Structural similarity detection using structure of control statements

Sudhamani M\(^a\)*, Lalitha Rangarajan\(^b\)

\(^a\)Department of Computer Science, University of Mysore, Mysore 570006, India

Abstract

In this work we address code clone or duplicated code detection which is one of the major factors that degrades the design and structure of software and lowers the software qualities such as readability, changeability and maintainability. Copying and pasting source code is common practice to reuse the software. While doing so, it is unlikely that structure of control statements is altered. The proposed method aims at detecting segments that have similar control statement structure. The method is language independent and can be executed on C, C++ or Java. The method can efficiently identify structurally similar clones.

1. Introduction

Software clones are similar code fragments that are repeated more than once in the software systems. Duplicated code may appear in several other parts of the same program or in different files of the same application program. There are several reasons to copy already existing code. Reusability is one of the reasons for copy paste activity. Despite being useful, such clones are harmful for software maintenance and evolution. For example, if there is a bug in a copied fragment and that fragment is reused then the bug is essentially being propagated in the software. Code clones are problematic for two reasons.

- Increase of maintenance cost due to unnecessary duplication of code.
- Incorrect program behavior due to incorrect changes to cloned code.

* Corresponding author. Tel.: +91-988-027-4580.
E-mail address sudhamysore@gmail.com
Code clone is a result of copy paste activity. Unauthorized copy paste of code results in plagiarism. This is a serious problem in recent days. Doing minor changes to the copied code such as changing loop statement is an easy task. Fig.1(a) and (b) shows two programs to do bubble sort with different loop statements using ‘for’ and ‘while’. Modification of control statement sequence is difficult to do without affecting the output of the program. For example the ‘if’ statement in the second ‘for’ loop in Fig 1.(a) is moved out of this ‘for’ loop to elsewhere the output will not be same. Alteration that yields same output as the original code requires time and thinking, persons engaged in illegal copy paste activity are not likely to have either.

![Fig.1 (a) Bubble sort 1 program; (b) Bubble sort 2 program.](image)

Basically there are two kinds of similarities.

1. **Textual similarity**: Two code fragments are similar if they are identical except change in variants (type I to type III below). These types of clones are results of copy paste.

2. **Functional similarity**: Functionalities of two code fragments are identical.

The following clone types were identified based on the kind of similarity that two code fragments can have.

- **Type I**: Exact identical types of code, except white space and comment.
- **Type II**: Structural similarity or syntactically identical fragments except change in identifier, literal and function names.
- **Type III**: Type III clone is result of copy paste with modifications like reordering/rearranging of copied statements or adding new statements or deleting some statements from the original code.
- **Type IV**: Two code fragments are functionally similar or semantic clones if the functionalities of two code fragments are same.

Software program is a collection of different types of statements like declarative statements, computational statements, logical statements, input-output statements and control statements. Execution of program depends on structure of control statements present in the program. Hence we calculate metrics based on control statements. There are several code clone detection techniques which use different types of metric values to compare two different programs. Mayrand et. al compute functional metrics using Datrix tool to detect function level metrics. The codes are first converted to Intermediate Representation of Language (IRL) and Abstract Syntax Tree. Points of comparison considered are Name, Layout, Expression and Control flow. Kontogiannis et. al designed two types of pattern matching technique. In first method numerical values of selected metrics are compared to characterize and classify source code fragments. In second method dynamic programming (DP) approach is used and statement level feature vectors are compared to characterize code program statements through best alignment between two
fragments in terms of deletion, insertion and substitution. Also these methods do not take into account the control statement structures for calculations of metrics. However the computation of metric in the proposed method considers structure of control statements and without the need of any transformation.

Control statements we have considered are selective statements (if, if-else, switch-case) and iterative statements (for, do-while, while and repeat-until). By structure of control statements, we mean the order of control statements in a program. Structural analysis of program is detection of usage of control statements and presence of other control statements inside. The paper is organized into five sections. Section 2 discusses the related work. Section 3 describes the implementation of the proposed system and the results. In section 4 we have described computation of one more metric and the performance is shown to be far better with the modified metric. The last section contains conclusion and future work.

Nomenclature

|   |   |
|---|---|
| C | Conditional statements |
| L | Looping statements |
| CST | Control structure table |
| D | Distance matrix |
| FMT | File metric table |

2. Related work

There is no consistent or precise definition in the literature for code clones. Identical or near identical source codes are considered to be clones by many researchers. Software clone detection is an active field of research. The following section describes different types of approaches to detect code clones.

- **String based**: String based techniques perform comparison based on characters. Comparison of two strings is usually done with calculating some form of edit distance. It is one of the simple and fastest clone detection approaches. It does not perform any syntactic or semantic analysis on source code. This method can detect type I clones. Tools based on string based technique are Dup and DupLoc.

- **Token based**: Each line of source code is converted into a sequence of tokens using lexer. Then these token sequences are compared efficiently through suffix tree algorithm or hashing algorithm. This technique is slightly slower than text based method, because of the tokenization process. This approach can detect both type I and type II clones. Tools based on token based technique are CCFinder, CPminer, JPlag, and CPD.

- **Tree based**: Source code is converted to Abstract Syntax Tree (AST) with appropriate parser. Then, using a tree matching technique similar sub trees are searched in the tree. When match is found corresponding source code of the similar sub trees are returned as clone pairs. The tools available using tree based technique are CloneDR, CCdiml, and Deckard.

- **PDG based**: Source code is converted to graph called Program Dependency Graph (PDG) where PDG contains control flow and data flow information of source code and hence carries semantic information. Then isomorphic sub graph matching algorithm is applied to find similar sub graphs which are returned as clones. This approach efficiently identifies type IV clones, but this technique is not scalable for large size programs and finding isomorphic sub graphs is NP hard problem. Tools based on PDG based technique are Scorpio, Duplix, PDG-DUP and GPlag.

- **Metric based**: Metrics-based approaches compute different metrics for code fragments and compares metric values to check similarity between two code fragments. This technique is scalable and straightforward. Tools created using the metrics method are eMetrics, Covet and Moss.
3. Proposed Model

The objective of the system is to find structural similarity of control statements, such as similar looping statements and conditional statements. Figure 2 shows the architecture of the proposed method. In preprocessing we remove white space, comments and empty lines from the input files. Program is scanned line by line to get information about order and nestedness of selective and iterative control statements. This information is recorded in a table called control structure table (CST). CST for bubble sort 1 and bubble sort 2 programs (figures 1(a) and 1(b)) are shown in table 1 and table 2. And we compute file metrics which contains information about total number of iterative and selective statements present in the particular program file and store this information on table called file metric table (FMT). These values are used in clone detection process. If control structures of two programs are similar then the respective programs are perhaps identical.

Distance matrix ($D$) is computed from two control structure tables. A row of program 1 (corresponding to a control statement) is compared with every row of program 2. Rows $i$ and $j$ of the program are compared using city block distance formula such as $|x_1 - x_2| + |y_1 - y_2|$. Table 3 shows distance matrix computed from table 1 and table 2. For example first row of table 1 is compared with second row of table 2 that is $|0-1| + |0-1|=2$ is entered in $(1, 2)$ of distance matrix (table 3). From this table we can find similar control lines present in two programs. Presence of zero in the position corresponding to similar control statement indicates structural similarity of that control statement.

![Fig.2. Architecture of proposed model](image)

| Sl.No | Type of control statement | L | C |
|-------|---------------------------|---|---|
| 1     | L                         | 0 | 0 |
| 2     | L                         | 1 | 1 |
| 3     | L                         | 0 | 1 |
| 4     | C                         | 0 | 0 |
| 5     | L                         | 0 | 0 |

| Sl.No | Type of control statement | L | C |
|-------|---------------------------|---|---|
| 1     | L                         | 0 | 0 |
| 2     | L                         | 1 | 1 |
| 3     | L                         | 0 | 1 |
| 4     | C                         | 0 | 0 |
| 5     | L                         | 0 | 0 |
For example zero at (1, 5) indicate similarity between first control statement (iterative statement) of program 1 with fifth control statement (iterative statement) of program 2 where as zero at (1, 4) indicate similarity of 'iterative statements' with 'conditional statement'. For each such zero we check the type of control statements whether it is looping statement or conditional statement. We count the zeros of only similar type of control statements.

We can calculate the similarity value ‘s’ using the formula given here.

\[
s = \frac{n}{\text{max}(D)}
\]  

where ‘n’ is the number of zeros in the distance matrix and ‘\text{max}(D)’ is the maximum dissimilarity between two control structures of two programs. This similarity measure shows high similarity between programs that are dissimilar. For example versions of beam search, bubble sort and linear search programs shows high similarity with queue programs. This is explained in detail in the experimental results section (section 4). Hence we have included additional count in computation of distance namely the total number of conditional and iterative statements present in the two programs. We compute similarity between programs only if programs are comparable in terms of number loops and conditional statements. A threshold of 20% variations in these numbers is fixed for computation of similarity. Similarity between two programs are computed only if these have \(x\) and \(x \pm 20\%\) (\(x\)) number of loops and \(y\) and \(y \pm 20\%\) (\(y\)) number of conditional statements.

### 4. Experimental results

The proposed method by similarity computation using formula (1) is tested with 15 distinct programs and 50 variants (data set 2). The similarity matrix of 5 programs and 15 variants (data set 1) followed by graph is shown in table 5 and graph 3(a). Here we observe that beam search (1 to 4), bubble sort (5 to 7), and linear search (11 and 12) show high similarity with queue programs.

You may observe that the program even if altered it is unlikely that more than 20% of loop and branch statements are introduced additionally or deleted from existing source lines. For this reason we compute similarity values only
if programs have comparable number of loops and branch statements. In table 4, the number of loops in queue program is 18 and 20% of 18 is 3.6. The number of loops in beam search is 2 which do not in the interval 18-3.6, 18+3.6. Hence these programs are not compared at all and these cannot qualify to clones of each other. The similarity matrix of 5 programs and 15 variants followed by graph is shown in table 6 and figure 3(b) (with consideration of FMT). You may observe that only the clones are picked with the new metric. We conducted experiment considering file metric table for 15 distinct programs with 50 variants and similarity graph is shown in figure 4. In fig 4 we can clearly observe the clone sets. The clones are generated by students, thus performance is also a measure of picking out plagiarized codes. No false positives are generated and no clones are deleted with the similarity measure that uses FMT.

**Time complexity:**

The control structure table and file metric table are generated in just one scan of the program. Thus the complexity for the two table generations for a program is \(\theta(m+n)\) where \(m, n\) are the number of source lines in the two programs. The complexity of distance matrix generation is \(\theta(m_1*n_1)\) where \(m_1\) and \(n_1\) are number of control statements in the two programs where \(m_1\) and \(n_1\) are far less than \(m\) and \(n\). The third step is to pick zeros in the distance matrix is \(m_1*n_1\) and compare file metrics of two programs is constant value 2. Thus the complexity is \(O(m_1*n_1)\) which is polynomial in order. AST and PDG based methods all examine structural similarity. However identifying subtree and isomorphic subgraphs is very complex.

| Program no. | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1          | 0  | 6.167 | 6.17 | 6.17 | 2.67 | 2.7 | 2.667 | 3.5 | 3.5 | 3.5 | 1.67 | 1.667 | 6.33 | 6.66 | 6.66 |
| 2          | 6.17 | 0  | 6.17 | 6.17 | 2.67 | 2.7 | 2.667 | 3.5 | 3.5 | 3.5 | 1.67 | 1.667 | 6.33 | 6.66 | 6.66 |
| 3          | 6.17 | 6.167 | 0  | 6.17 | 2.67 | 2.7 | 2.667 | 3.5 | 3.5 | 3.5 | 1.67 | 1.667 | 6.33 | 6.66 | 6.66 |
| 4          | 6.17 | 6.167 | 6.17 | 0  | 2.67 | 2.7 | 2.667 | 3.5 | 3.5 | 3.5 | 1.67 | 1.667 | 6.33 | 6.66 | 6.66 |
| 5          | 2.67 | 2.667 | 2.67 | 2.67 | 0  | 4  | 4  | 1.71 | 1.71 | 1.7 | 2.5 | 2.5 | 4.5 | 4.75 | 3.8 |
| 6          | 2.67 | 2.667 | 2.67 | 2.67 | 4  | 0  | 4  | 1.71 | 1.71 | 1.7 | 2.5 | 2.5 | 4.5 | 4.75 | 3.8 |
| 7          | 2.67 | 2.667 | 2.67 | 2.67 | 4  | 4  | 0  | 1.71 | 1.71 | 1.7 | 2.5 | 2.5 | 4.5 | 4.75 | 3.8 |
| 8          | 3.5  | 3.5  | 3.5  | 3.5  | 1.71 | 1.7 | 1.714 | 0  | 14.2 | 14  | 1.36 | 1.357 | 10.1 | 11  | 11 |
| 9          | 3.5  | 3.5  | 3.5  | 3.5  | 1.71 | 1.7 | 1.714 | 14.2 | 0  | 14  | 1.36 | 1.357 | 10.1 | 11  | 11 |
| 10         | 3.5  | 3.5  | 3.5  | 3.5  | 1.71 | 1.7 | 1.714 | 14.2 | 14.2 | 0  | 1.36 | 1.357 | 10.1 | 11  | 11 |
| 11         | 1.67 | 1.667 | 1.67 | 1.67 | 2.5 | 2.5 | 2.5 | 1.36 | 1.36 | 1.4 | 0  | 4  | 3.75 | 4  | 3.2 |
| 12         | 1.67 | 1.667 | 1.67 | 1.67 | 2.5 | 2.5 | 2.5 | 1.36 | 1.36 | 1.4 | 4  | 0  | 3.75 | 4  | 3.2 |
| 13         | 6.33 | 6.333 | 6.33 | 6.33 | 4.5 | 4.5 | 4.5 | 10.1 | 10.1 | 10 | 3.75 | 3.75 | 0  | 27  | 27 |
| 14         | 6.67 | 6.667 | 6.67 | 6.67 | 4.75 | 4.8 | 4.75 | 11  | 11  | 11 | 4  | 4  | 25.3 | 0  | 25.3 |
| 15         | 6.67 | 6.667 | 6.67 | 6.67 | 3.8 | 3.8 | 3.8 | 11  | 11  | 11 | 3.2 | 3.2 | 25.1 | 25.16667 | 0  |
Table 6. Similarity table with using file metric table

| Program no. | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1           | 0.0 | 6.166 | 6.166 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2           | 6.166 | 0.0 | 6.166 | 6.166 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 3           | 6.166 | 6.166 | 0.0 | 6.166 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4           | 6.166 | 6.166 | 6.166 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 5           | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 6           | 0.0 | 0.0 | 0.0 | 4.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 7           | 0.0 | 0.0 | 0.0 | 4.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 8           | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 14.214 | 14.214 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 9           | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 14.214 | 14.214 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 10          | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 14.214 | 14.214 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 11          | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 12          | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 13          | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 14          | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 25.333 | 0.0 | 0.0 | 0.0 | 0.0 | 25.3 | 0.0 | 25.3 |
| 15          | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 25.166 | 25.166 | 0.0 | 25.3 |

5. Conclusion and future work

The proposed method is simple straightforward and does not require any parser or lexer. The method efficiently identifies structurally similar programs as clones. Structure similarity of the programs is explored without transformation of source code into PDG (Program Dependency Graph) representation. This method is language independent and efficiently identifies all types of clones. The method proposed (particularly with additional consideration number of control statements (FMT) across programs) results in 100% correct classification.

The proposed method works best for C, C++ and java programs without object oriented concepts; in future the method would be extended to handle programs with object oriented concepts and to find functional similarity.

![Fig.3(a). Similarity graph without considering file metric comparison; (b). Similarity graph considering file metric comparison](image-url)
Fig. 4. Similarity graph for data set 2

References

1 Baker S., A Program for Identifying Duplicated Code, Computing Science and Statistics, vol. 24, 1992.
2 Johnson J H., Substring matching for clone detection and change tracking, in Proceedings of the International Conference on software Maintenance, 1994.
3 Baxter I D., Yahi H., Moura L., Anna M S., and Bier L., Clone Detection Using Abstract Syntax Trees, in proceedings of ICSM. IEEE, 1998.
4 Bellon S., Koschke R., Antoniol G., Krinke J., and Merlo E., Comparison and evaluation of clone detection tools, IEEE Transactions on Software Engineering, September 2007.
5 Ducasse, S., Rieger M., and Demeyer S., A Language Independent Approach for Detecting Duplicated Code. In Proceedings; IEEE International Conference on Software Maintenance, 1999.
6 Krinke J., Identifying Similar Code with Program Dependency Graphs, Proc. Eighth Working Conference, , Reverse Engineering., 2001.
7 Hotta K et. al., An Empirical Study on the Impact of Duplicate Code, Hindawi Publishing Corporation, Advances in Software Engineering., 2012.
8 Jiang L., and Glondu S., Deckard: Scalable and Accurate Tree-Based Detection of Code Clones.
9 Sadowski C., and Levin G., SimHash: Hash-Based Similarity Detection, 2007.
10 Zhang Q., et . al., Efficient Partial-Duplicate Based on Sequence Matching, 2010.
11 Roy C K and Cordy J R. A survey on software clone detection research. Tech. rep., 2007. TR 2007-541 School of Computing Queen’s University at Kingston Ontario, Canada.
12 Mayrand J, Leblanc C and Ettore Merlo M. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics, proc of ICSM conference 1996.
13 kodhai E, Perumal A, and Kannani S, Clone Detection using Textual and metric Analysis to figure out all Types of Clones International journal of Computer Communication and Information System (IJCCI)- Vol2, No1.ISSN:0976-1349 July-Dec 2010.
14 Vidya K and Thirukumar K, Identifying Functional Clones between Java Directory using Metric Based Systems International journal of Computer Communication and Information System (IJCCI)-Vol3, ISSN:2277-128x August 2013.
15 Anupama S, Phili S and Jacob K P. Code Clones in a program Test Sequence Identification, 2011 World Congress on Information and Communication Technologies, 12/2011.
16 www.research.cs.queensu.ca.
17 Kontogiannis K A, Demobl EMerlo M and Galler M Bernstein, Pattern Matching for Clone and Concept Detection proc of the second conference on Reverse Engineering pp:96-103,july, 1995.@IEEE,1995
18 Roy C K, Cordy J R and Koschke R, Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach, Science of Computer Programming, 74(2009) 470-495, 2009.