International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-9 Issue-2, December, 2019

Development of Cross Language Clone Detector for C, C++ & Java Repositories using Natural Language Processing

Sanjay B. Ankali, Latha Parthiban

Abstract*: Reusing the code with or without modification is common process in building all the large codebases of system software like Linux, gcc , and jdk. This process is referred to as software cloning or forking. Developers always find difficulty of bug fixes in porting large code base from one language to other native language during software porting. There exist many approaches in identifying software clones of same language that may not contribute for the developers involved in porting hence there is a need for cross language clone detector. This paper uses primary Natural Language Processing (NLP) approach using latent semantic analysis to find the cross language clones of other neighboring languages in terms of all 4 types of clones using latent semantic analysis algorithm that uses Singular value decomposition. It takes input as code(C, C++ or Java) and matches all the neighboring code clones in the static repository in terms of frequency of lines matched.

***Keywords* : Cross language Clones, Porting, Natural Language Processing**

I. INTRODUCTION

Because of several MLOC's of code available on Internet, code search is becoming more common now a day's [1]. It is easier for developers to get code online than to start coding from scratch [1]. But reusing code snippet from the online source adds maintenance cost and software quality [1] or may violate software licenses [1]. Google and other text search engines cannot search source code directly, but can find similar code based on the text search. Existing clone detection techniques can find clones of same language and very few recent studies show that they can find clones of same families. To help the managers and team leaders who involve in porting the code from one language to another, there needs a single code matching process that matches similar code clones in neighboring languages from large online or offline repositories [1].There are many approaches in finding the clones of similar languages [25],[26],[27],[28],[29]. With the invent of many mobile operating systems that provide similar features the popularity of multi language code clone match is on immediate need.

Research question if there are two large codebases of C, C++ & Java, can we find the software clones of same language and cross language? To answer this question through our work let us first define the types of clones.

Type 1. This is also referred as exact clone that has identical code fragment with comments and whitespace tolerance [28]. Example consider following codes.[32]

```
if  (A>=B) {                          if(A>=B){
M = A + B;                              M=A+B //comment1
N = A-B;} // Comment1                 N=A-B }
else                                  else
Z = A*B;         //Comment2           Z=A*B  //comment2
```

Type 2: two or more code fragments identical in syntactic and structural point of view with change in identifier names, variation in comments and indentation [23]. For example consider following code fragments.[32]

```
if(a1!=b1)                         if(m1>n1)//comment1
{  d1 = d1 + 1;                      y1=x1+n1;
}                                  else     y1=x1-m1else
c1 = d1 – a1;//Comment1
```

Type 3: Copied fragments with modifications consisting of addition and deletion of lines [23]. For example consider following code fragments.

```
if (x >= y) {       if (x>= y) {
m = d + y;                m = d + y;
n = d + 1;               alpha = 1;
}else                     d = d + 1;
      m = d - x;           } else m = d - x;
```

Type 4: two are more code fragments that are semantically same (functionality) with different syntax. For example consider following code fragments.[23]

```
int factorial (int n)                  int factorial(int  N)
{                                      {
int j , f=1;                            if (N == 0)
for (j=1; j<=n; j++)                    return 1 ;
f= f * j;                          else return N *factorial (N-1)
 return (f); }  }
```

As mentioned in the related work, there exist many approaches to find clones of same language. We build the cross language clone detector by applying topic modeling approach using tokenization, edit distance calculation and latent semantic analysis technique to find both same language clones as well as clones of different languages.

Revised Manuscript Received on December 30, 2019.
* Correspondence Author

Sanjay Ankali*, department of Computer Science & Engineering, KLECET Chikodi,,India.
Dr. Latha Parthiban, department of Computer Science, Pondicherry University,India.

Input to the clone detector is any source code(C,C++ or Java) and output is classification of types of clone.

Natural Language processing is domain of human computer interaction concerned with processing large amount of structured and unstructured data through techniques such as named entity recognition, sentimental analysis, text summarization, aspect mining and topic modeling [21].

In this paper we first tokenize the source and destination codes to find edit distance among two different codes to find commonalities in the neighboring language, and then filtering is applied to get maximum commonalities. Then topic modeling is applied to the maximum matched snippets using latent semantic analysis to match the cross language similarities which is matrix model. In the last phase based on the thresholds of commonalities we display type1, type2, type3, type4 clones.

II. RELATED WORK

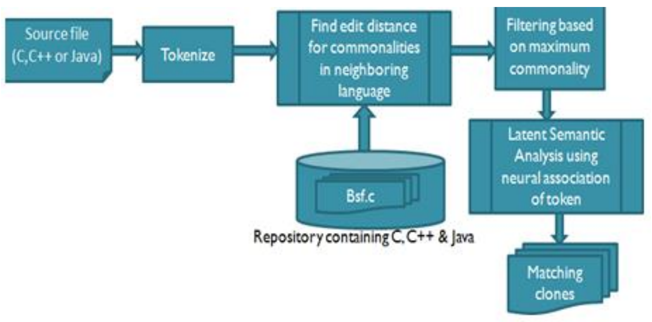

Source file (C,C++ or Java)
Tokenize
Find edit distance for commonalities in neighboring language
Filtering based on maximum commonality
Latent Semantic Analysis using neural association of token
Bsf.c
Repository containing C, C++ & Java
Matching clones

Fig.1 Architecture Cross language clone detection

Software cloning occur when programmer uses existing code to build new feature, where the existing code may remain same or changes can happen with minor addition and deletion of lines to meet the needs of application [3]. Cloning process is common for the developers who involve in software porting, and where these clones act as benchmarks for the quality of software created [4].

Many earlier researches say that clone management has to be carried out properly to avoid maintenance cost in some cases clone managements may not cause serious concern in [9] because consistency of reusing the code plays vital role [10],[11]. Most of the researchers demand separate standalone clone management process [30]. Using the basics of programming languages may also lead to software clone which is unknowingly done based on the syntax and semantics of the programming languages [9] or sometimes using the existing code intentionally to preserve the functionality to add new functionality [10]. IT industry in past has always used software cloning as easy weapon to create variants of software's. One of the studies reported that around 7–23% and 20–30% respectively are clones of a software module [10]. Another study shows that around 12.7% of the commercial software's are cloned codes [11]. Many approaches exist to find the clones they are text-based[13],[14], token- based[15], [16], [17], [18]), tree-based [15]; [19], graph- based [16], or deep learning techniques [17] that match the similar codes among same project or different projects. Because of the large code repositories online, programmers never start coding from scratch which they obviously find it way to save time and avoid tedious work[1],[18], we can also find evident in the [18] to say that 70% of the code in GITHUB are clones.

Only five existing papers propose the work of cross language clone detection where the first cross language clone detection was on .NET families where they share common intermediate language file after compilation [23] but the work also shows that preprocessing is applied intensively to eliminate the noise of intermediate file before applying the clone detection and also the work cannot detect clones of the code other that .NET family. Few other works used simple token matching techniques to find the clones but failed in accuracy of clone detection [11],[12]. The recent work Lawton Nichols et al [10] works on detecting functional clones or syntactic clones of same family clones that limits in scaling to large codebases. The survey shows that there exists no systematic approach to find cross language clone detection that can contribute significantly in software porting process.

All the above mentioned methods confined themselves to find cross language clones that have common intermediate language because of the parser that generate different parse tree even if two codes are similar at source level[9].

III. METHODOLOGY

This journal uses double-blind review process, which means that both the reviewer (s) and author (s) identities concealed from the reviewers, and vice versa, throughout the review process. All submitted manuscripts are reviewed by three reviewer one from India and rest two from overseas. We use LSA with neural association of all three languages keywords the Fig. 1 shows the architecture of cross language clone detector that includes following phases.

a) TOKENIZATION:

To find the commonalities among any two code documents we first run java tokenize to convert all the lines into tokens.

```
StringTokenizer st = new StringTokenier(data,"\r\n");
while(st.hasMoreTokens()){
code.add(st.nextToken().trim());
```

b) FIND EDIT DISTANCE BETWEEN 2 CODES:

We run edit distance algorithm to find the maximum similarities of tokens between source and destination code.

```
FOR i from 0 to s1.
      LENGTH
 set lv as i;
FOR j from 0 to s2. LENGTH
        IF (i EQUALS 0)
         set C[j] as j;
        ELSE IF (j > 0) {
            set  nv as C[j - 1];
            IF (s1.charAt(i - 1) != s2.charAt(j - 1))
                    nv = MINIMUM(nv, lv),
                 C[j] + 1;
                 set C[j - 1] as
            lv;
                 set lv as nv;
              }
        }
    IF (i > 0)
        set C[s2.length()] = lv;
    }
  return C[s2.length()];  }
```

c) LATENT SEMANTIC ANALYSIS

The maximum matched tokens among the source and destinations are given as input to latent semantic analysis tool with neural token association.

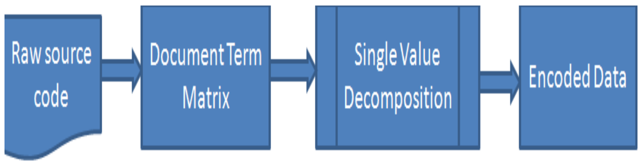


Fig. 2: Flow of Latent Semantic Analysis

IV. MATHEMATICAL DETAILS LATENT SEMANTIC ANALYSIS

The algorithm LSA flow is shown in Fig. 2 that is built with the idea of [31] that takes matrix M X N tokens by source files matrix were all the entries A(ij) is local frequency of a token i in some source file j then we convert local frequency to local weight matrix that gives token-source file co-occurrences in finding the functionality of the source code. In the second step singular value decomposition is applied to reduce the dimensionality of the weighted matrix. Finally the cosine similarities of reconstructed matrix shows the similarities among the two files in terms of percentage of line similarities

a) After finding local frequency of each token i , in file j. local weight matrix is found for each entry A(ij).

weight locij= log(freqlocij+ 1)

b) Entropy of token is calculated

$Weightglobi = 1 + \sum_{i=1}^{n} pij.\log(Pij)/\log(n)$

where *P(ij)* is the entropy of token i across all source file j.

$P(ij) = freqij\ loc\ /$

c) Find the local weight from local frequency which is

weight termij = weightloc ij/ Weightglob i

d) Perform dimension factorization and reduction using single value decomposition which is linear algebra of MX N matrix were all the entries are real numbers that can be decomposed into three matrices T, ∑, D^T

$$M = T \sum D^T$$

where *T* is M X M Matrix, D^T is N X N matrix with orthonormal columns and ∑ is M X N diagonal matrix

∑= D 0 0 0
0 0 0 0

e) Construct diagonal matrix in the decreasing order of the diagonal values and find inverse of this matrix for dimensionality reduction.,

$$Ms = T \sum_s D^T$$

f) Representations of tokens and source files can likewise be obtained by multiplying their corresponding decompositions by the reduced space singular value matrix $\sum_s$. Token similarity in s-space is given by $\sum_s$. and of source file is given by $D^T \sum_s D^T$

g) Finally similarity between 2 vectors *v1* & *v2* can be calculated by

cos(θ)= $v1.\ v2/|v1|.|v2|$

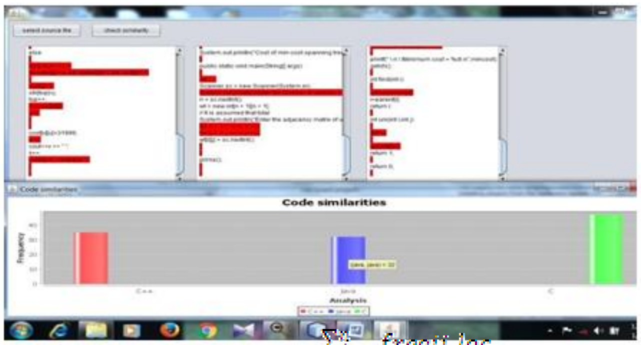
h) Token to token can be calculated by $M_s M_s^T$ and document similarity can be calculated by $M_s^T M_s$.

Singular Value Decomposition: It is the linear Algebra technique to reduce the dimensionality of the data being processed which includes following steps.

1. Find A^T and Multiply A and A^T
2. Find Eigen values for A.A^T
   a) A^T·A-C to get constants C1, C2
   b) Find s1= √C1 and s2= √C2
3. Construct diagonal matrix S by considering C1 & C2 values in descending order and find inverse of S
4. Replace C1 & C2 values in A^T·A-C. Compute Eigen value X1, X2 and place In Eigen vector V.
   a) V=[x1, x2]
   b) Find V^T
5. Compute U=AV S^{-1} later A can be decomposed by A=USV^T .

i) **Display** finally the percentage of the clones among input source file with the several files in a repositories are graphically displayed in terms of 4 different types of clones such astype1,2,3 & 4.

IV. RESULT AND DISCUSSION



$\sum_{j=1} freqij\ loc$

Fig. 3. C++ code match with similar Java & C code

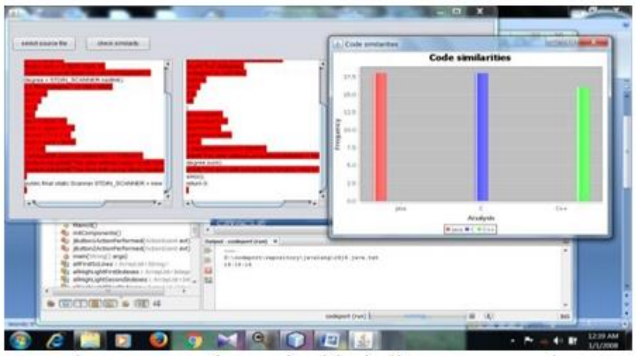


Fig. 4. Java code match with similar C & C++ code

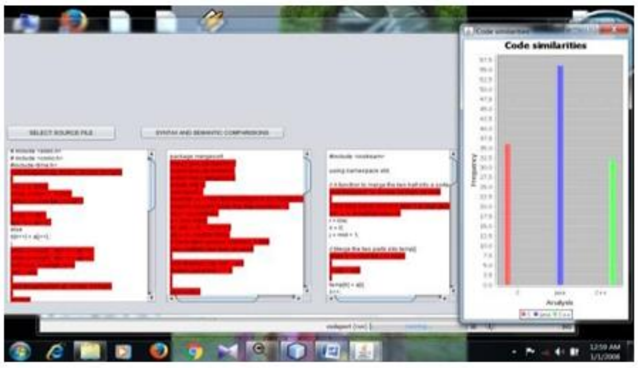


Fig. 5. C code match with similar Java & C++ code

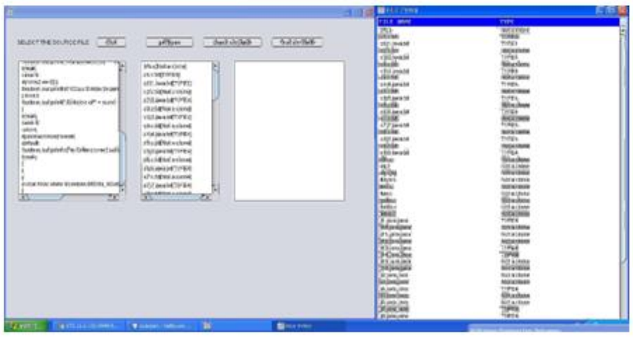


Fig. 6. Fetching all the clone matches in the repository given Java code

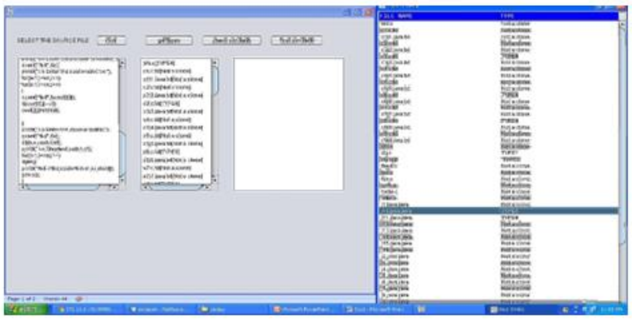


Fig. 7.matching clone types fgor file dij.c

Table-I: Showing Matching clones for Java, C & CPP files

Sl.No.	Code under test	Output files matched	Types of clones
1	c1j1.java	c1j1.java	Type1
		c1.c	Type3
		c2j2.java	Type4
		c3j3.java	Type2
		c4j4.java	Type4
		c5j5.java	Type3
2	dij.c	dij.c	Type1
		dij.cpp	Type2
		J10.java	Type4
		J11.java	Type4
3	mergesort.cpp	Mergesort.cpp	Type1
		Merge.c	Type4

Table-II: Showing Recall for Java, C & CPP files

Sl. No	Input files	No. of files matched	Recall
1	C (c1.c)	57	(52/57) 91.4%
2	CPP (Mergesort.cpp)	57	(57/57) 100%
3	JAVA(c2j2.java)	57	(43/57) 75%

V CONCLUSION

This research paper proposes the solution to find cross language clones of C, C++ and Java using primary NLP techniques such as tokenization, latent semantic analysis and classification. Method works with better precision on the repositories with less KLOC files.

The proposed work can be used to build the porting analyzer that helps to find following answers during code porting a) Bug fixes while porting the project from one language to other. b) Amount of code common among two different projects in terms of percentage of cloning and types of cloning. c) Common files among both the projects. d) Porting latency to convert project 1 to project2.

FUTURE ENHANCEMENT

The proposed work works well with projects containing Small files. Further improvements can be made in the accuracy in terms of precision and recall by using latent semantic indexing on even larger repositories.

REFERENCES

1. Chaiyong Ragkhitwetsagul, Jens Krinke Siamese: scalable and incremental code clone search via multiple code representations Empirical Software Engineering Springer Science+Business Media, LLC, part of Springer Nature 2019 https://doi.org/10.1007/s10664-019-09697-7
2. Nishi MA, Damevski K (2018) Scalable code clone detection and search based on adaptive prefix filtering. J Syst Softw 137:130–14.
3. Roy CK, Cordy JR (2008) NICAD: accurate detection of near-miss intentional clones using flexible pretty printing and code normalization. In: ICPC '08, pp 172–18.
4. Fowler M (1999) Refactoring: improving the design of existing code. Addison-Wesley, Boston.
5. Kapser C, Godfrey MW (2006) Cloning considered harmful considered harmful. In: Proceedings of the 13th Working Conference on Reverse Engineering (WCRE '06), Benevento, italy.
6. Aversano L, Cerulo L, Di Penta M (2007) How clones are maintained: an empirical study. In: Proceedingsof the 11th European conference on software maintenance and reengineering (CSMR '07), IEEE, Los Alamitos, California, USA, pp 81-90.
7. Juergens E, Deissenboeck F, Hummel B (2011) Code similarities beyond copy & paste. In: Proceedings of the 15th European conference on software maintenance and reengineering (CSMR '11), IEEE, pp 78–87.

8. Chatterji D, Carver JC, Kraft NA (2016) Code clones and developer behavior: results of two surveys of the clone research community. Empir Softw Eng 21(4):1476–1508.
9. Kamiya T, Kusumoto S, Inoue K (2002) CCFinder: a multilinguistic token-based code clone detection system for large scale source code. TSE 28(7):654–670.
10. Lawton Nichols et al Structural and Nominal Cross-Language. Clone Detection In: FASE 2019, LNCS 11424, pp. 247–263, 2019.
11. Harris S (2015) Simian – similarity analyser, version 2.4.http://www.harukizaemon.com/simian/, accessed:2016-02-14.
12. Prechelt L, Malpohl G, Philippsen M (2002) Finding plagiarisms among a set of programs with JPlag. J UnivComput Sci 8(11):1016– 1021.
13. Sajnani H, Saini V, Svajlenko J, Roy CK, Lopes CV (2016) SourcererCC: scaling code clone detection to big-code. In: ICSE'16, pp 1157–1168.
14. Schleimer S, Wilkerson DS, Aiken A (2003) Winnowing: local algorithms for document fingerprinting. In:SIGMOD '03, ACM, p 76.
15. Jiang L, Misherghi G, Su Z, Glondu S (2007) DECKARD: scalable And accurate tree-based detection of code clones. In: ICSE'07. IEEE, Minneapolis, pp 96-105.
16. Krinke J (2001) Identifying similar code with program dependence graphs. In: WCRE.
17. Li L, Feng H, Zhuang W, Meng N, Ryder B (2017) CCLearner: a deep learning-based clone detection approach. In: ICSME'17, pp 249–26.
18. Yang D, Martins P, Saini V, Lopes C (2017) Stack Overflow in Github: any snippets there? In: MSR '17.
19. Lopes CV, Maj P, Martins P, Saini V, Yang D, Zitny J, Sajnani H, Vitek J (2017) DejaVu: a map of code duplicates on GitHub. Proceedings of the ACM on Programming Languages (OOPSLA).
20. Foundations of statistical natural language processing By Christopher D.. Manning, Christopher D. Manning, Hinrich Schütze
21. *https://blog.aureusanalytics.com/blog/5-natural-language-processing-techniques-for-extracting-information*H. Poor, "A Hypertext History of Multiuser Dimensions," *MUDHistory*,http://www.ccs.neu.edu/home/pb/mud-history.html. 1986.
22. "https://en.wikipedia.org/wiki/Latent_semantic_analysisR. Nicole, "The Last Word on Decision Theory," *J. Computer Vision*, submitted for publication. (Pending publication)
23. AlOmari, F., Keivanloo, I., Roy, C.K., Rilling, J.: Detecting clones across Microsoft .NET programming languages. In: 19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, 15–18 October 2012, pp. 405–414 (2012). https://doi.org/10.1109/WCRE.2012.5
24. Kraft et al.: Cross-language clone detection. In: SEKE,pp. 54–59 (2008
25. Bellon et al : Comparison and eval-uation of clone detection tools. IEEE Trans. Softw. Eng. 33(9), 577–591 (2007)
26. Jiang et al : Deckard: scalable and accurate tree-based detection of code clones. In: Proceedings of the 29th International Conferenceon Software Engineering, pp. 96–105. IEEE Computer Society (2007)
27. Kamiya et al: a multilinguistic token- basedcode clone detection system for large scale source code. IEEE Trans. Softw. Eng.28(7), 654–670 (2002)
28. Rattan et al.: Software clone detection: a systematic review.Inf. Softw. Technol. 55(7), 1165–1199 (2013)
29. Rieger, M.: Effective clone detection without language barriers. Ph.D. thesis, Uni-versity of Bern (2005)
30. J. Cheng et al.: On the feasibility of detecting cross-platform code clones via identifier similarity. In: Proceedings of the 5th International Workshop on Software Mining, pp. 39–42. ACM (2016)
31. http://www.ling.ohio-tate.edu/~reidy/LSAtutorial.pdf
32. Sanjay Ankali, Latha Parthiban, I3Publication(2016)

AUTHORS PROFILE

1.Sanjay Ankali is Research scholar at VTU, RRC-Belagavi-590018 and working as Assistant Professor in department of CSE at KLECET, Chikodi, India-591201. His research interest is in the field of Software Engineering.

Dr. Latha Parthiban is working as Assistant Professor in department of Computer Science at Community College, Pondicherry University, India-605008. She received Bachelors of Engineering in Electronics from Madras University in the year 1994. M. E from Anna University in the year 2008 and Ph D from Pondicherry University in the year 2010. Her research interest is in Software Engineering, Big Data Analytics, and Computer Networking.