

DETECTION OF LOGICAL CLONE IN CODE USING DATA DEPENDENCY AND EXPRESSION LIST

¹SYED MOHDFAZALULHAQUE, ²V SRIKANTH, ³E. SREENIVASA REDDY

¹Maulana Azad National Urdu University, fazal.manuu@gmail.com

²K L University, srikanth_vemuru@yahoo.com

³Acharya Nagarjuna University, esreddy67@gmail.com

ABSTRACT

Code plagiarism is a main issue in various institutes and software industries. We don't have a perfect approach in detecting the code copied. In various countries like INDIA, USA and UK majority of industries and institutes have gained their own tool for detection of code plagiarism. *The developed tools will identify the code program similarities based on statements written in programming language.* Our proposed work is to develop an approach which detects the dependencies based on data. We consider data program for tracking similarities on code. *The list of expression and data dependencies are detected based on code copied.*

We prepared a dependency matrix which checks the dependencies of data in the program and compares with the list using efficient method.

Keywords: Detection, Code Cloning, Method Of Matrix, List Related To Expression, Dependency Data.

1. INTRODUCTION

Code cloning is a sensitive problem in this area. There are various issues related to plagiarism based on the previous works done in IEEE, This was stated in plagiarism editor [1]. Cloning is a basic un-expected increasing threat in the field of software development and academic research which leads to threat of integrality. These types of threats are basically used in information technology, when software engineers try to develop the software rapidly. It is not a fraud, but this issue can reduce the efficiency of code and its working behavior.

It is involved as a stealing process or procedure in copying the work of another. Based on the academia, code cloning is treated as a dishonest or fraud in terms of offence which is subjected to censure expulsion and dishonest. Various types of courses based on languages and their assignment have been made and written based on type of software to be developed.

Similar type of problems occurs or relate to the classes, because students don't pay much of the attention in writing the program or doing practicals. They basically do the process of paste and copy in

the program unconditionally. This type of tendency should change in the behavior of the students.

If any type of tool is available or developed which can identify the copied copy with the support of lecturer may be punishable to the students. In relate to the study, an academician should have a Toro information of the language and should teach the students clearly in such a way that the student should not re-use the available code.

This toro understanding between the student and academician can reduce the code cloning occurrence in development of software or code. Our proposed work is to develop a tool, which helps the academia in identifying the clones or similarities in the code. This is totally a new concept of identifying dependencies of data in the program and comparing with it.

2. RELATED WORK

We have studies various type of tools which are used to detect code plagiarism based on language keywords and instruction related to the language. We studied various tools which are used to detect the code clones, but they don't have the capability of detecting the line change or position change in a statement. i.e they cannot

detect metric structure of the system in terms of exact percentage.

University of Madrid has developed a tool named PK2, which is used to detect the compound assignment in a code with small fragments of given source code. This type of tools failure, when reshuffling is done in a statement or some additional information is added to it unnecessarily.

Dependency data matrix method is developed [9], this method is based on assignment of data statements in the program. This type of method is elaborated and checked for various cases.

Structure Of Expression Clone

Expression structure clones Expression structure is an abstraction over expression in which we ignore the specifics of measures, literals, and dimensions and keep the embedded functions and the order of operators. The example in Figure 4.2 will be used to better illustrate the details. The top rule shows how to calculate the percentage part of web sales in total sales, whereas the bottom one shows how to calculate percentage of regular sales in total sales. However, if we replace the measures with a, b, and c (in the order of their appearance from left to right) and the literal 0 with L, the two expressions are the same: $a = b / c$ where $c \neq L$ and thus their expression structure is the same. If two expressions have the same expression structure, we consider them to be expression structure clones. A reasonable assumption is that not all expressions in the application have the same structure. In the analysis, we first determine all different expression structures that occur, which we call expression structure patterns, and then we assign each expression to the corresponding pattern. That is, an application has multiple expression structure patterns

$$\text{WEBSALES_TG_P} = \text{WEBSALES_TG_CAD} / \text{TOTALSALES_TG_CAD} \quad \text{where} \\ \text{TOTALSALES_TG_CAD} \neq 0.$$

$$\text{REGCOST_FC_P} = \text{REGCOST_FC_CAD} / \text{TOTALCOST_FC_CAD} \quad \text{where} \\ \text{TOTALCOST_FC_CAD} \neq 0.$$

and each pattern has one or more expressions associated with it. We call those expressions instances of the particular expression structure pattern. This clone type was of particular interest since the number of expression structure patterns is very small compared to the total number of expressions. This was the case with all the applications we analyzed and the concrete results

will be reported. The results of our analyses show that some expression structure patterns are quite common and some of them are very well known in the domain. The measure language does not have any means of accelerating the creation of rules that have the same structure; we will use this fact in later sections when proposing the improvements to the language.

Corresponding metrics clones We define two expressions to be corresponding metrics clones if they share: the expression structure, and metrics in corresponding measures. Dimensions are ignored for this classification. Explanation of two expressions that satisfy these conditions. Both expressions have the same expression structure, and green color highlights the corresponding metrics, which are also the same: TOTALSALES, WEBSALES, and REGSALES (Regular Sales). Similarly, both expressions in the original cloning example are corresponding metric clones. Each expression in the application belongs to a certain corresponding metrics clones pattern, which is defined by the expression structure and a set of corresponding metrics in the expression. In case two expressions share the same expression structure and corresponding metrics, they are instances of the same corresponding metrics clones pattern. This classification groups expressions and rules possibly created by copy/paste operations: we group expressions together if we suspect that one of them can be created by copy/pasting and then modifying versions/roles/uoms in the other. By interacting with the users of the measure language we have learned that many times they actually use this kind of copy/paste methodology to create new rules and expressions, which is considered to be a bad practice. However, there are good reasons why the users go for it - many rules and expressions are quite similar. The two rules can be easily created from each other by copy/pasting and changing the versions and roles. The main cause of this is the way measure language is created: it doesn't take into account that there will be many expressions that share components. It can be noted that two expressions can have more similarities than just the corresponding metrics; they can have the same corresponding versions and/or roles/uoms. Expressions in Figure 4.3 also have the same corresponding UOMs. We do analysis for each of the combinations (of which there is $2^3 = 8$, the number of elements in the power set of {Version, Role, UOM}), but we report only the same metrics situation, since that is still a valid

corresponding metrics clone but it reports the biggest amount of cloning (which is expected since other variants are subsets of this one). 4.4 Importance Analysis Results We analyzed usages of the measure language to understand the importance and specifics of the two mentioned clone types. The results and the analysis of the results are given in the next two subsections. 28

```
TOTALSALES_TG_CAD =
WEBSALES_TG_CAD +
REGSALES_TG_CAD
TOTALSALES_FC_CAD =
WEBSALES_FC_CAD +
REGSALES_FC_CAD
```

Corresponding metrics clones: expressions have the same expression structure and corresponding metrics. Expression structure clones To determine the importance of this clone type we analyzed each application individually. We considered the main component of the importance to be the frequency of occurrence; we have also analyzed the nature of instances of this clone type to make sure they are not too simple and deserve further investigation. The frequency of occurrence of this clone type is defined as the percentage of expressions for which there exists at least one more expression with the same expression structure, (We have mentioned earlier that if two expressions have the same expression structure, they are considered to be expression structure clones.) This implies that in case an expression has a unique structure in the application, it will not be considered a clone. If we use the terminology, the frequency of occurrence is the percentage of expressions that are instances of those expression structure clone patterns that have at least two instances. We did an additional check to make sure that these instances deserve further investigation. For example, there can be 100 expressions in the application and 50 different expression structure clone patterns which are distributed such that each pattern has two instances. This situation would imply that there are many patterns but they are duplicated only once, which would make the process of improving such duplication harder. For our purposes, the situation where the number of patterns is small compared to the total number of expressions would be ideal. The results described in the next paragraph show that this is the case. The results for the described criteria for each of the three applications we analyzed can be

found. The first column in the table shows the total number of expressions in the application; the second column shows the frequency of occurrence and, as we can see, the substantial percentage of expressions have a non-unique expression structure; the last column shows the number of patterns with 2+ instances - the number of patterns is very small compared to the number of expressions, which means that many expressions

3. EXPRESSION LIST CREATION

Our proposed work is developed in VB.net and SQL. List of expression are created based on dependencies of data in the statement. For creating list of programs, we first have to store the variables along with data types. Next we scan the statements of the program to check for statements of data dependencies.

Fun 1()

```
{int Num1,
Num 2;
float
Num3,
Num4;
Num1 =
Num3, *
Num4;

Num2 = (Num2 + Num3)/Num4
;}
```

Figure 1: Assignment Statements.

int	=	int	+	float	/	Float
int	=	float	*			Float

Figure 2: Expression Lists

The lines which are identified are stored separately in a linked list. Now then variable names are replaced using the data types. The operations performed on the variable will remain same for all the operations. Let us assume the statement related to assignment in figure 1. The list for the expression is shown in figure 2. The

list for expression function is created and shown as.

Let us assume that function has 4 dependencies on data statements which are listed in expression lookup and created. The next step is to check the listed link function by another function. The if function posses the identical number in the list and if it is matched with all , then it is stated as copied 100%. Even the dependency of data statement are shuffled it do not affect the working of the method in the code cloned.

Data Dependency using expression List Method

Input: Function list of two programs

Output: Percentage of matching function expression

1. Repeat for each function (f1) in program1
2. Repeat for each list (l1) in function f1
3. Repeat for each expression (e1) in l1
 - a. repeat for each function (f2) in program2
 - b. repeat for each list (l2) in function f2
 - c. repeat for each expression (e2) in l2


```

          if (size (e1) = size(e2))
          if elements of
            e1 same as
            e2 count++
          
```
 - d. save the higher counter for each list
4. Return list of matched counter
5. End.

Figure 3. Algorithm of Expression List

4. DATA DEPENDENCY MATRIX CREATION

The matrix of data dependency algorithm verifies the similarities and size of matrix which is common. It is assigned that clone will change the position of function as well as variable name, but the used variable and function of them will be similar. Using our algorithm such type of clone code detection is done.

The algorithm compares the list expressions will all the functions in the lists with some other function. This process is advantages in our method and it gives good results in identification of logical clones too.

In cases like others, we can use another variable which don't affect the logic of the program but disguise the code clone. Likewise the dependent of data in the statement may also be done. Those problems will affect the working of the dependency data matrix algorithm. To solve this problem a framework approach of column and row shift is followed.

Another possible way of code coping is to shuffle the position of the variables. A code of the program may have many variables; each of these variable is easily copied or cloned by renaming the name of the variable at its location of instruction. It is the best way of hiding the code clone. Programs of such matrices will not be similar.

We use column shift and row shift approach in adjusting the dependences of the variables at left upper side of the matrix. The example of this is shown below in figure 5. The figure shows how column and row shifts are done for the given code figure 4.

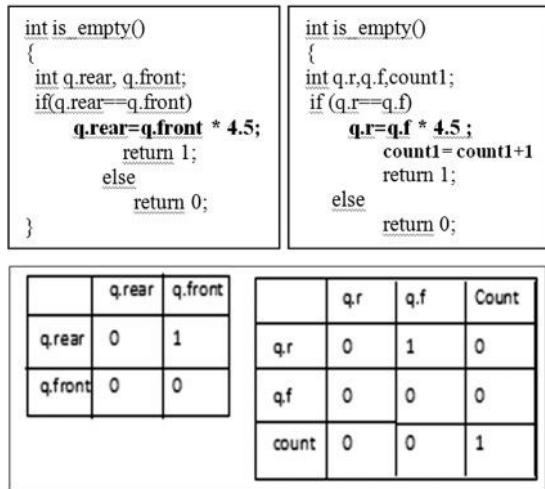


Figure 4: Same Statements But Changed Variables Name And Sequence

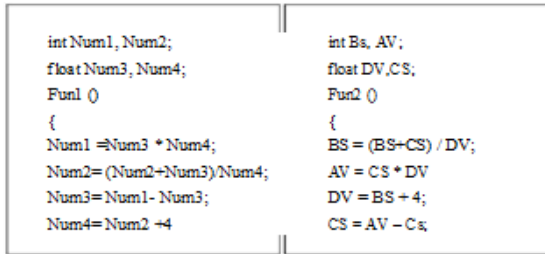


Figure 5: Column And Row Procedure Shifting

The example shows names of variable along with change of position without affecting the code performance. We show a matrix of data dependency of (First1 and First2) is read as variables in the program. The matrix of data dependency is given in figure 5A (1) and figure 5B(1) for first1 and first2. The first operation is related to shift row for fine tuning the larger variable dependencies on side upper of a matrix which is shown in figure 5a(2) and also in figure 5b(2).

The operation second is done based on shift column for altering the higher dependencies on hand left size which is shown in figure 5a(2) and figure 5b(3). Cloning will add more than one variable which may be unwanted. Dependency data matrix is a copy function which is a different from actual matrix function. The figure 6.

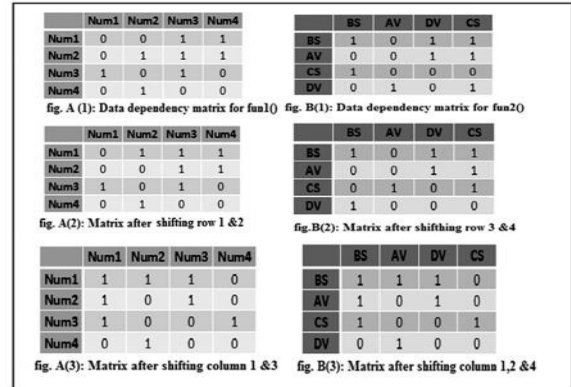


Figure 6: Code Of Modified Vs Code Of Original By Adding New Variable And Dependency Data Matrices Of Related Code. This Will Illustrate Matrices Of Similar Code With Different From Original Are Shown Below.

In some cases using a added variable will increase the size of the second matrix. As seen so the second code is a copy of the first code, but will not in a case to find the similarity detection in the matrix. Their need a modification in the algorithm, the modification that has to be done is a small size matrix is checked frame by frame with bigger matrix. If the matrix small is identified in matrix bigger it is considered and said as code cloned.

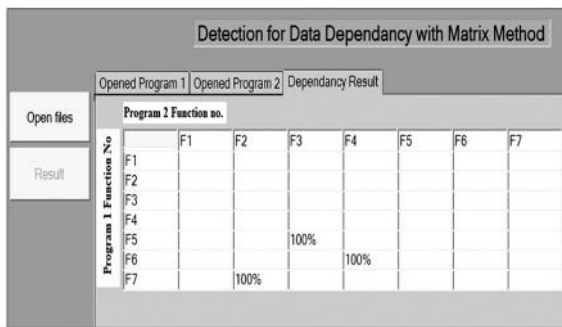
Likewise the clone may have combination of all various functions in major and smaller reduced functions to cove the program dependencies of matrix, the matrix gives different results. The result shows that it is falsely detection when compared to the identical size of matrix. This may be the drawback of the system.

Hence using an extra variable in the function of an expression list when matched, our algorithms give a result of more than 50 % matched with the first function. the result of comparison for each list matrix added for un necessary variable or shuffled variable in the statement will not affect the method which is proposed.

5. RESULTS

Novel framework approach is developed to identify the matrix data method and list expression method which is proposed and implemented.

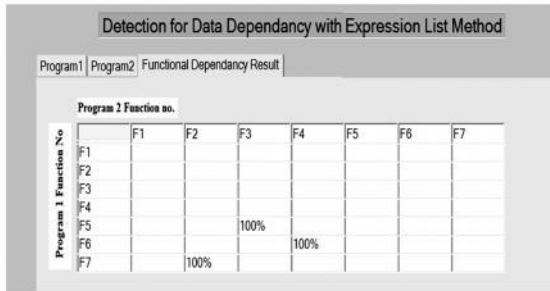
The proposed method is checked for C++ and C program code to identify the percentage of dependency related to clone. The results are shown in Figure 7 and figure 8. These figures show the comparison of two programs related to function. The result of first two programs show how to create double linked list and its input related. The program second is cloned which has a function shuffled positions. In program 3, functions with dependency statements of data clearly show that 100% code is cloned and detected based on all 3 functions.



		Program 2 Function no.						
Open files		F1	F2	F3	F4	F5	F6	F7
Program 1 Function No	F1							
	F2							
	F3							
	F4							
	F5			100%				
	F6				100%			
	F7		100%					

Figure 7: Method Uses Linked List Results By Matrix

We also used programs based on queues and compared with the operations, results in figure 9 and 10 shows the dependency for the 2 functions.

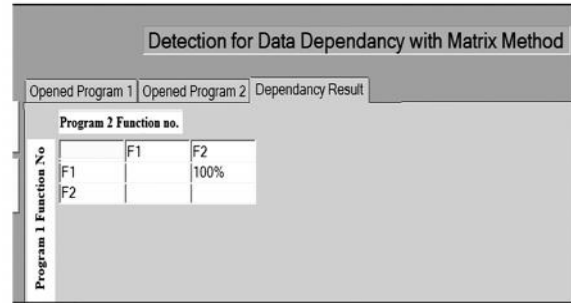


		Program 2 Function no.						
Program 1		F1	F2	F3	F4	F5	F6	F7
Program 1 Function No	F1							
	F2							
	F3							
	F4							
	F5			100%				
	F6				100%			
	F7		100%					

Figure 8: Shows Results Of Expression Method On Double Linked List Program

Using matrix and list expression method first1 of prg1 is nearly 50% similarity and dependence on 2 function of program 2. Hence we state that expression method is much effective than matrix method. We also can detect clones using clone identification program which contains a shuffle code in the program or function or by introducing a un necessary variable. The figure 10 shows the comparison

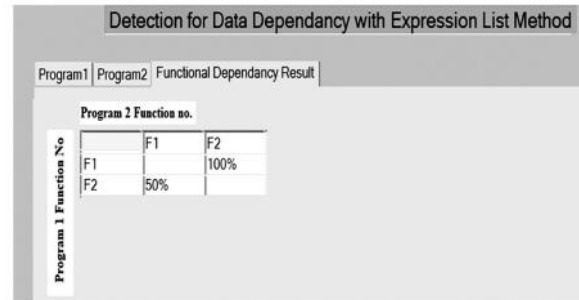
results of two programs related to queue and changed position of queue variable in function.



		Program 2 Function no.	
Opened Program 1		F1	F2
Program 1 Function No	F1		
	F2		100%

Figure 9: Circular Queue Program With Matrix Method

The result observed show that an academican or a manager can easily identify the code cloned using these cases of program or assignments done using programming language.



		Program 2 Function no.	
Program 1		F1	F2
Program 1 Function No	F1		100%
	F2	50%	

Figure 10: Circular Queue Program With Expression List Method

6. CONCLUSION

Our proposed work is used to develop an algorithm is based on list expression method and dependencies of data which solves the related problems of matrix method. This algorithm develops checks for clones in the program function by function. An experimental study is done and demonstrated for checking the effectiveness of the developed tool which is applicable for practice.

The proposed work is developed to check the code clone in the programs if the program has much dependencies on data.

REFERENCES

- [1] OkyayKaynak, Robin Braun, Ian Kennedy, "Guest Editorial Plagiarism", IEEE Transactions on Education, 51, No.2, May 2008, pp.149-150.
- [2] Georgina Cosma and Mike Joy, "Towards a Definition of Source-Code Plagiarism", IEEE Transactions on Education, 51, No.2, May 2008, pp. 195-200.
- [3] Francisco Rosales, Antonio García, Santiago Rodríguez, José L. Pedraza, Rafael Méndez, and Manuel M. Nieto, "Detection of Plagiarism in Programming Assignments", IEEE Transactions on Education, 51, No. 2, May 2008, pp.174-183.
- [4] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker, "Shared Information and Program Plagiarism Detection", IEEE Trans. Inf. Theory, 50, No. 7, pp. 1545-1551, Jul. 2004.
- [5] A. Parker and J. O. Hamblen, "Computer Algorithm for Plagiarism Detection", IEEE Trans. Educ., 32, No. 2, pp. 94-99, May 1989.
- [6] S. Schleimer, D. Wilkerson, and A. Aiken, "Winnowing: Local Algorithms for Document Fingerprinting", in Proc. 22nd Association for Computing Machinery Special Interest Group Management of Data Int. Conf., San Diego, CA, Jun. 2003, pp. 76-85.
- [7] Chao Liu, Chen Chen, Jiawei Han, Philip S. Yu, "GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis", KDD'06, Philadelphia, Pennsylvania, USA. August 20-23, 2006.
- [8] Michael Wolfe, "High Performance Compilers for Parallel Computing", Addition-Wesley, Redwood City, 1996.
- [9] P. Hudak. Modular domain specific languages and tools. In Proceedings of the 5th International Conference on Software Reuse, ICSR '2014, pages 134–, Washington, DC, USA, 1998. IEEE Computer Society.
- [10] J. Howard Johnson. Identifying redundancy in source code using fingerprints. In Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering - Volume 1, CASCON '93, pages 171–183. IBM Press, 2015.
- [11] Cory Kasper and Michael W. Godfrey. "cloning considered harmful" considered harmful. In Proceedings of the 13th Working Conference on Reverse Engineering, WCRE '06, pages 19–28, Washington, DC, USA, 2016. IEEE Computer Society.
- [12] Paul Laird and Stephen Barrett. Towards dynamic evolution of domain specific languages. In Proceedings of the Second international conference on Software Language Engineering, SLE'09, pages 144–153, Berlin, Heidelberg, 2015. Springer-Verlag.
- [13] Yu-Seung Ma and Duk-Kuyn Woo. Applying a code clone detection method to domain analysis of device drivers. In Proceedings of the 14th Asia-Pacific Software Engineering Conference, APSEC '07, pages 254–261, Washington, DC, USA, 2015. IEEE Computer Society.
- [14] Mika V. Mantylä and Casper Lassenius. Subjective evaluation of software evolvability using code smells: An empirical study. Empirical Softw. Engg., 11(3):395–431, September 2016.
- [15] Mark Marcel van Amstel, Marcel van den Brand and Luc Engelen. An exercise in iterative domain-specific language design. In Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), IWPSE-EVOL '10, pages 48–57, New York, NY, USA, 2014. ACM.
- [16] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. ACM Comput. Surv., 37(4):316–344, December 2015.
- [17] Akito Monden, Daikai Nakae, Toshihiro Kamiya, Shin-ichi Sato, and Ken-ichi Matsumoto. Software quality analysis by code clones in industrial legacy software. In Proceedings of the 8th International Symposium on Software Metrics, METRICS '02, pages 87–, Washington, DC, USA, 2013. IEEE Computer Society.