Code Clone Detection using Graphs and Adjacency Structures

Mukesh Kumar[a], Lalit Kumar Sagar[b], Saurabh Kumar[c]
[a, b, c] *Assistant Professor, Dr. K.N.M.I.E.T, India*

***Abstract*: - Code clone detection is the common aspect of reuse activity. Copying code fragments and then reuse with or without modifications are common activities in software development. This type of reuse approach of existing code is called code cloning and the pasted code fragment is called a clone of the original [1].Existing approaches does not make use of adjacency structures and their properties. In this paper we present an efficient way of finding code clones by using adjacency structure. We developed a directed graph of the source code and parsed the information into adjacency structure. By using the properties of adjacency structure we can find the in-degree and out-degree of a particular directed graph. Our insight is to deduce the flow on a particular node of the directed graphs to find the similarity between the nodes of the directed graph. We implemented this algorithm practically by using the tool named as control flow graph factory.**

***Keywords:* Control flow graphs; Adjacency structure; In-degree; Out-degree.**

I. INTRODUCTION

1.1 Code Cloning

Code duplication or copying a code fragment and then reuse by pasting with or without any modifications is a well known code smell in software maintenance. Several studies show that about 5% to 20% of software systems can contain duplicated code, which are basically the results of copying existing code fragments and using then by pasting with or without minor modifications [1].

A code fragment CF1, which is a sequence of code line is clone to another code fragment CF2, if they have similar properties i.e. F (CF1) = F(CF2), where "F" is a similar function .Two fragments that have similar properties are referred as clone pair (CF1, CF2) and when many fragment are similar then they form clone class or clone group [2].

1.2 Type of clones:-

There are two main kinds of similarity between code fragments. Fragments can be similar based on the similarity of their textual similarity [3] [4] [5], or they can be similar based on their functionality (independent of their text) [6] [7] [8] [9]. The first kind of clone is often the result of copying a code fragment and pasting into another location. In the following we provide the types of clones based on both the textual (Types 1 to 3) [10], and functional (Type 4**) [**11**]** similarities.

Type-1: Identical code fragments except for variations in whitespace, layout and comments.

Type-2: Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments.

Type-3: Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments.

Type-4: Two or more code fragments that perform the same computation but are implemented by different syntactic variants.

II. AUTHOR ARTWORK

A tool "Control Flow Graph Factory" is used to generate control flow graphs. This tool generates different types of graphs like Byte code graphs, Basic Block Graph. A basic introduction of Control Flow Graph Factory is given below.

2.1 Control Flow Graph Factory

Control Flow Graph Factory is an Eclipse plug-in which generates control flow graphs from java code, edit them and export to GraphXML, DOT or several image formats.

- **Features**
  - Automatic generation of several types of control flow graphs from Java byte code like:
    - Byte Code Graphs
    - Source Code Graphs
    - Basic Block Graphs
- Editing of control flow graphs
  - Move, create, delete, rename, ... nodes
- Multiple algorithms for automatic layout (serial, hierarchical)

- Export in GraphXML, DOT format or as an image (JPEG, BMP, ICO, PNG)
- Printing support

There are several steps to generate a control flow graph. These steps are explained below.

1. To generate the graph for the method “main” select the method in “package explorer” and open the context menu "Create Control Flow Graph". Select submenu "Source code graph" to generate a source code graph for this method.

2. Generate a byte code or a basic code graph in the same way. For that use the context menu in the package explorer "Create Control Flow Graph/Byte code graph" or "Create Control Flow Graph/Basic block graph.

3. For export the graph in DOT, GraphXML format or to an image use the export functions provided by the Control flow graph Factory. For finding the geometry information of the graphs (may be basic block, source code, byte code) export the graph by export geometric info. This geometry information, gives the information about all the vertices and edges which are connected with each other. For example:- Take the java code that print hello using “While” loop.

```
Package test1;
Public class tes1 {
Public static void main(String[] args) {
       int i=0;
        while(i<10)
        {
        System.out.print ("hello");
        }}}
```

Take a second java code that print hello using “For” loop.

```
Package test2;
Public class tse2 {
Public static void main (String [] args)
{
int i=0;
   for (i=0;i<10; i++)
{
     System.out.print ("hello");
     }}}
```

The Basic block Graph of “While” loop code is:-

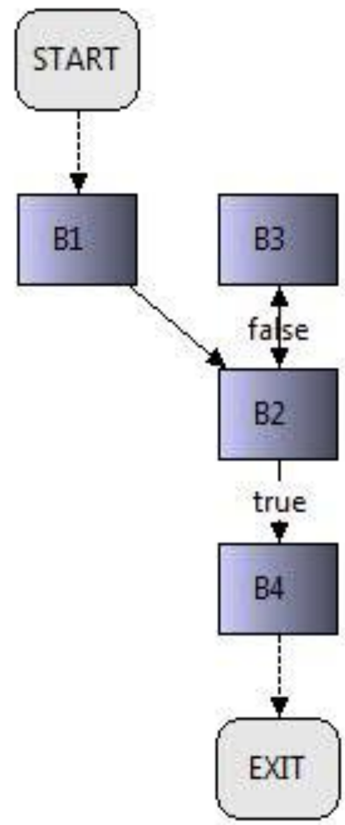

Fig.1 Basic Block Graph

The exported information of the graph is:-

2 [label="B1"]

3 [label="B2"]

4 [label="B3"]

5 [label="B4"]

6 [label="EXIT"]

7 [label="START"]

7 -> 2 [label=""]

2 -> 3 [label=""]

4 -> 3 [label=""]

3 -> 4 [label="false"]

3 -> 5 [label="true"]

5 -> 6 [label=""]

Similarly the Basic Block Graph of “For” loop is:-

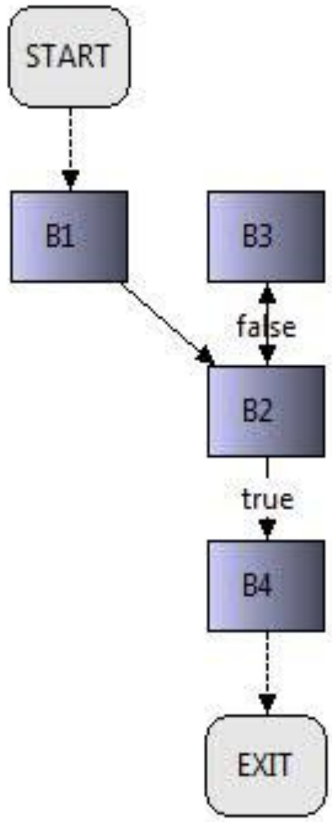

Fig.2 Basic Block Graph

The adjacency matrix for the exported information is in the below matrix (Fig. 3).

ROWS

COLUMNS

	2	3	4	5	6	7
2	0	1	0	0	0	0
3	0	0	1	1	0	0
4	0	1	0	0	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	0
7	1	0	0	0	0	0

Fig.3. Adjacency Matrix.

This means 6 nodes are presented in the graph and

- Node 7 is connected with node 2.
- Node 2 is connected with node 3.
- Node 4 is connected with node 3.
- Node 3 is connected with node 4.
- Node 3 is connected with node 5.
- Node 5 is connected with node 6.

So the exported information is used in the adjacency matrix to find the" out-degree" of a graph.

- An adjacency matrix is created on the basis of exported information.
- Take the transpose of adjacency matrix.
- Multiplication of adjacency matrix with its transpose provides the information about its out-degree and of the directed graph.

Again the exported information is used in the adjacency matrix to find the "in-degree" of a graph.

- An adjacency matrix is created on the basis of exported information.
- Take transpose of adjacency matrix.
- But at that time we multiply transpose of matrix and adjacency matrix. The multiplication of transpose of matrix and adjacency matrix gives in-degree of the graph.

The above process gives the "in-degree" and "out-degree" of a graph. "In-degree" and "out-degree" of two graphs is compared according to "in-degree" of one graph with "in-degree" of second graph, and similarly for "out-degree".

The above algorithm gives the information about which nodes are similar in two graphs, and how the information flows from one node to another node, and how many "in-degree" and "out-degree" each node have. Then comparison of these two graphs is done on the basis of their in-degree and out-degree.

Here is an example to understand the following procedure:-

Example 1 includes an algorithm to find the out-degree of a graph using following steps:

1. Take a directed graph with 3 nodes.
2. Draw an adjacency matrix for that graph.
3. Take the transpose of the adjacency matrix which is obtained from the graph.
4. Multiply adjacency matrix and transpose of adjacency matrix. ($A.A^T$).
5. Find a new matrix, the new matrix diagonal element gives the out-degree of that graph.

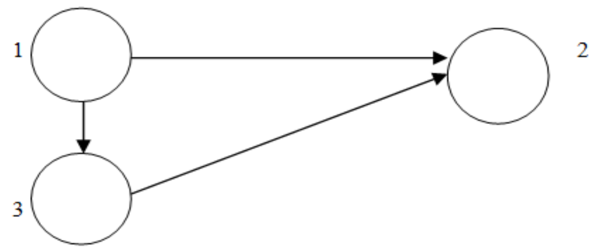

Fig.4 Directed graph

	1	2	3
1	0	1	1
2	0	0	0
3	0	1	0

Fig.5 Adjacency Matrix

	1	2	3
1	0	0	0
2	1	0	1
3	1	0	0

Fig.6 Transpose of adjacency matrix

	1	2	3
1	2	0	1
2	0	0	0
3	1	0	1

Fig.7 ($A.A^T$)

Example 2 uses an algorithm to find the in-degree of a graph:-

1. Take a directed graph with 3 nodes.
2. Draw an adjacency matrix for that graph.
3. Take the transpose of the matrix which is obtained from the graph.
4. Multiply transpose of adjacency matrix and adjacency matrix. ($A^T.A$).
5. Find a new matrix, the new matrix diagonal element gives the in-degree of that graph.

When in-degree and out-degree of a graph is available, compare it with two graphs.

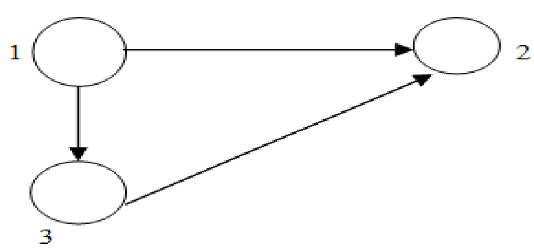

Fig.8 Directed Graph

	1	2	3
1	0	1	1
2	0	0	0
3	0	1	0

Fig.9 Adjacency Matrix

	1	2	3
1	0	0	0
2	1	0	1
3	1	0	0

Fig.10 Transpose of Adjacency matrix

	1	2	3
1	0	0	0
2	0	2	1
3	0	1	1

Fig.11 ($A^T.A$)

The above algorithm finds in-degree and out-degree of any graph, so based on in-degree and out-degree algorithm can compares two graphs, and find out which node is similar to each other.

This is the mathematical approach to find out the clones with similar nodes from a source code.

<u>Psuedo code</u>

Input :- 2d array of strings

output :- matrix which contains the outdegree in 1^{st} row and indegree in 2^{nd} row of a graph.

```
for aa←1 to count_of_lines do
 If text[aa].contains("->")
  adj[0] = Source_of_edge
  adj[1] = target_of_edge
  l++
 end if
end for
//Calculating minimum and maximum
for aa←1 to l
 for j←0 to 1
  if adj[aa][j] > max
   max = adj[aa][i]
  endif
  if (adj[aa][j] < min)
```

```
            min = adj[j][k]
          endif
        endfor
      endfor
   //Forming Adjacency
      for j←i-1 to 0
        adjacency[adj[j][0] - min][adj[j][1] - min] = 1
      endfor
 //Forming transpose
      for j←0 to max-min
        for k←0 to max-min
          if adjacency[j][k]==1
            adjacencyT[k][j] =1
          endif
        end for
      endfor
 //Mutiplying Adjacency and its transpose for outdegree
      for i1←0 to max-min
        for j←0 to max-min
          for k←0 to max-min
            outdegree[i1][j]  += adjacency[i1][k]
*adjacencyT[k][j]
          endfor
        endfor
      endfor
//Mutiplying transpose and Adjacency for indegree
      for i1←0 to max-min
        for j←0 to max-min
          for k←0 to max-min
            indegree[i1][j]  += adjacencyT[i1][k]
*adjacency[k][j]
          end for
        end for
   end for
// forming a club-up matrix which contains the outdegree in 1st row and indegree in 2nd row

for i1← 0 to max - min
        clubup[0][i1] = outdegree[i1][i1]
        clubup[1][i1] = indegree[i1][i1]
endfor
```

REFERENCES

[1]. C.K. Roy, J.R. Cordy, "A survey on software clone detection research", Technical Report: 541, 2007, p. 115
[2]. C.k.Roy, J. Cordy, and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools": A qualitative Approach. Sci. Comp. Prog., 74(7) (2009), 470–495.
[3]. Stefan Bellon. Detection of Software Clones Tool Comparison Experiment. Tool Comparison Experiment presented at the 1st IEEE International Workshop on Source Code Analysis and Manipulation, Montreal, Canada, October 2002.
[4]. Stefan Bellon.Vergleich von techniken zur erkennung duplizierten quellcodes. Diploma Thesis, No. 1998, University of Stuttgart (Germany), Institute for Software Technology, September 2002.
[5]. Rainer Koschke, Raimar Falke and Pierre Frenzel. Clone Detection Using Abstract Syntax Suffix Trees. In Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06), pp. 253-262, Benevento, Italy, October 2006.
[6]. Raghavan Komondoor and Susan Horwitz. Effective, Automatic Procedure Extraction. In Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC'03), pp. 33-42, Portland, Oregon, USA, May 2003.
[7]. Jens Krinke. Identifying Similar Code with Program Dependence Graphs. In Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01), pp. 301-309, Stuttgart, Germany, October 2001.
[8]. Neil Davey, Paul Barson, Simon Field, Ray J Frank. The Development of a Software Clone Detector. International Journal of Applied Software Technology, Vol. 1(3/4):219- 236, 1995
[9]. Gilad Mishne and Maarten de Rijke. Source Code Retrieval Using Conceptual Similarity. In Proceeding of the 2004 Conference on Computer Assisted Information Retrieval (RIAO'04), pp. 539-554, Avignon (Vaucluse), France, April 2004.
[10]. S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, "Comparison and evaluation of clone detection tools" IEEE Transactions on Software Engineering 33 (9) (2007) 577_591.
[11]. M. Gabel, L. Jiang, Z. Su, "Scalable detection of semantic clones", in: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008, 2008, pp. 321_330.