Functionally Equivalent Clone Detection Using IOT-Behavior Algorithm

Xia Li, Tiantian Wang, Xiaohong Su, Peijun Ma
School of Computer Science and Technology
Harbin Institute of Technology
Harbin, China
sxh@hit.edu.cn

***Abstract*—This paper presents an algorithm for the detection of functionally equivalent code clones in C code. The functionally equivalent code clones is the forth type of clones, which means that two or more code fragments that do the same calculation but with different syntax. Thus, we can detect the functionally equivalent clones by observing the input-output behavior. We propose the definition of input-output behavior by including not only the values of input-output, but also the number and types of input sets and output sets. We call this as the IOT (input, output and types)-Behavior of C code fragment. Our algorithm has been tested in open source code.**

Keywords- Clone Detection; Functionally Equivalent Clones; IOT-Behavior

I. INTRODUCTION

Clone code refers to the same or similar code fragment in source code file [1]. In the process of the software development, program developers often introduce clones due to code copy and paste, design ideas reuse, same or similar functions and coding habits. The research indicates that clone code accounted for about 7-23% [2] in large software systems. With the changing and increasing of demands, the size of software system is becoming larger, which leads to more code clones. The existence of code clone increases the length of the code in software system, making it more difficult for software maintenance and further evolution. For all these cases, it is necessary to detect code clone.

Roy[2] divides code clone into four types as follows based on the similarity of text or function, of which the top three types are code clones with the similarity of text, the fourth is the functional equivalent code clone.

Type 1: the totally same code fragments except the space, format and comments.

Type 2: the syntactically same code fragments except identifiers, constants and types.

Type 3: the modified code fragments after copying and pasting, such as change, add, or delete the statement in code fragments.

Type 4: two or more code fragments that do the same calculation but with different syntax.

So far, the study for the top three types is more, while research of the fourth type is less. Jiang and Su [3, 4] put forward the method for identifying functionally equivalent clone code fragments. This method combines the automatic random testing and static program analysis together to find the duplicated code fragments of functionally equivalence. The references define the concept of functionally equivalence, considering whether the program can obtain the same output when given the same input. It uses sliding window method extract candidate code fragments, but it may produce a lot of meaningless code fragment pieces.

Reference [5] adopts the K-nearest neighbor algorithm to extract candidate code fragments with lower time complexity and getting higher cohesive and more meaningful code fragments. But it doesn't take the parameter information of code fragments into account, while different parameters can lead to the inequivalence of functions. According to the reference [5] without considering the number and type of parameter of candidate code fragment, we put forward the IOT-Behavior algorithm to extract the candidate code clone fragment, which can reduce the time complexity of dynamic testing. First of all, this paper adopts the K-nearest neighbor algorithm to extract the higher cohesive code fragments. Then it classifies the code fragments into the preliminary sets via IOT information. Last, it groups the functionally equivalent code fragments using the approach combining automatically random assignment and dynamic testing. The experimental results showed that the approach can detect functionally equivalent code fragments accurately and effectively with a lower time complexity.

II. RELATED WORK

In the detection of code clone, a lot of methods have been put forward, which are token-based, tree-based and PDG-based methods.

Toshihiro Kamiya [6] proposed the detection method based on the token which includes a series of conversion and token-by-token comparison of the source files. They optimize the method and develop a tool called CCFinder. The tool can detect the clone of C,C++,Java source code.

Ira D.Baxter [7] used the tree-based algorithm to put forward a simple and practical method. This method can produce macro for every clone which has been detected, and use macro call to replace the original clone code.

In recent years, the code clone detection method based on similarity subgraph was studied. First the source code is expressed as PDG, then similar code is identified according to the PDG isomorphism. The similarity subgraph-based method not only consider the grammar of the program structure but also take the data flow information into

account, laid a theoretical foundation for the semantic-level similar code detection. However there are still two key problems didn't get good solution. One is limit of recognition ability of code diversity, especially it can only detect a single function, and it has no effect on the detection of code which uses different module structure to realize similar function. Another is high computational complexity. Reference [8] show the grammar of the program and the semantic structure by using the improved program dependence graphs and do the standardization to the program dependence graph.

However, the detection for functionally equivalent code clone is less. Reference [3,4] proposed the first scalable algorithm that can mine functionally equivalent code fragments, in this method, input/output is the main analysis object. The benefit of this approach is that it focuses on a piece of code external observable behavior, and it is not sensitive to code conversion or internal different realization. This approach provides a new train of thought for detecting functionally equivalent code clone. Reference [5] proposed a novel method to detect functionally equivalent code fragments via K-nearest neighbor algorithm. This method is the combination of abstract syntax tree, control dependence graph, K-nearest neighbor clustering algorithm, automatically random assignment and dynamic testing. It can achieve the purpose of detecting, but it didn't take the static analysis into account, which can effectively reduce the time complexity.

Due to the shortcomings of the above methods, this paper proposes a new method to detect functionally equivalent code fragments via the IOT-Behavior with a lower time complexity.

III. The Approach of Functionally Equivalent Clone Detection Using IOT-Behavior Algorithm

A. *The model to detect functionally equivalent clone using IOT-Behavior algorithm*

Our approach works at the method level in C code. In this paper, we discuss how to detect the function equivalence code clone with a lower time complexity. The functionally equivalent code detection involves 4 sub-processes: Abstraction, filtering, testing and collection, as shown in Figure 1 and discussed in the sections that follow.

The functionally equivalent clone detection using IOT-Behavior algorithm can be described as follow:

1. Class the methods that are clustered by the k-nearest neighbor clustering method.

2. Class methods that have the same number of parameters. Discard all classes containing only one method.

3. For each class of methods left after step 2, class methods that have the same number of input parameters and the same number of output parameters. Discard all classes that have only one method.

4. For each class of methods left after step 3, class methods that have the same type of parameters. Discard all classes existing only one method.

5. For each class remaining after step 4, for each method f in the class:

a. generate the random number sequence based on the parameter list and then the sequence needs different permutation.

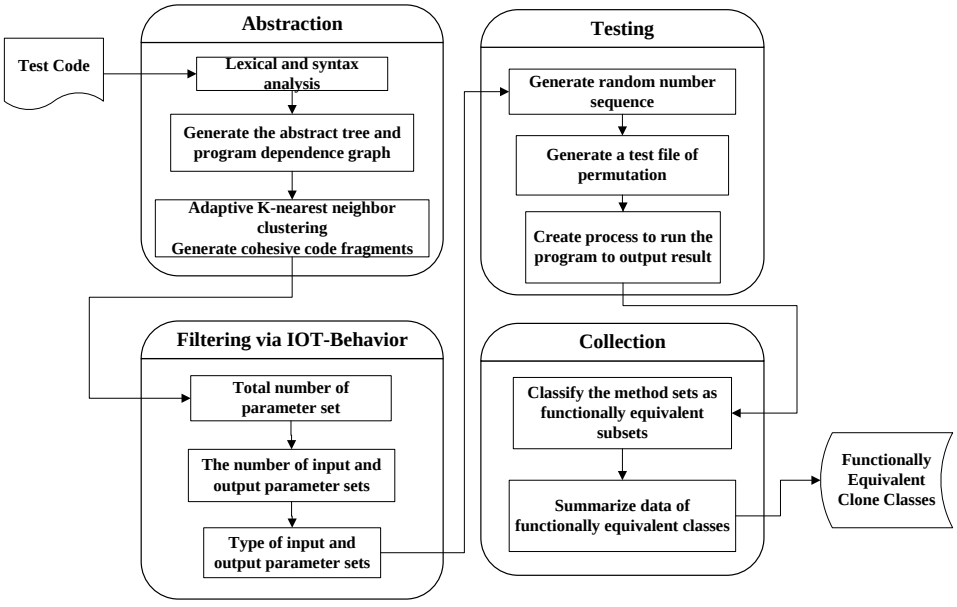

Figure 1. The model to detect the functionally equivalent code clone via the IOT-Behavior algorithm

b. call f using the random number sequence created in step a, record the value of outputs. Store the results in result vector.

c. repeat the step a and step b until the all methods in the class are called to output results.

d. push method f in the functionally equivalent class vector according to the result stored in the result vector. if the method f has the same result with the the other methods, then they will be divided into one class of functionally equivalence.

6. Discard the functionally equivalent classes which have only one method. Output the method information of functionally equivalent classes.

B. Abstraction

The first step uses the static analysis tool which is achieved by the defining parsing rules to produce the abstract syntax tree(AST) and the program dependence graph(PDG). AST and PDG can provide the information of data dependency and control dependency attributes. Then the K-nearest neighbor clustering method is used to create a matrix according to the attributes and the analyzed program is divided into several cohesive code fragments via the matrix. Table I demonstrates the cohesive code fragments of a piece of code via the K-nearest neighbor clustering method. The parameter marked with & represents output parameter.

C. Filtering

As shown in the step 2 of Figure 1, the filtering phase uses the code abstractions to identify preliminary sets of potential method clones, what are called the candidate clone sets. Two filters are used in this step. One is the number of input and output parameters, another is the type of parameters. The filtering section is divided into three steps by observing the two filters. The procedure of filtering is named IOT-Behavior Algorithm.

First, the total number of parameters are calculated. The methods with same number of parameters will be grouped as the same candidate clone sets. For example, as shown the methods in Table I, applying the total number of parameters would get 2 candidate clone sets – {name_1, name_2, name_3, name_4} and {name_5, name_6}. The seventh method – name_7 would be filtered out from the candidate clone sets, because it has only one parameter, detailed information is shown in Table II.

TABLE I. SAMPLE METHODS FOR TYPE AND NUMBER OF PARAMETER ANALYSIS

Method	parameter
name_1	int *& output,int * input
name_2	int *& out,int * in
name_3	int& n,int & a[]
name_4	int & a[],int& n
name_5	int& i,int& j,char& temp,char & a[],int n
name_6	int& i, int& j,int& temp, int & a[],int n
name_7	int &a

TABLE II. SAMPLE METHODS FOR TOTAL NUMBER OF PARAMETER ANALYSIS

Method	Total number of parameter
name_1	2
name_2	2
name_3	2
name_4	2
name_5	5
name_6	5
name_7	1

Second, the number of input and output parameters are taken into account separately. The 2 candidate clone sets returned by the first step of filtering are further divided into 3 sets. From the data shown in Table III for example, the second step of filtering would result in such candidate clone stes - {name_1, name_2}, {name_3, name_4} and {name_5, name_6}.

Last, the second filter is used. Using the type properties, the candidate clones are further reduced by comparing the type of parameters. The final candidate method clones are formed as new sets – {name_1, name_ 2} and {name_ 3, name_ 4}. The second candidate clone sets {name_5, name_6} returned by the second step will not be considered since the name_5 and name_6 do not have the same parameter type. We can see the type information in Table IV.

The net result of the two filters, which the methods regarded as final candidate clones, would have the same parameter list, including the number and type. The reduction in size of equivalence class is important, because it can reduce the number of methods to do dynamic testing at next level of clone detection, which has the most time complexity.

TABLE III. SAMPLE METHODS FOR NUMBER OF INPUT AND OUTPUT PARAMETER ANALYSIS

Method	Number of input parameter	Number of output parameter
name_1	1	1
name_2	1	1
name_3	0	2
name_4	0	2
name_5	1	4
name_6	1	4

TABLE IV. SAMPLE METHODS FOR TYPE OF INPUT AND OUTPUT PARAMETER ANALYSIS

Method	Input Parameter		Output Parameter	
	Type	Number	Type	Number
name_1	int	1	int	1
name_2	int	1	int	1
name_3			int int[]	1 1
name_4			int int[]	1 1
name_5	int	1	int char[] char	2 1 1
name_6	int	1	int int[]	3 1

D. Testing

If two methods are functionally equivalent, the outputs would be identical when given the same inputs. A random number sequence can be generated for each class. A test file of permutation of the number sequence can also be generated for each class. The data in test file will be used as the input to run the methods that in final candidate clones.

E. Collection

According to the test file, the processes are created to run the program and every process can generate one output list. The methods are classified as different equivalence subsets according to the output. If output of two methods is same, then the two methods will be classified as one class. The collection step also outputs a statistical summary of the clone data including: location, source code and size of the clones and clone classes. All data will be collected and stored in the clone collection.

F. IOT-Behavior Algorithm

The process of IOT-Behavior algorithm is shown in Figure 2.If we set the number of parameters in one method as n, time complexity of direct dynamic testing is O(n!), while time complexity of IOT-Behavior algorithm is O(1), then time complexity of combining dynamic testing and IOT-Behavior algorithm is $\alpha O(1)+\beta O(n!)$, $0 \leqslant \alpha \leqslant 1$, $0 \leqslant \beta \leqslant 1$, $\alpha + \beta = 1$, α is the probability of using IOT-Behavior to be discarded but not using dynamic testing, β is the probability of using dynamic testing. We know that $\alpha O(1)+\beta O(n!) \leqslant O(n!)$, which shows that using IOT-Behavior algorithm can reduce the time complexity.

IV. Experimental Results and Analysis

The aim of proposing the IOT-Behavior algorithm is to reduce the time complexity of reference [5]. We choose open source program Simens to do experiment, which is also selected as the experimental data in reference [5]. There are 7 files in Simens: Print_tokens, Print_tokens2, Replace, Schedule, Schedule2, Tcas and Tot_info. We recorded the number of candidate clones generated by using IOT-Behavior algorithm and not using IOT-Behavior algorithm. The statistics information is demonstrated in Table V. In Table V, #N_IOT means the number of candidate clones produced by using the approach in reference [5]; #IOT expresses the number of methods in final candidate clone sets generated by using IOT-Behavior algorithm; #Rr is calculated as the formula: #Rr = (N_IOT－IOT)÷N_IOT, implying the reduction ratio.

```
Algorithm IOT-Behavior(F,R)
1:Input:F: the set of method needed to be filtered
2:Output: the result of filtering R
3:R←Ø
4:for fi = F.first to F.end
5:   if R≠Ø
6:       for Rj = R.first to R.end
7:           if number_parameter(fi) = number_parameter(Rj)
8:               Rj.pushback(fi)
9:   else R1.pushback(fi)
10:for Rj = R.first to R.end
11:     if Rj.size = 1
12:         R.pop(Rj), F.pop(Rj)
13: R←Ø
14:for fi = F.first to F.end
15:     if R≠Ø
16:         for Rj = R.first to R.end
17:             if number_in(fi) = number_in(Rj) && number_out(fi) = number_out(fi)
18:                 Rj.pushback(fi)
19:     else R1.pushback(fi)
20: for Rj = R.first to R.end
21:     if Rj.size = 1
22:         R.pop(Rj), F.pop(Rj)
23: R←Ø
24: for fi = F.first to F.end
25:     if R≠Ø
26:         for Rj = R.first to R.end
27:             if type(fi) = type(Rj)
28:                 Rj.pushback(fi)
29:     else R1.pushback(fi)
30: for Rj = R.first to R.end
31:      if Rj.size = 1
32:          R.pop(Rj)
33:return R
```

Figure 2. IOT-Behavior algorithm

In addition to Simens, we tested our algorithm on UCC source code. The results are shown in Table VI.

After manual analysis, we found that the reduction part of candidate clones are not equivalent to the reserved clones, which suggests that the IOT-Behavior algorithm don't affect the accuracy and correctness of the clone detection. Compared with method in [5], this method can obtain fewer number of the final functionally equivalent candidate clones. The methods filtered from the initial candidate clone sets do not influence the detection. So this algorithm proposed in this paper can detect functionally equivalent code fragments with a lower time complexity.

TABLE V. RESULT FROM RUNNING IOT-BEHAVIOR AND NO IOT-BAHAVIOR ON SIMENS

Program	Description	LOC	#N_IOT	#IOT	#Rr
Print_tokens	Lexical analyzer	565	39	31	20.51%
Print_tokens2	Lexical analyzer	508	98	86	12.24%
Replace	Pattern replacement	563	55	39	29.09%
Schedule	Priority scheduler	368	41	32	21.95%
Schedule2	Priority scheduler	307	47	42	10.64%
Tcas	Altitude separation	173	13	8	38.46%
Tot_info	Information measure	406	28	16	42.86%

TABLE VI. AVERAGE RESULT FROM RUNNING IOT-BEHAVIOR ALGORITHM AND NO IOT-BAHAVIOR ALGORITHM ON OPEN SOURCE PROGRAM

Program	LOC	#NIOT	#IOT	#Rr
Simens	2890	321	254	20.87%
UCC	3140	201	133	33.83%

V. Conclusion

An improved algorithm for detecting functionally equivalent code fragments is proposed in this paper. An objective definition of functionally equivalent clones using the IOT-Behavior of code is presented. Compared with [5], this proposed algorithm can get functional equivalent code fragments with a lower time complexity, and without decreasing the accuracy and effectiveness in analyzing the program.

ACKNOWLEDGMENT

This research is supported by the National Natural Science Foundation of China (Grant No. 61173021) and the Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20112302120052 and 20092302110040).

REFERENCES

[1] Toshihiro Kam iya, Shinji Kusumoto, Katsuro Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code [J]. IEEE Transactions on Softw are Engineering, 2002, 28 (7) : 6542670.

[2] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach[J]. Science of Computer Programming. 2009, 74: 470-495.

[3] Lingxiao Jiang. Scalable Detection of Similar Code[D]. Techniques and Applications. Computer. 2009:82-119.

[4] Lingxiao Jiang, Zhendong Su. Automatic Mining of Functionally Equivalent Code Fragments via Random Testing[C]. ISSTA '09 Proceedings of the eighteenth international symposium on Software testing and analysis. 2009:81-91.

[5] Dandan Kong, Xiaohong Su, Shitang Wu, Tiantian Wang, Peijun Ma. Detect Functionally Equivalent Code Fragments via K-nearest Neighbor Algorithm. Fifth International Conference on Advanced Computational Intelligence (ICACI2012). 2012:94-98.

[6] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue. CCFinder:a multilinguistic token-based code clone detection system for large scale source code[J]. IEEE Transactions on Software Engineering, 2002, 28(7): 654-670.

[7] Ira D. Baxter, Andrew Yahin, Leonardo Moura, et al. Clone detection using abstract syntax trees[A].In:Proceedings of the International Conference on Software Maintenance[C]. Maryland, USA: IEEE CS, 1998. 368-377.

[8] Tiantian Wang. The study to identify structure semantics similar programs. Harbin Institute of Technology Doctoral Dissertation. 2009