法进行了验证，该算法能够有效检测出针对变量类型改变、增加无效变量和变量顺序改变等抄袭手段，提高了针对这些抄袭手段检测的准确度。（责编 杨晨）

参考文献
[1] 刘欣，段云所，陈钟 . 程序可执行代码同源性度量技术研究 [J]. 软件学报 ,2004，(15) ：141 - 148.
[2] 刘文伟，刘坚 . 一个重建 GCC 抽象语法树的方法 [J]. 计算机工程与应用，2004，(18) ：125 - 128.
[3] 阮彤，沈备军，居德华，等 . 面向对象软件度量工具的软件结构 [J]. 计算机研究与发展，2000，37 (4) ：401 - 406.
[4] 向永红，李更生，袁勇，等 . 串的最大匹配算法 [J]. 计算机工程与科学，2003，25 (4) ：72 - 74.
[5] 高灿 , 侯秀萍 , 孙士明 . 基于抽象语法树的修改影响分析方法 [J]. 长春工业大学学报 (自然科学版)，2012,8(04):387 - 390.
[6] 王伟，苏小红，马培军，等 . 标识符重命名不一致性缺陷的检测 [J]. 哈尔滨工业大学学报，2011，(01) : 89 - 94.
[7] 于世英，袁雪梅，卢海涛，等 . 基于序列聚类的相似代码检测算法 [J]. 智能系统学报，2013，(02) ：52 - 57.