An Effective Software Clone Detection Using Distance Clustering

D. Gayathri Devi #1, Dr.M.Punithavalli *2
#Assistant Professor, Department of Computer Science, Research Scholar Karpagam University, Coimbatore, Tamilnadu,India
[1] dgayadevi@gmail.com
* Director and Head, Sri Ramakrishna Engineering College, Coimbatore, Tamilnadu,India
[2] mpunitha_srcw@yahoo.co.in

Abstract – As the computer is a rapidly evolving, there is tremendous need of software development for different purpose. The complexity of software development differs, and the developers take the easier way of implementation by copying fragments which leads to code clone. This paper presents a technique for detecting code clone using fragment distance with clustering. First, we tokenize the source code into tokens. Second, by distance and clustering we find the similarity until all clusters are merged. Third, we evaluate and find the code fragments using distance cluster DC and finally, we provide the examples using distance.

Keywords− Metrics, Edit Distance, Detection, Code Clone, Fragment

I. INTRODUCTION

Code Clone is a phenomenon that occurs frequently in large systems. These code clones arises due to certain reasons such as making a copy of a code fragment. This leads to code clone, on justification it is considered to be a bad practice. Especially during the maintenance, this unjustified code gives rise to numerous problems: 1) if one repairs a bug in a system with code clone, all possible clone of that bug must be checked. 2) If the code clone increases the size of the code, the compile time will be more [1].

Techniques and tools for detecting code clone are thus highly desired commodity especially in software maintenance community and several researches has been proposed a number of approaches with capable results. However, still the code clone arises in large software systems.

Code clones in software system are one of the major factors in decreasing maintainability. Many code clone detection methods have been proposed to find code duplication automatically from large scale

software. However, it is still hard to find code duplication to improve maintainability because there are many code duplications that should remain. A code clone is a code portion in source file that is identical or similar to another. This is a major problem in software development for different reasons. Thus the source code becomes larger and more difficult to understand. Clones seem to be a desirable approach to development as it is associated with reuse, implementation speed-up and development. However, the code implication can be very negative.

II. CODE CLONE

A. Definition

A code clone, in general, means a code fragment that has identical or similar code fragment in source code. However, there is no generic definition for code clone. Several methods of code clone detection have been proposed, and each of them has its own definition about code clone. Still code cloning is considered as a serious problem in industrial software, [2], [3], [5], [11] [12], [13], [14].

The literature on the topic has been described many situations that can lead to the duplication of code within software system and are considered to be spiteful cloning. Developers may duplicate code because the short term cost of forming the proper abstraction may prevail over the cost of duplicate code [15], [16]. Developers may also duplicate code when they fully understand the problem, or the solution, but they are aware of the code that can provide some functionality [4][6][7].

B. Code clone types

Type I:

Identical code fragments except for variations in whitespace and comments called Exact clone.

Type II:

Structurally or syntactically identical fragments except for variations in identifiers, literals, types, layouts and comments called Renamed clones.

Type III:

Copied fragments with further modifications. Statements can be changed, added or removed in addition to variations in identifiers, literals, types, layouts and comments called Gapped clones.

Type IV:

Code fragments that perform similar functionality, but are implemented by different syntactic variants called Semantic clones.

II. RELATED WORK

A brief introduction to all the basic algorithms of string matching

A. *Hamming Distance*:

It is the technique of finding strings that match a pattern approximately. The transmissions in hamming distance are done using binary. Every letter is encoded in a string of same length. The problem of approximate string matching is typically divided into two sub problems: finding approximate substring matches inside a given string and finding dictionary strings that match the pattern approximately.

B. A*lgorithm using Levenshtein Distance:*

Levenshtein distance is a metric for measuring the amount of difference between two sequences (i.e. an edit distance). The term edit distance is often used to refer specifically to Levenshtein distance. The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character.

C. Euclidean distance:

It is used to cluster the vectors using metric. The vectors in one clone are considered to be similar. D(v1,v2) = || v1- v2||.

D. Damerau Levenshtein Distance:

The Damerau-Levenshtein distance metric is a task, from finite strings haggard from an character, to an integers. It is a distance metric with that of given strings. The distance D(s1, s2) is defined as: One inserting a character, substituting one character for another , deleting a character, and transposing two adjacent characters. There may be any combinations of these above four operations. It convert the string s1 to s2, but the length of the shortest sequence is the distance between the two strings.

E. *Boyer Moore Algorithm:*

It is a particularly efficient string searching algorithm, and it has been the standard benchmark for the practical string search literature. The algorithm preprocesses the target string (key) that is being searched for, but not the string being searched in (unlike some algorithms that preprocess the string to be searched and can then amortize the expense of the preprocessing by searching repeatedly). The execution time of the Boyer Moore algorithm, while still linear in the size of the string being searched, can have a significantly lower constant factor than many other search algorithms: it doesn't need to check every character of the string to be searched, but rather skips over some of them. Generally the algorithm gets faster as the key being searched for becomes longer. Its efficiency derives from the fact that with each unsuccessful attempt to find a match between the search string and the text it is searching, it uses the information gained from that attempt to rule out as many positions of the text as possible where the string cannot match. BMH approach uses only the Bad character Heuristic of Boyer Moore for skipping comparisons rather than both the Bad Character and Good Suffix Heuristics.

F. *Jaccard Similarity:*

It is a measure of the similarity between two binary vectors. It can be simply used to measure the similarity of the strings.

IV. APPROACH

Algorithm Description

We follow the standard definition and use editing distance as the measure for similarity of tokens or fragments.

Editing Distance: The *editing distance* of two strings S1 and S2, denoted by $\delta(S1, S2)$, is the *least progression* of edit operations.

Sting Similarity Two string S1 and S2 are σ-*similar*, if $\delta(S1, S2) < \sigma$.

Clone Pair Two code fragments C1 and C2 are called a *clone pair* if the string S1 and S2 are σ-similar.

A. Similarity Measure

Before clustering a similarity or distance measure must be determined. The measure reflects the degree of closeness of objects and corresponds to the characteristics to distinguish the clusters. Since clustering is the grouping of similar instance or objects some sort of measures can be determined whether the two objects are similar.

B. Metric

To quantify a metric a measure d must satisfy the following four conditions:

Let a and b be any two objects in a set and d(a,b) be the distance between a and b.

1. The distance between any two points must be non negative that is $d(a,b) \geq 0$.
2. The distance between two objects must be zero if and only if the two objects are identical, $d(a,b) = 0$ if $a = b$
3. Distance must be symmetric, distance from a to b is the same as the distance from b to a. i.e
$$d(a,d) = d(b,a)$$
4. The measure must satisfy the triangle inequality, which is
$$d(a,c) \leq d(a,b) + d(b,c).$$

C. Clustering

Clustering is a partition of data into groups of similar objects. Clustering can be considered the most important unsupervised learning problem: so every other problem deals with finding structure in a collection of unlabeled data. Many clustering technique are based on the techniques known as partial and hierarchical clustering like fuzzy c-means, k-means and hierarchical clustering agglomerative. Clustering is a process of examining the points and grouping the points into "clusters" according to distance measure.

V METHODOLOGY

A. Tokenizing

The input to the program is the source code. Each line of the source code is partitioned into tokens based on lexical rules. There are different rules in acquiring the token for particular lexical. These token forms the token sequence that are to be compared. All unnecessary code such as whitespaces, tabs etc are removed from token sequence. The token of the source files are then concatenated into a token sequence, such that finding multiple files is performed as a single file. In this pace the whitespace and comments are removed from the token sequence for further process.

In next step, the token are fed into distance algorithm and identify the similar statements and are then partitioned into clusters. Next, identify the sequence of clusters and on examining the overall similarity and finally calculate the percentage of similarity of the sequence.

B. Representation of source code

In this process the source code are reconstructed as $m \times k$ matrix where each row represents the tokens for m source files. It is given by $File^s = (T_{s,1}, T_{s,2}, \ldots T_{s,k}) where\ T_{s,i} = 1$ indicate the token in the source file else$T_{s,i} = 0$. The k represents the number of source files. The table 1 shows the representation of source files along with the token in source files.

Table 1: Source Files with Token

	T1	T2	…	Tk
File 1	0	1	…	1
File 2	1	0	…	1
⋮	⋮	⋮	…	⋮
File m	1	1	…	1

C. Euclidean Distance

The similarity of any two programs can be determined with the help of distance measure. Euclidean distance, the most common distance measure, is the geometric distance in multidimensional space. Here, for identifying similar statements Euclidean distance by equation (1) is used for computing the similarity of source code (*Sc*) as $SC_i$and Sc_j, and the similarity can be indicated by $Sim(Sc_i, Sc_j) = (token_i, token_j)$. In addition the Euclidean distance is normalized in the equation (2). This results in $m \times m$ matrix.

$$ED(sci, scj) = \sqrt{\sum_{n=1}^{k} \left(t_{i,n} - t_{j,n}\right)^2} \quad (1)$$

Normalized ED:

$$NED(sci, scj) = 1 - \sqrt{\frac{\sum_{n=1}^{k} \left(t_{i,n} - t_{j,n}\right)^2}{k}} \quad (2)$$

Consider the data matrix i.e tokens

Source Code	Tok1	Tok2	Tok3	Tok4	…
File A	$X_{A,1}$	$X_{A,2}$	$X_{A,3}$	$X_{A,4}$	⋯
File B	$X_{B,1}$	$X_{B,2}$	$X_{B,3}$	$X_{B,4}$	⋯
File C	$X_{C,1}$	$X_{C,2}$	$X_{C,3}$	$X_{C,4}$	⋯
⋮	⋮	⋮	⋮	⋮	

Calculate the token-wise matrix as a distance matrix

Source Code	A	B	C	D	…
File A	$ed(A,A)$	$ed(A,B)$	$ed(A,C)$	$ed(A,D)$	
File B	$ed(B,A)$	$ed(B,B)$	$ed(B,C)$	$ed(B,D)$	
File C	$ed(C,A)$	$ed(C,B)$	$ed(C,C)$	$ed(C,D)$	
⋮	⋮	⋮	⋮	⋮	

Iterating the above matrix in the same manner all the clusters would merge. Euclidean distance is widely used in clustering problems, including clustering text. As it satisfies all the four metric condition and therefore it is a true metric.

D. Agglomerative hierarchical clustering

Each object initially represents a cluster of its own. Then the clusters are successively merged until the desired cluster structure is obtained. To calculate agglomerative clustering, we use the distance matrix. From the distance matrix, the distances between points is examined and are merged together with their distance as clusters.

1. Find the smallest distance between two elements
2. The two elements are clustered together, that becomes the new element
3. Iterations step1 and step2 are repeated until all elements are clustered

The similarity is obtained by Euclidean distance. Initially the entire source code or files are predicted to be clusters. On comparison the same token or code are identified and combined into cluster until the terminal condition. Finally, the pair of clones is merged into the cluster and the clusters are obtained as a result.

Clone Detection Algorithm

1. Input: Input the source file
2. Length of source file
   2.1 Set the length of m and n source code to s and t respectively.
   2.2 If both the length of s and t respectively is equal to zero then exit.
3. Tokenizing : Partition source file into tokens
4. Initialization of matrix
   4.1 Create a matrix with m rows and n columns.
   4.2 Initialize the first row and column as 1 to m and 1 to n respectively.
5. Inspecting each characters
   5.1 Evaluate each fragment of s and t from 1 to m and 1 to n respectively
   5.2 If s[i] = t[j] the fragments gets equal and the distance value is 1.

5.3 If s[i] ≠ t[j] the fragments are not equal the distance value is 0.

6. Find the similar statements using Euclidean distance
7. Partition the similar statements into clusters.
   7.1 Each token be a cluster
   7.2 Find a pair of the most similar cluster and merge
8. Find the identical sequence of clusters
9. Identify the overall similarity of code sequence
10. Find the percentage of similarity using Boolean algebra of two sequence

E. Finding identical sequence of clusters

After finding each statement as clusters, we search for all pairs of sequence of statements, which are identical. Detected pairs are clones and have to be checked for similarity on the statement level in following phase.

F. Examining overall similarity

After finding the identical clusters, set the clone candidates. Assume the candidates are sequence of statements. To check the pair of similarity between sc1 and sc2 is compared using the Euclidean distance. If the distance between them is below threshold, then it is reported as clones. The overall distance between two sequences cannot be obtained by summing the distance. The similarity value SC is a set of pairs which consists of token. The function $simSC(SC1, SC2)$ compares all tokens with SC1 with SC2, and then count the number of matched pairs *n*.

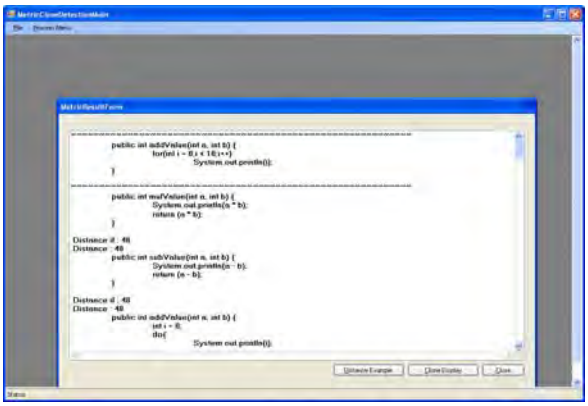

Fig.1 Similariy Distance

Fig. 1 specify the similarity distance of the fragements. The goal id to access the possibility of detecting clones with distance. We perform the distance based on token strings instead of images strings. Since the distance is computed on token the algorithm report them as clones. Our method is equivalent to perform *n/2* times for *n* number of lines. Where *n* is the total number of fragements in source code. Experiments have been performed in Java systems. Most of the cod have been imbedded in control structure containing loop statements. The result gives the great interest in clone detection, since we do not need to know the distance of non-clone fragements. The Percentage of similarity is given by the similarity of code divided by the total number of lines of code.

VI EXPERIMENTAL RESULTS

We have evaluated clone code rate (number of detected code) and its scalability. Our results indicate that Distance Clustering DC performs significantly better than distance and cluster. We measure the clone rate by the number of lines of codes that are within detected clone. Here the minimum number of required for clones (mintok) was set to 20 or 50, tramp (size of clustering ranged from 4 to mc (no merging of token for clustering) and similarity between 1.0 and 2.0. The following figure 2 shows the similarity and the corresponding code rate.

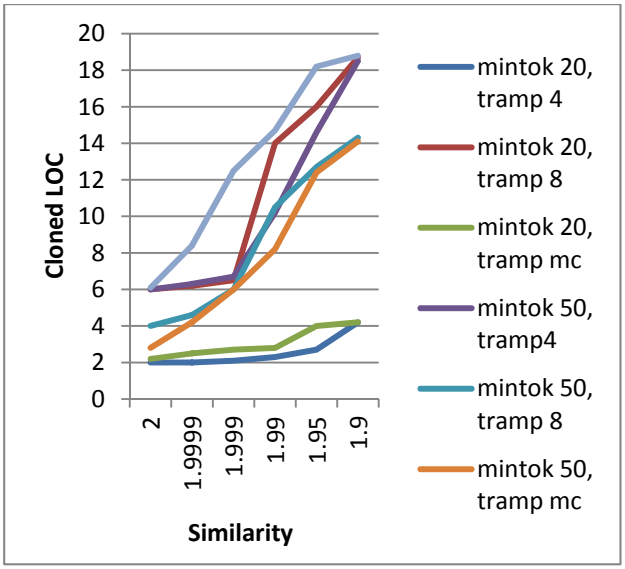

Fig. 2 Cloned LOC

For evaluating the proposed technique the source code is selected form the industrial and the IT company. The source code is considered for the categories of payroll, accounts, stores, transport, textile-production, It industries that are developed for various categories. The resulted code clone rate % is indicated in the Fig 2, are considered for four source files A, B, D and I. The code clone rate of global view from 35% on A source file relevance to 55% on I source file relevance. The code clone rate of proposed distance clustering (DC) view is greater than 50%. Evaluating clone percentage per file identifies the degree of clone 35% to 55% for A, B, D and I and 45% to 70% for C, E, F, G and H. The code clone rate DC is greater than 50%. From the result, it can be observed that the DC view is better than clustering view and distance view. The following Fig 3 shows the difference between existing and proposed based on code clone rate.

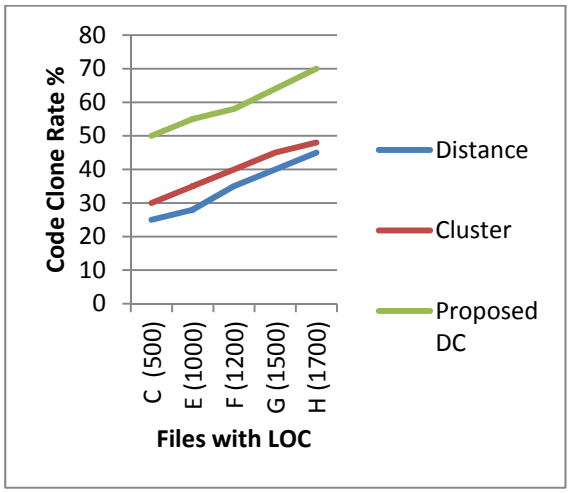

Fig 3 Percentage of Cloned rate

VII. CONCLUSION

In this paper, we have presented an automatic clone detecting technique. First, by preprocessing the source code is partitioned. Second, on extraction we split the functions from the source code. Third, we evaluate and find the code fragments using Distance clustering and finally, we provide the detected code clone. Finally, we evaluated the performance by analyzing structural clones found in software systems. On detecting code clones of code fragments, it saves comprehension time and space. We believe that our technique is scalable and useful. On detecting code clones, the quality of code is improved. This tool detects a significant amount of code clones. Identification and subsequent unification of simple clones is beneficial in software maintenance. Our main goal is identify clones and quantify the amount of similarity present.

ACKNOWLEDGMENT

We would like to thank fellow colleagues for valuable insights and comments that helped to improve this paper. And this work was supported by my parents, family members and Mr.P.RamKumar, for their grand support and making the work successful.

REFERENCES

[1] G. S. Manku, A. Jain and A.D. Sarma, "Detecting Near- Duplicates for Web Crawling", Proc. of *WWW*, 2007, pp. 141- 150.
[2] Mark Sevalnev, "Error –Correcting codes introduction, Hamming Distance", 2008.
[3] Gregory V. Bard," Spelling-Error Tolerant, Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric", 2007.
[4] N.Gode and R.Koschke. Incremental clone detection. In European Conference on Software Maintenance and Reengineering, 2009.
[5] R. Koschke, R. Falke and P. Frenzel. Clone Detection Using Abstract Syntax Suffix Trees. In *WCRE*, pp. 253-262, 2006.
[6] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: Finding Copy- Paste and Related Bugs in Large-Scale Software Code. *IEEE TSE*,32(3):176-192, 2006.
[7] J. Mayrand, C. Leblanc and E. Merlo. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In *ICSM*, pp. 244-253, 1996.
[8] Ref: V. Hodge and J. Austin, "A Comparison of a Novel Spell Checker and Standard Spell Checking Algorithms Pattern Recognition", vol. 15 no. 5,pp. 1073-1081, 2003.
[9] C.Roy, J.Cordy, and R.Koschke. Comparison and evaluation of code clone detection techniques and tools: a qualitative approach. May 2009.
[10] S.Schime, C.Vielhauer, J.Dittmann. Using Adapted Leveinsten Distance for On-Line Signature Authentication. Proceeding of the pattern Recognition, 17th International Conference on, 2004.
[11] S.Schime, C.Vielhauer, J.Dittmann. Similarity searching for on-line handwritten documents. – Journal on Multimodal user Interface, 2007 – Springer.
[12] B. Simpson and F. Toussi. *Hsqldb User Guide*. The HSQLDB Development Group, http://hsqldb.sourceforge.net/web/hsqlDocsFrame.html, July 2004. Visited May 2005.
[13] Takeda H., Veerkamp P., Tomiyama T., and Yoshikawam H. Modeling design processes.*AI Magazine*, Winter:pp 37–48., 1990.
[14] V Vaishnavi and B. Kuechler. Design research in information systems. www.isworld.org/Researchdesign/drisISworld.ht , May 2005. Georgia state University and University of Nevada, Visited June 2005.
[15] Li Yujian, Liu Bo, A Normalized Levenshtein Distance Metric, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 6, pp. 1091-1095, June, 2007
[16] Zhao, Zuo-Peng, Yin, Zhi-Min, Wang, Qian-Pin, Xu, Xin-Zheng, Jiang, Hai-Feng and Jisuanji Yingyong "An improved algorithm of Levenshtein Distance and its application in data processing" Journal of Computer Applications. Vol. 29, no. 2, pp. 424-426. Feb. 2009 indepandant.

Authors

1. D.Gayathri Devi graduated with M.C.A., M.Phil in Computer Science., She has presented number of papers in National Conferences and International Seminars and Conferences. She is guiding research scholars. Her area of interests includes Software Engineering and Data Mining. She has 9 years of teaching experience. Currently she is working as a Assistant Professor Department of Computer Science at Sri Ramakrishna College of Arts and Science for Women, Coimbatore, Tamilnadu, India.

2. Dr.M.Punithavalli Ph.D, is presently working as Director and Head, Department of Computer Application at Sri Ramakrishna Engineering College, Coimbatore ,Tamilnadu, India. She has 19 years of teaching experience and presented National and International Conferences and Produced more than 18 M.Phil and several Ph.D so far. Her area of interest includes Data Mining, Software Engineering and Networking etc.