Enhanced Code Clone Detection and Analysis using Neural Network Algorithm

Jagjit Singh[1], Sukhpreet Kaur[2]
[1]*M.Tech (Scholar),* [2]*Assistant Professor*
Department of computer science & engineering, Sri Guru Granth Sahib World University, Fatehgarh Sahib, Punjab

Abstract - Duplicated code proves easy and cheap during the software development phase, but it makes software maintenance much harder. Software clone has a number of negative effects on the quality of the software. So there is a need to detect the clones to figure out the problems and to help better software understandability and maintenance. This paper propose a hybrid method that combines neural network with metric based method to yield structurally meaningful near-miss clones and implemented using MATLAB. It is a new clone detection method that has been shown to yield get high precision and high recall in detecting near-miss intentional clones.

Keywords - Code Cloning, Similarity, Neural Network, and Metric based Functions.

I. INTRODUCTION

Reusing of code with minor changes is common thing in today's era in software Development Company [1, 2, 3]. As a result a software consists of various fragments that are very similar to each other. From previous results it has been shown that from 7% to 20% it has been seen that code cloning has been done on the code. Code cloning is always intentional and there are numerous ways of doing code cloning [4, 5]. Code cloning also leads to difficulty in code maintenance. Duplicate code also leads to complexity when some enhancement or modification is going to be done [6]. Code detection is very important in software industry due to following reasons:

- Plagiarism detection
- Code mining
- Copyright Protection and ;
- Code Compaction

Over the last years many techniques has been recommended for code cloning [7, 8, 9]. In this paper, code cloning optimization will be done using neural network algorithm in addition with metrics based technique to enhance the accuracy of code cloning system. This algorithm will find out various types of code like type-1, type-2 etc. [10]. The remainder of the paper is organized as Section 2, 3 will discuss the proposed techniques basic concept. Section 4 will discuss the proposed work methodology. Section 5 contains the results and analysis. Finally section 6 contains the conclusion.

II. METRIC BASED TECHNIQUE

In metric based method, various metrics are used to find code clones to find the actual quantity of clones [11]. These metrics are related to each other on the basis of code class, function, method etc. The source code is mainly parsed into tree to get the maximum number of software metrics. There are several metrics that has been used to analyze code cloning. Various available metrics are Lines of Code (LOC) metric, CBO (Coupling between Object classes). In addition to this there are more metrics that has can be used for code cloning as shown below:

- ***LOC*** *(Lines of Code) calculates the lines of code of a specified unit.*
- ***NOM*** *(Number of Methods) calculates the methods in a class.*
- ***LCOM-CK*** *(Lack of Cohesion of Methods) describes the lack of cohesion between the methods of a class.*
- ***CBO*** *(Coupling between Object classes) gives the number of classes to which a class is coupled.*
- ***NOC*** *(Number of Children) is the number of subclasses to a certain class in its block.*
- ***RFC*** *(Response for a Class) reflects the number of methods which can executed in response to an object of the class.*
- ***DIT*** *(Depth of Inheritance Tree) represents the maximum inheritance path from the class to the main root class.*
- ***WMC*** *(Weighted Methods per Class) it is the total of weights for the methods of a class.*
- ***LCOM-HS*** *(Lack of Cohesion of Methods, proposed by Henderson-Sellers) describes the lack of cohesion*

A typical neural network consists of various number of neurons called units that are arranges in form of layers and each of which is connected to next layer via layers. Neural network mainly used for training. Usually, BPNN is used for solving many problems by using the simple output elements [13]. It is the mostly used learning algorithm in the neural network. BPNN is used with fuzzy encoder for understanding the human like reasoning activities of the fuzzy logic system.
BPNN consists of three layers that are: Input layer, Hidden layer and the output layer. The basic use of training the BPNN is for adjusting the weights among the layers for producing the expected output. The activation function of the hidden and the output layer with the sigmoid function and is given by:

$$f(x) = \frac{1}{1 + e^{-x}}$$

The value ranges from 0 to 1 for every unit range.

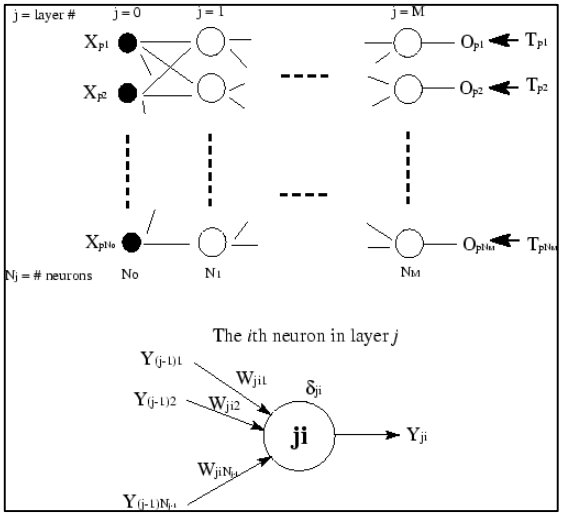

Fig.1: Neural Network Working Model

IV. CLONE DETECTION

Cloning mainly occurs because programmers find that it is cheaper and quicker to use the copy and paste feature than writing the code from scratch [14,15]. Sometimes programmers intent on implementing new functionality find some working code that performs a computation nearly identical to the one desired copy it entirely and then modify in place. Thus it is very important to understand the meaning of code cloning and various terms related to it [16].

4.1 Clone Relation Terms

Clone is mainly find out from main class or clone class. These mainly focus on similarity between two classes and their relation can be described on the basis of relations (i.e., a reflexive, transitive, and symmetric relation).

- **Clone Pair:** Two fragments are considered to be clone pair if two classes have some same properties. E.g. there are two code fragments ; fragment 1 & fragment 2so now it can be represented as:

$(G1(a), G2(a)), (G1(b), G2(b))$

If we assume to extend the granularity size of cloned fragments, we get basically two clone pairs,

$$(G1(a + b), G2(a + b))$$

And if we consider the granularity not to

$(G1(a), G2(a)), (G1(b), G2(b)),\ (G1(a + b), G2(a + b))$;

Each of these fragments is termed as a simple clone.

- **Clone Class:** It is the maximum of sets that contains similar data in same class. We get a clone class of $(G1(b), G2(b))$ where the three code portions $G1(b), G2(b)$ form clone pairs with each other $(G1(b), G2(b)), (G2(b), G3(a))$ and $(F1(b), F3(a))$ res ult in three clone pairs.
- **Clone Communities:** it is termed as clone communities that have maximum aggregation of similar data.
- **Clone Class Family:** It is the class of clones that have similar data domain.

V. PROPOSED WORK

5.1 Implemented Metrics

Below Figure shows some of these metrics that has been proposed in our work.

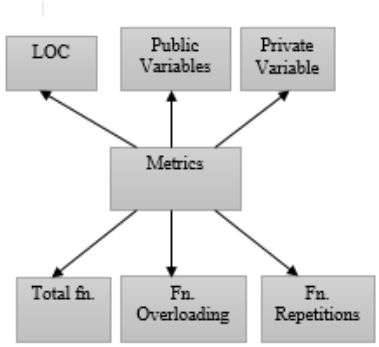

Fig.2: Various proposed metrics

Clone detection is concerned with finding similar pattern in source code, interpreting and using them in design, testing and other software engineering problems [17,18,19]. They can be based on text, lexical or syntactic structure, or semantics. A piece of code, A, is semantically similar to another piece of code, B, if B subsumes the functionality of A, in other words, they have "similar" condition. Duplicated fragments will be significantly increase the work to be done when enhancing or adapting code, and increases the maintenance cost.

5.2 Methodology

In this work NN and metric based approach will be used for clone detection. The whole implementation will take place in following manner:

1. Input data
2. Apply Metric based method to get features extraction. Clone detection first parses the source code and then performs the program analysis on the parsed code. All similar code segments are identified and then inconsistency detection is performed. The entire procedure is executed and the results are stored in database. As first part of the analysis through parsing of source code, they find the similarity and in the second phase the metrics are calculated and finally they are detected.

   In textual analysis all types of codes fragments are detected. Each metrics values are stored in a particular database. The input source is identified using metrics and the similarities of code are detected. The metrics values the possible potential clone pairs are extracted. The metrics are computed for each of the methods identified and the values are stored in the database. The various metric values for the code fragment. The descriptive statistics of the metric values obtained for the various methods. While computed metrics values, the method pairs with equal or similar set of values are identified by comparison of the records in the database. In proposed method various metrics has been taken like Public Variables, Private Variables, No. of variables, Function Overloading, No. of functions. Metrics are calculated from names, layout, and expression and (simple) control flow of functions and clones is defined as a pair of whole function bodies with similar metrics values
3. After feature extraction, next step will be the classification of code clones using Neural Network. For the prediction of code clone, data is collected and normalized. Then a single layer perception neural network is created and trained with the given dataset. After training, the network is tested using the testing dataset and it predicts whether the software project classes have the code clones or not.
4. Now we test and validate the neural network implementation using FAR, FRR, Precision, Recall and accuracy parameter.

5.3 Work Model

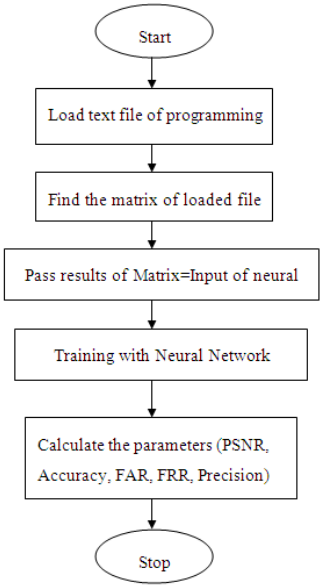

Fig.3: Proposed Flowchart

5.4 Algorithm

```
[Filename,Pathname] = Uigetfile('*.Txt','SELECT A CODE FILE IN TEXT FORMAT')

Get Public
Get Private
Get Overloaded
Get LOC
Get Total Functions
Find Total Repetitions
Get Details = Details';

Metrices(1) = Loc;
Metrices(2) = Count;
Metrices(3) = Round(Total_Rep/4);
Metrices(4) = Total_Overloaded;
Metrices(5) = Public_Count;
Metrices(6) = Private_Count;

Net = Newff(Training_Set,Target,20);
Net.Trainparam.Epochs = 50;
Net.Trainparam.Goal = 0.01;
[Net,Tr,E] = Train(Net,Training_Set,Target);
Error_Rate = Mean(E);
J = Sim(Net,Target);
J = Mean(J);
Result = J;

Find FAR
FRR
Accuracy
```

VI. RESULTS AND DISCUSSION

a. Analysis

Below figures represent the code cloning analysis using proposed technique in MATLAB 2010a. Clone cloning was estimated for 3 code clone manuals. Three clone pairs were rated for different manuals named as software manual-1, software manual-2, and software manual-3 in terms of accuracy.

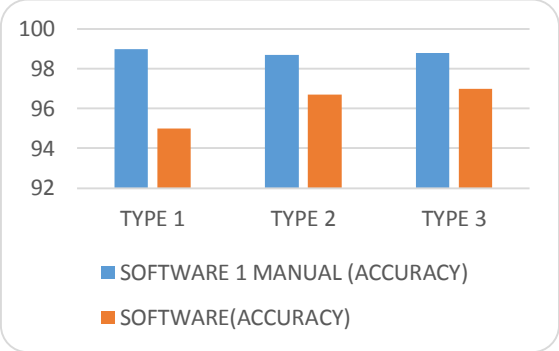

Fig.4: Software 1 Manual (Accuracy)

This work proposed a technique to detect tree types of clones in software. Through an empirical study on a dataset that includes over code lines, they revealed that approximately 95% code clones are found to be of type-1, for type-2 it has found to be 96.7% and for type-3 it has found to be 97% using proposed algorithm. Below table shows the empirical obtained values for proposed as well as traditional manual method of finding code clones in the software.

Table 1: Software 1 Manual Accuracy

CATEGORY	SOFTWARE 1 MANUAL (ACCURACY)	SOFTWARE (ACCURACY)
TYPE 1	99	95
TYPE 2	98.7	96.7
TYPE 3	98.8	97

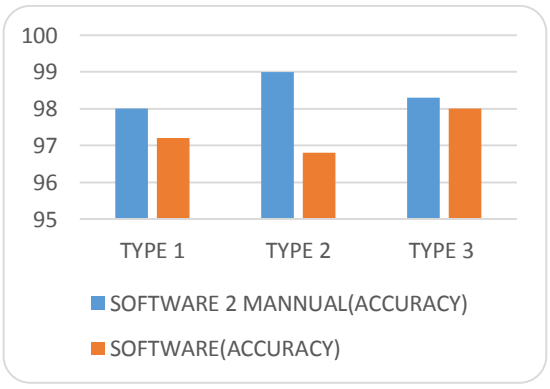

Fig.5: Software 2 Manual(Accuracy)

A clone detector algorithm must try to find pieces of code of high similarity in a system's source text. The main problem is that it is not known beforehand which code fragments may be repeated. Above figure shows the accuracy for code detection in software manual-2 and it has been shown that approximately 97% code clones are found to be of type-1, for type-2 it has found to be 96.7% and for type-3 it has found to be 98% using proposed algorithm. Below table shows the empirical obtained values for proposed as well as traditional manual method of finding code clones in the software.

Table 2: Software 2 Manual Accuracy

CATEGORY	SOFTWARE 2 MANNUAL (ACCURACY)	SOFTWARE (ACCURACY)
TYPE 1	98	97.2
TYPE 2	99	96.8
TYPE 3	98.3	98

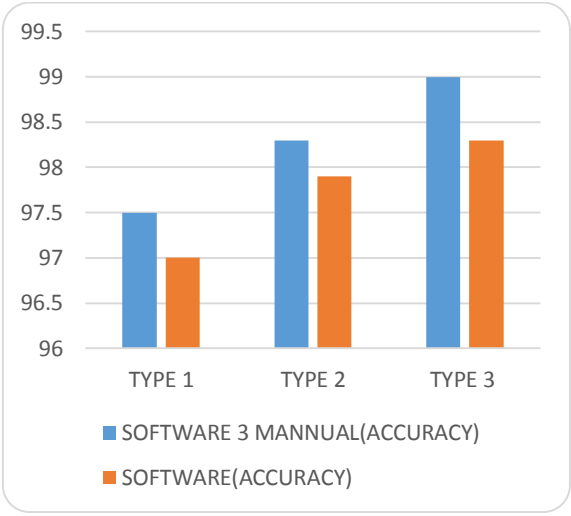

Fig.5: Software 3 Manual (Accuracy)

Above figure shows that after extracting the original source code, clones are subjected to a manual analysis where false positive clones are filtered out by a human expert as well as automatically and it has been found out that 97% code clones are found to be of type-1, for type-2 it has found to be 97.7% and for type-3 it has found to be 98% using proposed algorithm. Below table shows the empirical obtained values for proposed as well as traditional manual method of finding code clones in the software.

Table 3: Software 3 Manual Accuracy

CATEGORY	SOFTWARE 3 MANNUAL(ACCURACY)	SOFTWARE (ACCURACY)
TYPE 1	97.5	97
TYPE 2	98.3	97.9
TYPE 3	99	98.3

VII. CONCLUSION AND FUTURE SCOPE

Cloning of code has become one of the easiest ways to complete a project, who does not want to invest their time on doing programming their project. It's a loss for those who really works hard for the project coding. The date no such method has present who can evaluate the cloning for several languages with one piece of code. The purpose research work has overcome the drawbacks of the previous attempts by removing the bar of the language which follows the architecture of C++. The results have been verified using FEED FORWARD BACK PROPOGATION NEURAL NETWORK over the metrics. A successful accuracy of 97.9% have been achieved. Through the current research quite effective, but still there is a scope of improvement in it. The future research workers may try their hand in enhancing the current algorithm for NON OBJECT ORIENTED PROGRAMMING architecture.

VIII. REFERENCES

[1] R. Koschke, Survey of research on software clones, in: Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, 2006, p. 24.

[2] N. Kraft, B. Bonds, R. Smith, Cross-language clone detection, in: Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering, SEKE 2008, 2008, p. 6.

[3] J. Krinke, Identifying similar code with program dependence graphs, in: Proceedings of the 8th Working Conference on Reverse Engineering, WCRE2001, 2001, pp. 301_309.

[4] I. Landwerth, Clone Detective. Last Accessed November 2008. URL http://www.codeplex.com/CloneDetectiveVS.

[5] F. Lanubile, T. Mallardo, Finding function clones in web applications, in: Proceedings of the 7th European Conference on Software Maintenance andReengineering, CSMR 2003, 2003, pp. 379_386.

[6] S. Lee, I. Jeong, SDD: High performance code clone detection system for large scale source code, in: Proceedings of the Object Oriented ProgrammingSystems Languages and Applications Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems,Languages, and Applications, OOPSLA Companion 2005, 2005, pp. 140_141.

[7] A. Leitão, Detection of redundant code using R2D2, Software Quality Journal 12 (4) (2004) 361_382.

[8] H. Li, S. Thompson, Clone detection and removal for erlang/OTP within a refactoring environment, in: ACM/SIGPLAN Workshop Partial Evaluationand Semantics-Based Program Manipulation, Proceedings of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation,2009, pp. 169_178.

[9] C. Liu, C. Chen, J. Han, P. Yu, GPLAG: Detection of software plagiarism by program dependence graph analysis, in: Proceedings of the 12th ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2006, 2006, pp. 872_881.

[10] B. Baker, R. Giancarlo, Sparse dynamic programming for longest common subsequence from fragments, Journal Algorithms 42 (2) (2002) 231_254.

[11] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, K. Kontogiannis, Measuring clone based reengineering opportunities, in: Proceedings of the IEEE Symposium on Software Metrics, METRICS 1999, 1999, pp. 292_303.

[12] D. Wang, "Pattern recognition: neural networks in perspective, "IEEE,vol. 8, 1993, pp.5-60.

[13] H.A. Rowley, S. Baluja, T. Kanade,"Neural Network-Based Face Detection, " PAMI, 1998, pp. 23 – 28.

[14] S. Bellon, Vergleich von technikenzurerkennungdupliziertenquellcodes, Diploma Thesis, University of Stuttgart, 2002.

[15] S. Bellon, R. Koschke, Detection of software clone: Tool comparison experiment, December 2007. http://www.bauhaus-stuttgart.de/clones/.

[16] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, Comparison and evaluation of clone detection tools, Transactions on Software Engineering 33(9) (2007) 577_591.

[17] M. Bruntink, A. Deursen, R. Engelen, T. Tourwe, On the use of clone detection for identifying crosscutting concern code, Transactions on SoftwareEngineering 31 (10) (2005) 804_818.

[18] P. Bulychev, M. Minea, Duplicate code detection using anti-unification, in: Spring Young Researchers Colloquium on Software Engineering, SYRCoSE2008, 2008, p. 4.

[19] P. Bulychev, CloneDigger K. Church, J. Helfman, Dotplot: A program for exploring self-similarity in millions of lines for text and code, Journal of American Statistical Association (2) (1993) 153_174.