IJCSC 0973-7391
Volume 7 • Number 1 Sept 2015 - March 2016 pp. 259-263
available online at www.csjournals.com

Design and Analysis of a Levenshtein Distance Based Code Clones Detection Algorithm

Jai Bhagwan[1]
[1]Assistant Professor, Department of Computer Science & Engineering, Guru Jambheshwar University of Science & Technology, Hisar, India

Abstract: Code clones detection is becoming a hot issue as the number of web and desktop based applications are increasing day to day. Reusing the existing modules with or without little change can result in code clones while developing software. The clones can be categorized as type-1, type-2, type-3 and type-4.The significance of cloning is that it increases the risk of software maintenance and increases the complexity as well. By clone detection and re-factorization, the maintenance process can be made easy. Various techniques have been developed in recent years and can be based on string matching, token-based, semantic-based, tree-based etc.In this paper, a novel method has been proposed usingLevenshtein Distance method and type-1 clones have been detected.A tool named JB Clone Scanner is developed in order to implement and validate the proposed technique.
Keywords: Code Clones, LOCs (Lines of Code), JB Clone Scanner, Levenshtein Distance.

1. Introduction

The demand for a new software application is increasing day by day and the numbers of professional are not increasing in that ratio. Software professional have found reusability as a powerful tool which helps in software development with ease [8]. Reusing a code segment with or without a minor change is known as "Code Cloning" while developing software applications. The pasted code is known as code clone [4] [6]. Clones can be categorized as follows [12]:

- Type-1: These are the clones which are identical except the variations of whitespaces and comments.
- Type-2: These are syntactically identified code fragments exceptthe variations of identifiers, comments, literals, types, and layouts.
- Type-3: These are copied code with additional modifications. Here, statements can be added, modified or removed in addition to disparities in identifiers, literals, types, comments or layouts.
- Type-4: These are code clones which known by identified by functional similarities of code segments.Here, the syntaxes implementation of these code segments may be different.

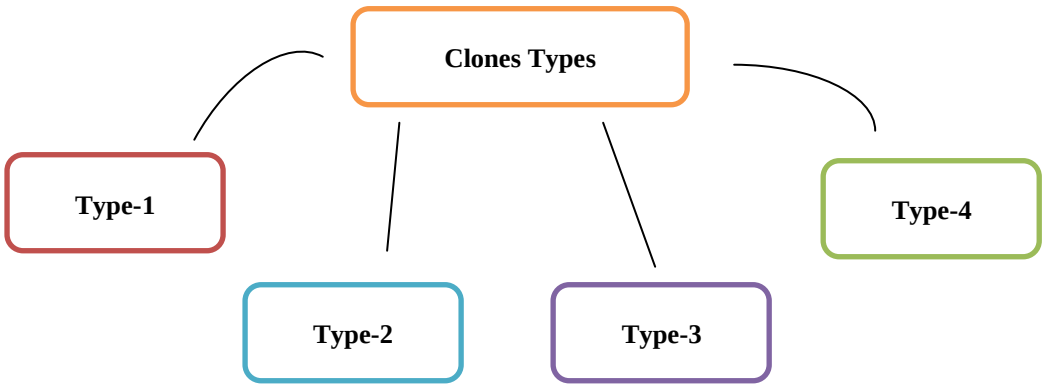

Figure 1. Classification of Code Clones

The code clones increase the software complexity and it would be difficult to maintain the software having a large number of clones [9]. One way of removing the clone is refactoring [5]. There are various approaches available to detection the clones in a software source code. Among these few important approaches are described as [11]:

- Textual-Based Approach –In this approach, complete lines are matched using hashing techniques of strings.

DOI: 10.090592/IJCSC.2016.045

- Token-Based Approach – In this approach, the lines' token sequences are used to compare the code and clones are detected based on that.
- Metric-Based Approach – Here, the code is not mapped directly but different metrics are collected and compared in order to find the clones.
- Abstract Syntax-Based Approach – In the AST approach, the subtrees of the AST (Abstract Syntax Tree) of programming code are generated using a hash function and then subtrees are compared through tree matching.

In this paper, the Levenshtein Distance-based algorithm has been designed to detect clones of software programs. The rest of the paper is organized in four parts. In section 2, literature review or related work is being presented. Section 3 will describe the proposed algorithm. Section 4 will present results and discussion and in section 5, a conclusion has been drawn.

Levenshtein Distance Algorithm was introduced by Valdimir Levenshtein in 1965. The pseudo-code of this algorithm is shown in figure 2.

Levenshtein Distance Algorithm Pseudo-Code

```
LevenshteinDistanceAlgo (Source, Target)
Int Distance = new Int [Source.Length + 1, Target.Length + 1)
If (Source.Lenght = = 0)
|       return Target.Length
End If
If (Target.Lenght = = 0)
|       return Source.Length
End If
For Int I=0 to Source.Length
|       Distance [I, 0] = I
End For
For Int J=0 to Target.Length
|       Distance [0, J] = J
End For
For Int I=1 to Source.Length
|       For Int J=1 to Target.Lengh
|       |       Int Cost = (Target [J - 1] = = Source [I - 1] ? 0 : 1
|       |       Distance [I, J] = Min (Min (Distance [I - 1, J ] + 1, Distance [I, J - 1] + 1),
|       |       Distance [I - 1, J - 1] + cost)
|       End For
End For
return Distance [Source.Length, Destination.Length]
```

Figure 1. Levenshtein Distance Algorithm Pseudo-Code

2. Literature Review

Various techniques have been proposed to detect code clones in software systems in the past. In [4], scientists designed a new technique for code clone detection which is based on transformation source text and token comparisons. A tool named CCFinder has been developed using various optimization techniques which can detect clones from C, C++, COBOL and, Java-based projects. Various case studies have been applied on CCFinder and it has been found effective. In [5], the authors proposed a technique for detection of some higher level similarities in source codes using a data-mining technique. In order to detect the clones, the scientist developed a tool named Clone Miner and tested with various case studies. The researchers [6] proposed a hybrid technique which is a combination of a textual and metric-based method. Authors utilized various metrics and compared with other approaches and found that the proposed technique is more efficient and accurate. The scientists [7] said that automatic categorization is a new and effective method for software archive. The function-oriented approach is better than object-oriented for categorization of software modules. Naïve Bayes

DOI: 10.090592/IJCSC.2016.045

scheme performed better than other existing techniques. In [9] the authors described a practical solution for the detection of higher level similarities among classes and files. First, simple clones were detected using a conventional token-based approach. Then, the authors found clones using Frequent Itemset Mining technique which was a novel technique. The experiments confirmed that the proposed technique was better. The author in [10] presented three clones' detection algorithms. The scientist worked on transformed sequence similarity and sub-trees. In [11], the researchers proposed a hybrid technique which is a combination of metrics and textual based approaches. The proposed technique provided less complexity and gave accurate results. In [12], the scientists provided a vast comparison of clone detection techniques and tools. The authors in [3] designed a token-based approach for code clone detection which is accurate and scalable. Authors conducted experiments using Linux kernel 2.6.38.6 and JDK 7 source code. The experiments disclosed that the proposed tool Deckard detected clones with less execution time. Authors of [1] proposed a Light Weight Hybrid technique using textual and metrics approaches for detection of method-level clones in Java and C projects. The authors designed a tool named CloneManager which detected clones in order to validate the proposed technique. In [2], the researchers proposed a code clone detection hybrid technique that depends on metrics and template conversion based techniques. After simulation, it was found that the proposed technique is better and having less complex than other existing techniques.

3. Proposed Method

The proposed method has been designed by using Levenshtein Distance method which was introduced by Vladimir Levenshtein in 1965. The proposed algorithm is depicted in figure 3.

Proposed Algorithm Pseudo-Code

```
Foreach Object in DestinationFiles
|       ListElements.Add (Object.Value)
End Foreach
For I=0 to ListElements.Count
|       For J=I+1 to ListElements.Count
|       |       If ListElements[I] = = ListElements[J]
|       |       |     ListClones.Add (ListElements[I])
|       |       End If
|       End For
End For
ListClones.RemoveDuplicate
Foreach Object in DestinationFiles
|       ListFiles.Add  (Object.Value)
|       ListLoc.Add (ListFiles.Count)
|       Foreach ItemClone in ListClones
|       |       Foreach ItemFile in ListFiles
|       |       |       Cost = LevenshteinDistanceAlgo (ItemClone, ItemFile)
|       |       |       If (Cost = = 0)
|       |       |       |       VarClones ++
|       |       |       End If
|       |       End Foreach
|       End Foreach
|       ListCloneFound.Add (VarClones)
End Foreach
```

Figure 3. Clone Detection Algorithm

DOI: 10.090592/IJCSC.2016.045

Volume 7 • Number 1 Sept 2015 - March 2016 pp. 259-263
available online at www.csjournals.com

4. Simulation Results

For the purpose of simulation a computing machine was used having 2GB of RAM, Intel Quad Core 2.0 GHz processor and Windows 7 OS. A tool named as JB Clone Scanner was developed using C#.net 2008 in order to validate the proposed algorithm. An experiment was conducted using 15Java Programs. Table 1 represents the results obtained by JB Clone Scanner.

Table 1. Clones Detected by Proposed Tool

Programs (Modules)	LOC (Lines of Code)	No. of Clones Detected
1	43	11
2	30	9
3	52	13
4	32	9
5	22	7
6	19	6
7	58	15
8	25	9
9	18	5
10	19	7
11	22	6
12	16	4
13	13	3
14	34	9
15	24	8

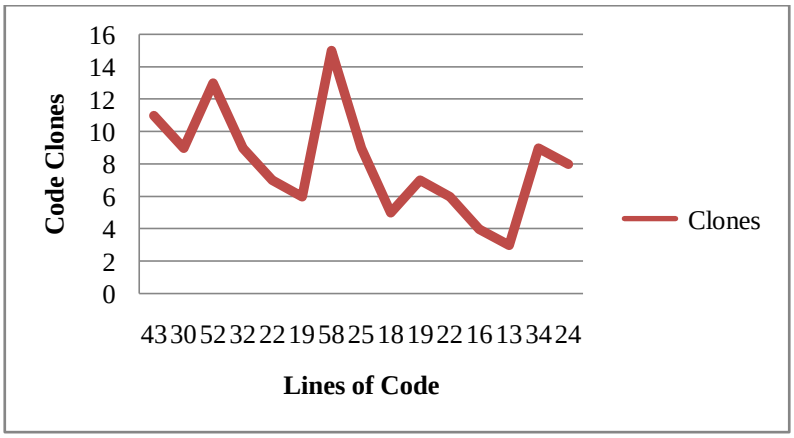

Figure 4. Clones Detected in various Programming Modules.

DOI: 10.090592/IJCSC.2016.045

Figure 4 is demonstrating the type-1 clones detected by JB Clone Scanner.

5. Conclusion

Code clone detection is an interesting research topic. Various accurate and efficient techniques have been proposed and simulated. In this paper, a Levenshtein Distance based string matching algorithm has been designed in order to find the code clones among various software programs. The proposed approach has been validated by developing a tool named JB Clone Scanner. The experiment was carried out using 15 Java Programs. The proposed technique is capable to find out type-1 clones. In the future, a hybrid algorithm using existing techniques can be developed for better results as well as to find out other types of clones.

References

[1] E. Kodhai and S. Kanmani, "Method-level Code Clone Detection through LWH (Light Weight Hybrid) Approach," Journal of Software Engineering Research and Development, Vol. 2, pp. 1-29, 2014.

[2] G. R. Goda and A. Damodaram, "An Efficient Software Clone Detection System based on the Textual Comparison of Dynamic Methods and Metrics Computation," International Journal of Computer Applications, Vol. 86, No. 6, pp. 41-45, 2014.

[3] Y. Yuan and Y. Guo, "Boreas: An Accurate and Scalable Token-Based Approach to Code Clone Detection," ASE 12, Essen, Germany, pp. 286-289, 2012.

[4] T. Kamiya, S. Kusumoto and K. Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code," IEEE Transactions on Software Engineering, Vol. 28, No.7, pp. 654-670, 2002.

[5] H. A. Basit and S. Jarzabek, "A Data Mining Approach for Detecting Higher-level Clones in Software," IEEE Transactions on Software Engineering, pp. 1-18, 2007.

[6] E. Kodhai, S. Kanmani and A. Kamatchi, "Detection of Type-1 and Type-2 Code Clones Using Textual Analysis and Metrics," Internation Conference on Recent Trends in Information, Telecommunication and Computing, IEEE, pp. 241-243, 2010.

[7] P. S. Sandhu, M. Bala and H. Singh, "Automatic Categorization of Software Modules," International Journal of Computer Science and Network Security, Vol. 7, No. 8, pp. 114-119, 2007.

[8] P. S. Sandhu, J. Singh, H. Singh, "Approaches for Categorization of Reusable Software Components," Journal of Computer Science, Vol. 3, No. 5, pp. 266-273, 2007.

[9] H. A. Basit and S. Jarzabek, "Detecting Higher-level Similarity Patterns in Programs," European Software Engineering Conference, ACM SIGSOFT, 2005.

[10] K. Greenan, "Method-level Code Clone Detection on Transformed Abstract Syntax Trees Using Sequence Matching Algorithms," Department of Computer Science, University of California, 2005.

[11] E. Kodhai, A. Perumal and S. Kanmani, "Clone Detection using Textual and Metric Analysis to figure out all Types of Clones," International Journal of Computer Communication and Information System, Vol. 2, No. 1, pp. 99-103, 2010.

[12] K. R. Chanchal, J. R. Cordy and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," Science of Computer Programming, Elsevier, Vol. 74, pp. 470-495, 2009.

DOI: 10.090592/IJCSC.2016.045