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 fixins 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 invention 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) {
  M=A+B;  //comment1
  N=A-B;  //comment1
} else {
  else
  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)
  { d1 = d1 + 1;  
    y1=x1+n1;  
} else
  y1=x1-m1
```

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

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

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 j , f=1;  
  for (j=1; j<n; j++)
    f= f * j; 
  return N *factorial(N-1) 
}  
```

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.

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

![Fig.1 Architecture Cross language clone detection](image)

**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 management 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.

```java
StringTokenizer st = new StringTokenizer(data,"\n\r\t\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.

```java
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] = MINIMUM(nv, lv),
    set C[j - 1] as
    lv;
    set lv as nv;
}
```
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+ ∑j=1 n pij.log (Pij)/log(n)

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

P(ij) = freqij / loci

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 MXN matrix were all the entries are real numbers that can be decomposed into three matrices T, ∑, D^T

M= T∑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

Σ=

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.

M= T∑Σ^−1 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 Σ. Token similarity in s-space is given by ∑Σ and of source file is given by D^TΣ ∑ 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,M^T and document similarity can be calculated by M^T,Ms

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

   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 as type 1, 2, 3 & 4.

IV. RESULT AND DISCUSSION

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

Fig. 4. Java code match with similar C & C++ code
Development of Cross Language Clone Detector for C, C++ & Java Repositories using Natural Language Processing

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 for file dij.c

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

<table>
<thead>
<tr>
<th>Sl.No.</th>
<th>Code under test</th>
<th>Output files matched</th>
<th>Types of clones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c1j1.java</td>
<td>c1j1.java</td>
<td>Type 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c1.c</td>
<td>Type 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c2j2.java</td>
<td>Type 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c3j3.java</td>
<td>Type 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c4j4.java</td>
<td>Type 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c5j5.java</td>
<td>Type 3</td>
</tr>
<tr>
<td>2</td>
<td>dij.c</td>
<td>dij.c</td>
<td>Type 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dij.cpp</td>
<td>Type 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J10.java</td>
<td>Type 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J11.java</td>
<td>Type 4</td>
</tr>
<tr>
<td>3</td>
<td>mergesort.cpp</td>
<td>Mergesort.cpp</td>
<td>Type 1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Input files</th>
<th>No. of files matched</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C (c1.c)</td>
<td>57</td>
<td>(52/57) 91.4%</td>
</tr>
<tr>
<td>2</td>
<td>CPP (Mergesort.cpp)</td>
<td>57</td>
<td>(57/57) 100%</td>
</tr>
<tr>
<td>3</td>
<td>JAVA(c2j2.java)</td>
<td>57</td>
<td>(43/57) 75%</td>
</tr>
</tbody>
</table>

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


20. Foundations of statistical natural language processing By Christopher D. Manning, Christopher D Manning, Hinrich Schütze


32. Sanjay Ankali, Latha Parthiban, I3Publication(2016)

AUTHORS PROFILE

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.