Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

A novel approach in detecting code clones in Java using DFS

Vishwachi Choudhary *, Sonam Gupta

Department of Computer Science and Engineering, Ajay Kumar Garg Engineering College (AKGEC), Ghaziabad, India

ARTICLE INFO

Article history:
Received 9 January 2017
Received in revised form
5 April 2017
Accepted 7 April 2017

Keywords:
Clones
Types
Abstract syntax trees
Templates

ABSTRACT

Code is the rudimentary element of any software. Code clones may be defined as the segments of the program which are akin to one another. The similarity may be either syntactic or semantic. Cloning is easy to implement but hard to detect. Many researches have been carried out in order to find the methods for detecting these clones of code as problems are encountered at the time of maintenance due to these clones in codes. This further increases the cost of maintenance. The objective of our work is to precisely detect the code clones. Here, an approach is proposed based on the Abstract Syntax Tree method. The purpose for adopting AST is that it gives better detection results as compared to other techniques and is considered to be the best approach for detecting type 3 code clones. Furthermore, AST offers syntactic knowledge which can be leveraged to filter certain types of clones. The results obtained clearly shows that the technique adopted is able to precisely detect the near-miss clones as compared to the tools namely NICAD and CLAN.

1. Introduction

Code cloning is the act of copying the segments of code and pasting it to another place. At the first glimpse it seems to be a fascinating concept as the programmer doesn't need to write the same code again and again if the working of two code segments needs to be similar, but copy-paste strategy is a short term win.

Copying the code from one position and pasting it to another has various pitfalls which come into sight at the time of maintenance and testing of the software. If there are complications in the original code that was pasted it will be disseminated to the cloned/pasted segment too. For example, if a programmer makes any slight modification in the code and if the same change is not made in the cloned part then it may produce inconsistencies. In the large software systems it becomes really strenuous to uncover where this code has been reused. Searching in entire program is time consuming and practically an infeasible job. Clones produces bad impact on the design and also on the system improvement and modification as it is quite common that the person who developed the original system is not the one who is maintaining it. In the long run, the software may become so complex that even minor changes are hard to make. Clone detection came into existence to solve this problem.

With the help of clone detection technique, we can easily find out where the clone exists and can remove them beforehand so that they don't create any problem in future.

The studies reveal that almost (5-10 %) of the source of large computer programs is duplicated code (Baxter et al., 1998).

2. Types of code clones

There are various levels of clones as identified by Bellon et al. (2007). They are:

- TYPE-1: the codes which are exactly similar to one other without any kind of difference in the source code are placed under Type-1 clones. They may also be termed as syntactically similar codes.
- TYPE-2: the codes which are similar to each other except some of the changes in the white spaces, variable names, data type, arguments etc. are put under Type-2 code clones. They are also syntactically similar codes.
- TYPE-3:the codes with further modifications allowed in the source code like some of the additional code lines may be added or the ones present in one may not be present in another but both performing the same function are placed under Type-3 code clones.

* Corresponding Author.
Email Address: vishwachi.choudhary@gmail.com (V. Choudhary)
https://doi.org/10.21833/ijaas.2017.05.004
2313-626X/

- TYPE-4: they are semantically or behaviorally similar code segments. They don't have anything common in the source code but the functions performed by them are exactly the similar of each other.

The example of each kind of the clone is given in Table 1.

Table 1: Example of four types of clones

Source code (a)	Type-1 clone(b)	Type-2 clone(c)	Type-3 clone (d)	Type-4 clone(e)
$int\ main()$ $\{int\ a = 1, int\ x$ $= a + 2; return\ x;\}$	$int\ main()$ $\{int\ a = 1;$ $Int\ x = a + 2;$ $return\ x;\ //\ output\}$	$int\ func()$ $\{int\ c = 1;\ int\ q$ $= c + 2;$ $return\ q;\}$	$int\ main()$ $\{float\ s = 1; float\ t = s + 2; t$ $= t/++s;\ return\ t;\}$	$int\ func2()$ $\{int\ s = 2;$ $return\ ++s;\}$

3. Root causes for code clones

A study by Kontogiannis et al. (1996) reveals why programmers just copy and paste the code. They identified the following reasons by observing the programmers in their daily practice:

- Sometimes it may be due to the short time limits given to the programmers by the client for the development of the software.
- Systems are modularized based on the principles such as minimizing coupling, information hiding and maximizing cohesion. In the end –at least for the systems written in ordinary programming languages- the system is composed of fixed set of modules (Koschke, 2007). Ideally, if the system needs to be updated, only few modifications will be required.
- Another root cause is that programmers often reuse the copied text as a template and then customized the template in the pasted context (Koschke, 2007). Other potential reasons such as time pressure, educational deficiencies, development process, and short sightedness must also be investigated.
- Phobia of fresh code.
- Complexity of the system.

4. Clone detection methods

There are various methods of detecting the clones which includes:

4.1. Text based

They are language independent and provide an easy way to detect the clones among various programming languages. The major shortcoming of this method is that it can detect only Type-1 clones along with some of the Type-2 clones which minor changes such as different formatting style.

4.2. Token based

In this technique, the code is first of all transformed into the token sequence. After that the sequence is formed from some set of tokens which are then compared to find the clones. The major advantage of token based technique is that it is fast with higher recall values.

4.3. Syntax tree based

Here, we use the parser to build parse trees or abstract syntax trees from the source code. The trees thus obtained can be processed further using the tree- matching to find the clones.

Roy et al. (2009) explained that the abstract syntax tree or parse tree contains the complete information about the source code. In order to find the clones using the syntax tree approach, the sub-trees are compared and those which come out to be similar are considered as the clones. The code corresponding to these sub-trees are returned as clone pairs.

4.4. Graph based

A program dependency graph (PDG) represents control and data flow dependencies of a function of source code (Rattan et al., 2013). In other words, it considers the semantic information encoded in the dependency graph. Clones may be identified as isomorphic sub-graphs in a program dependency graph (Krinke, 2001).

4.5. Metrics based

In Metrics- based approach, a number of metrics are assessed for the code segments which can involve the number of lines, number of input statements, number of output statements, return statements, function calls etc. in each of the segments. The metric values are then compared instead of the source code directly. The two segments whose metrics values comes out to be similar to each other are considered as clone pairs.

5. Proposed approach

Observing the advantages and disadvantages of various techniques developed so-far, here abstract Syntax Tree based approach is used to detect the code clones. Our approach will find the syntactic clones in linear time and space.

Here we used the Depth First Search (DFS) algorithm which is an algorithm for searching in a tree. One starts at the root and explores as far as possible along each branch before backtracking. The approach adopted is as follows:

1. Firstly the code will be passed into the ANTLR parser. ANTLR (another tool for language

recognition is a parser generator that uses LL (*) for parsing (https:// en.wikipedia.org/wiki/ANTLR). ANTLR can generate lexers, parsers, tree parsers and combined lexer parsers (https:// en.wikipedia.org/wiki/ANTLR). The purpose of doing so is to obtain the syntax tree representation of the code. The example of AST formed for a particular code is (Fig. 1):

$x = a + b;$
$y = a * b;$
$while(y > a)$
$\{a = a + 1;$
$x = a + b;$
$\}$

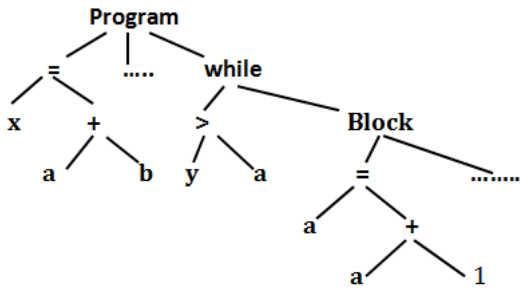

Fig. 1: AST for source code

2. The representation of tree acquired goes through the steps discussed below:

- DFS (Depth First Search) is applied to both the trees in parallel.
- Then for each of the node of the tree, convert it into the template. The procedure for template conversion is as follows:

  - Template conversion is the procedure of converting the source code into a new form which is uniform intermediate representation of source code.
  - Type 1 clones are exactly similar to each other so there is no need to convert them into templates.
  - For type 2 clones, the clone methods may contain difference in names of variables ,identifiers, data types, white spaces etc. for converting them into template, we can replace all the identifiers names into a common name as 'X' and all the data types into a common data type 'DATA'.
  - For type 3 and 4: in case of type 3-4 clone detection, various constructs like branches, iterations can also be changed. Therefore we need a general method for converting them into a form which is common. The method for the conversion is given in the Table 2:
  - Then for each node (converted into template) check if the children of the node in the tree exist. If it exists, store them in prefix order in an array (apply this procedure on both the trees whose nodes are now present in the form of templates)
  - Compare the elements in both the arrays. If similar elements exist, store them in a separate list.
  - Now, for all the elements/nodes which exist in the list, apply Levenshtein (1966) distance algorithm to find out the distance between the nodes.

- It is applied considering two nodes at a time and comparing them element-by-element.
- If the two nodes comes out to be
- Exactly similar, their cost will be set as 0 otherwise 1 in the opposite case.
- Now for all the pairs of nodes in the tree whose Levenshtein (1966) distance/ cost comes out to be 0 are stored in an array and are marked as the clone pairs.

Table 2: Template conversion method

No.	Equivalence category	Possible constructs	Proposed pattern
1.	Iterative equivalence	For while do-while	Iteration <initial> <condition> <inc/dec>
2.	Conditional equivalence	If else else-if ?: Switch	Selection <condition>
3.	Input equivalence	Scanf system.in input.readline	Read<variable>
4.	Output equivalence	Printf system.out	Write<variable>
5.	Declaration equivalence	int char float double string	Multiple declaration to single declaration
6.	Braces	{ }	Braces are removed in the code

6. Results

The proposed approach has been tested on various open source software available. The implementation is done with the help of the self-created tool with input of JAVA project files. The tool is able to find out precisely Type1, Type 2 and Type 3 code clones. The project sources used is shown in the Table 3.

Table 3: Open source projects used

Project name	Lines of line of code
Java Netbeans-Javadoc	14K
Spule	13K
EIRC	11K
Eclipse-ant	35K
JHotDraw	40K

The source codes of the above projects are fed into our system and the clones are detected in their source codes. The results obtained are as follows in Table 4. The results obtained in the form of clone pairs are in Table 5.

7. Comparison with existing tools

The tool developed using the proposed approach is being compared with the existing tools. The two tools are used other than the proposed one. They are NICAD and CLAN. They all are applied onto the projects. The results obtained are in Figs. 2 and 3.

Table 4: Result of proposed approach

Project Name	Type-1 clones	Type-2 clones	Type-3 clones
Java Netbeans-javadoc	196	205	300
Spule	60	70	125
EIRC	122	125	160
Eclipse-ant	380	370	445
JHotDraw	303	320	640

Table 5: Results in clone pairs

Project Name	Type-1 clone pairs	Type-2 clone pairs	Type-3 clone pairs
Java Netbeans-javadoc	190	200	302
Spule	60	68	115
EIRC	116	121	148
Eclipse-ant	360	370	420
JHotDraw	290	301	595

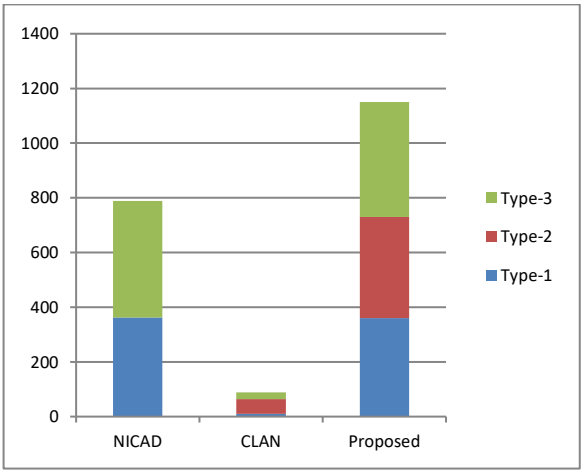

Fig. 2: Clones in eclipse-ant

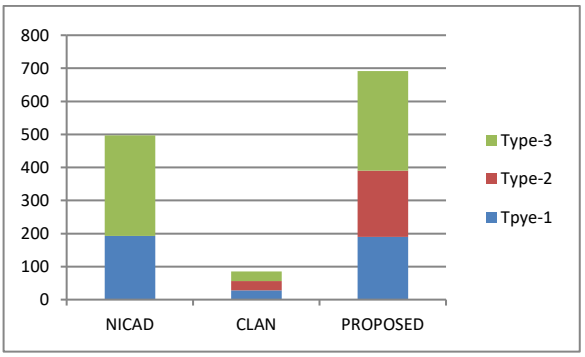

Fig. 3: Clones in java net beans

8. Conclusion and future work

In this paper, we have proposed an approach to detect Type-1, 2 and 3 code clones. The proposed approach quickly detects Type-2 and 3 clones which normally are not being detected by all the existing approaches and if they do so, then not as precisely as the proposed approach.

In this approach we are able to feed only a single source code file at a time. For future work, we may apply the detection at the directory level which may contain multiple numbers of files in it and detects the clone pairs in them.

References

Baxter ID, Yahin A, Moura L, Sant'Anna M, and Bier L (1998). Clone detection using abstract syntax trees. In the International Conference on Software Maintenance, IEEE, Bethesda, USA: 368-377. https://doi.org/10.1109/ICSM.1998.738528

Bellon S, Koschke R, Antoniol G, Krinke J, and Merlo E (2007). Comparison and evaluation of clone detection tools. Transactions on Software Engineering, 33(9): 577-591.

Kontogiannis KA, DeMori R, Merlo E, Galler M, and Bernstein M (1996). Pattern matching for clone and concept detection. Journal of Automated Software Engineering, 3(1–2): 77-108.

Koschke R (2007). Survey of research on software clones. In the Dagstuhl Seminar on Duplication, Redundancy, and Similarity in Software, 06301, LZI, Merzig, Germany: 1-24. Available online at: http://drops.dagstuhl.de/opus/volltexte/2007/962/pdf/06301.KoschkeRainer.962.pdf

Krinke J (2001). Identifying similar code with program dependence graphs. In the 8th Working Conference on Reverse Engineering, IEEE, Stuttgart, Germany: 301-309. https://doi.org/10.1109/WCRE.2001.957835

Levenshtein A (1966). Binary codes capable of correcting deletions insertions and reversals. Soviet Physics Doklady, 10(8): 707-710.

Rattan D, Bhatia R, and Singh M (2013). Software clone detection: A systematic review. Information and Software Technology, 55(7): 1165-1199.

Roy CK, Cordy JR, and Koschke R (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of Computer Programming, 74(7): 470-495.