# FAST AND EFFICIENT SOURCE CODE PLAGIARISM IDENTIFICATION BASED ON TOKENIZATION

Sandeep kaur
Research Scholar
School of Computer Science and Engineering
Lovely Professional university, Phagwara(Punjab),India

***Abstract :*** When a code fragment is copied by another software system for reusing purpose, this is known as cloning. The code that is copied called clone[1]. Clone Detection is a process that is used for detecting clones in a software system. Code cloning or software cloning makes the work of the programmers easier by reusing existing code. But it increases the maintenance cost. If any error is occurred in a code fragment that is cloned, then that error have to be removed from all cloned modules. So the code clone detection is very indispensable part of software engineering. There are so many techniques have been proposed for code clone detection. These are like Text based, Metric based, Token based, AST-based, and PDG-based. All these methods have some limitations. Some techniques take more time for clone detection like PDG and AST and some need to improve the efficiency.

So we proposed a tokenization based method in which Needleman Wunsch algorithm will be used for comparison of two source files with token values. Needleman Wunsch is a global sequence alignment method that is used for two end to end sequence alignments. It is based upon dynamic programming and also an optimal matching algorithm. It takes very less time for comparing two sequences and also identifying sequences with gaps efficiently as compared to previous technique of our base paper [2] in which Smith Waterman algorithm have used.

***IndexTerms*** **- Clone Detection, Clone fragment, code clone, Token value, Duplicate code.**

## I INTRODUCTION

Code cloning is a process of reusing existed code, logic and design of a system [3]. There could be so many reasons behind it like merging two files to make a new system, complexity of code, time limit that is assigned to programmers or code may be copied accidently. Code cloning also has so many flaws as it increases maintenance cost, bug propagation and hard to grasped by maintainer [1]. So to find the duplicate fragments of code manually is very tedious and time consuming task. At present, so many code clone detection techniques have been developed. These can be Text-based, Token-based, AST or PDG based. All these techniques have their own benefits or weaknesses in terms of time and efficiency. In this paper, we have proposed a technique based on tokenization that replaced Smith Waterman sequence alignment algorithm with Needleman Wunsch algorithm. Needleman Wunsch is faster and more efficient sequence alignment algorithm than Smith Waterman [4].

## II PROBLEM FORMULATION

Code clone detection is major area of research for developers recently. So many techniques and tools have been proposed for clone detection. Some tools detects type1 (exact clone), type 2(renamed clone) and some detect all types including gap clone. All these techniques are based on Text-based, Metric-based, Token-based, PDG (Program dependency graph) based or AST (Abstract Syntax Tree) based techniques. Each of these tools has its own strengths and weaknesses according to their way of represented code, granularity and matching algorithms. There are so many matching algorithms have been proposed before like LCS, Suffix-

tree, Suffix-array, Smith waterman, PDG-based and AST-based etc.

The Smith Waterman algorithm is a tokenized based local Sequence alignment algorithm that detects the region of highest similarity including gaps between two sequence[2]. But this does not align the edges of similar sequences. The Smith Waterman makes more comparisons when matching the score to 0 and when searching the largest value score during the trace back [4].

To alleviate the above problems in Smith Waterman algorithm, the proposed technique in this research work is used tokenized based technique in which clones are detected by using Needleman Wunsch algorithm. Needleman Wunsch is a global sequence alignment method which performs end to end comparisons between two potentially equivalent sequences and identifies the equivalent alignments even if they contain some gaps.

**Comparisons of time and space complexity between Smith Waterman and Needleman Wunsch(NW)**

**Time complexity of Needleman Wunsch algorithm:** Following step by step approach is to be followed for analyzing time complexity:

1. Determine the first row and column: **O(r + s)**

2. Scoring in the table with all the values T (i,j) where i ranges from 1…r and j ranges from 1…s computing the scoring function : **O(r x s)**

3 The trace back : **O(r + s)**

4 Finding a final path, suppose s>r. so max(r, s): **O(s)**

This overall time complexity is

O(r + s) + O(rs) + O(r + s) + O(s)

When r and s are very large, the lower order terms are discarded and thus the total time complexity is in the **O(rs)** [4].

**Time complexity of Smith Waterman:** Same procedure is followed for analyzing complexity of SW:

1. Determine the first row and column: **O(r + s)**

2. Filling in the table : **O(r x s)**

3 The trace back takes: **O(r x s)**

4 Finding a final path, if suppose s > r this step can take max(r, s): **O(s)**

This overall time complexity is

O(r + s) + O(rs) + O(rs) + O(s)

When r and s are very large, the lower order terms are ignored , So the total time complexity is in the **O(2rs)** [4] .

Thus whole running time of Needleman Wunsch is lesser than that of Smith waterman [4].

**Space complexity of both:** Space consumed by both algorithms is equal O(rs). Because both use same matrix and same quantity of space is required for trace back [4].

**Accuracy:** Accuracy of Needleman Wunsch algorithm is better than Smith Waterman when the sequence being aligned might be quite diverse[4].

**III LITERATURE REVIEW**

In this section, we will discuss about the related work of code clone detection. These all papers [3][5][1][6] are survey papers of code clone detection. These gives us the very basic knowledge of code cloning, its types, clone detection phases and all-inclusive of recent techniques and tools. These papers represent the scenario based comparison of all techniques. Toshihiro kamiya et al.[7] has proposed a tool cc-finder which detects code clone in multiple languages by converting them into token sequences and compare them with suffix tree algorithm for extracting clones. Rainer Koshke et al.[8] has proposed a method for detecting syntactic clones using combined approach of tokenization and AST. Hamid Abdul Basit[9] gives us a clone detection tool RTF(Repeated Token Finder) .In this suffix array algorithm has been used. Warren Toomey[10] has proposed a tool named CT-compare. In which first source files are parsed into tokens and then separate into tuples of n successive tokens and hashed them for comparison. Yang Yuan et al.[11] have proposed a technique Boreas that is a scalable and accurate token based approach.

This tool uses Counting Based characteristics matrix for comparison. Rajnish Kumar[12] has proposed a method based on program slicing. Benjamin Hummel[13] has proposed a index based method for code clone detection. This is a real time detection technique that is scalable and incremental to very big code. Vera Wahler et al.[14] gives a technique in which source files are transformed into XML and then comparison is performed with frequent data mining technique. Zhenmin Li et al.[15] give us tool named CP-miner that uses frequent sub sequence mining algorithm for code clone detection. Hiroaki Murakami[2] has proposed a tool named CDSW in which Smith Waterman algorithm used for token value comparison. Abhilash CB [16]**:** This is a comparative study on global and local alignment. This study puts light on all concepts related to pair wise sequence alignment that are of further two types: Global sequence alignment algorithm like Needleman Wunsch algorithm and Local sequence alignment algorithm like Smith Waterman algorithm. Shakuntala Baichoo[4]**:** This paper is a review of sequence analysis method's time and space complexity and put light on their properties in a inclusive way[4]. Here, the memory capacity and time complexity of Needleman wunsch and Smith Waterman algorithm is compared in which Needleman wunsch algorithm proved better in time and both are equal in space complexity.

**NEEDLEMAN WUNSCH (NW) ALGORITHM**

The Needleman Wunsch(NW) algorithm[17] is a global sequence alignment method that is an example of dynamic programming. NW provides a mechanism of acquiring the best global alignment of two sequences including gaps by dividing the problem into several sub parts. There are following steps that to be followed by NW:

**I Initialized a Matrix:** First the initial matrix is created with M+2columns and N+2rows (where M & N are length of two sequences). Extra row and column is given, so as to align with gap at the starting of matrix as shown in below Table 1.

Suppose mismatch, match and gap are chosen as -2, 2, and -1 and the two sequences are

DLIMDIILD

DLMLDILD

**II Scoring the Matrix:** In second step, matrix filling is performed from the upper left hand

**Table1: Table initialization**

| - | - | D | L | I | M | D | I | I | L | D |
|---|---|---|---|---|---|---|---|---|---|---|
| - | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 |
| D | -1 | | | | | | | | | |
| L | -2 | | | | | | | | | |
| M | -3 | | | | | | | | | |
| L | -4 | | | | | | | | | |
| D | -5 | | | | | | | | | |
| I | -6 | | | | | | | | | |
| L | -7 | | | | | | | | | |
| D | -8 | | | | | | | | | |

Corner. Scores of each cell in matrix are calculated by using following formula:

$$M(i, j) = (Max(M_{i-1,j-1} + S(A_i, B_j)$$

$$M_{i-1, j} + gap, M_{i, j-1} + gap)$$

$$S(A_i, B_j)=\{ match (a_i = b_j) , mismatch (a_i != b_j)$$

In above example of sequences match= 2, mismatch= -2, gap= -1

**Table2: scoring matrix**

| | - | D | L | I | M | D | I | I | L | D |
|---|---|---|---|---|---|---|---|---|---|---|
| - | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 |
| D | -1 | 2 | 1 | 0 | -1 | -2 | -3 | -4 | -5 | -6 |
| L | -2 | 1 | 4 | 3 | 2 | 1 | 0 | -1 | -2 | -3 |
| M | -3 | 0 | 3 | 2 | 5 | 4 | 3 | 2 | 1 | 0 |
| L | -4 | -1 | 2 | 1 | 4 | 3 | 2 | 1 | 4 | 3 |
| D | -5 | -2 | 1 | 0 | 3 | 6 | 5 | 4 | 3 | 6 |
| I | -6 | -3 | 0 | 3 | 2 | 5 | 8 | 7 | 6 | 5 |
| L | -7 | -4 | -1 | 2 | 1 | 4 | 7 | 6 | 9 | 8 |
| D | -8 | -3 | -2 | 1 | 0 | 3 | 6 | 5 | 8 | 10 |

**III Trace back the matrix:** The diagonal movements during trace backing indicate a match and vertical and horizontal movements show gaps. The Table3 is example of trace back step of above example:

**Table3 Trace backing**

| - | - | D | L | I | M | D | I | I | L | D |
|---|---|---|---|---|---|---|---|---|---|---|
| - | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 |
| D | -1 | 2 | 1 | 0 | -1 | -2 | -3 | -4 | -5 | -6 |
| L | -2 | 1 | 4 | 3 | 2 | 1 | 0 | -1 | -2 | -3 |
| M | -3 | 0 | 3 | 2 | 5 | 4 | 3 | 2 | 1 | 0 |
| L | -4 | -1 | 2 | 1 | 4 | 3 | 2 | 1 | 4 | 3 |
| D | -5 | -2 | 1 | 0 | 3 | 6 | 5 | 4 | 3 | 6 |
| I | -6 | -3 | 0 | 3 | 2 | 5 | 8 | 7 | 6 | 5 |
| L | -7 | -4 | -1 | 2 | 1 | 4 | 7 | 6 | 9 | 8 |
| D | -8 | -3 | -2 | 1 | 0 | 3 | 6 | 5 | 8 | 10 |

**This analysis results the following sequence:**

D L I M – D I I L D
| | | | | | | |
D L – M L D I – L D

**IV PROPOSED METHOD**

The proposed method will be able to detect code clone of Type 1(exact clone), Type 2(Syntactic clone) and type 3(gapped clone). This method consists of following procedure:

**I. First, it takes following inputs**

a) Source files

b) Total Number of tokens

c) Maximum gap rate

d) Score parameters (match, mismatch, gap)

**II. Performing Parsing and Transformation**

**III. Statement hash**

**IV. Recognizing identical hash sequences**

**V. Discovering tokens with gaps**

**VI. Mapping to the original code**

The following Figure1 shows the procedure of proposed method.

**II. Performing Parsing and Transformations**

The input target files are parsed into token values. The special tokens are used instead of user defined identifiers. So that code clones can be find which have exactly not same variables.

**III. Statement hashing**

In statement hashing, every statement that is between ";", opening brace ("{") and closing brace ("}") has been assigned a hash value. A number of tokens in every statement are attached to its hash value.

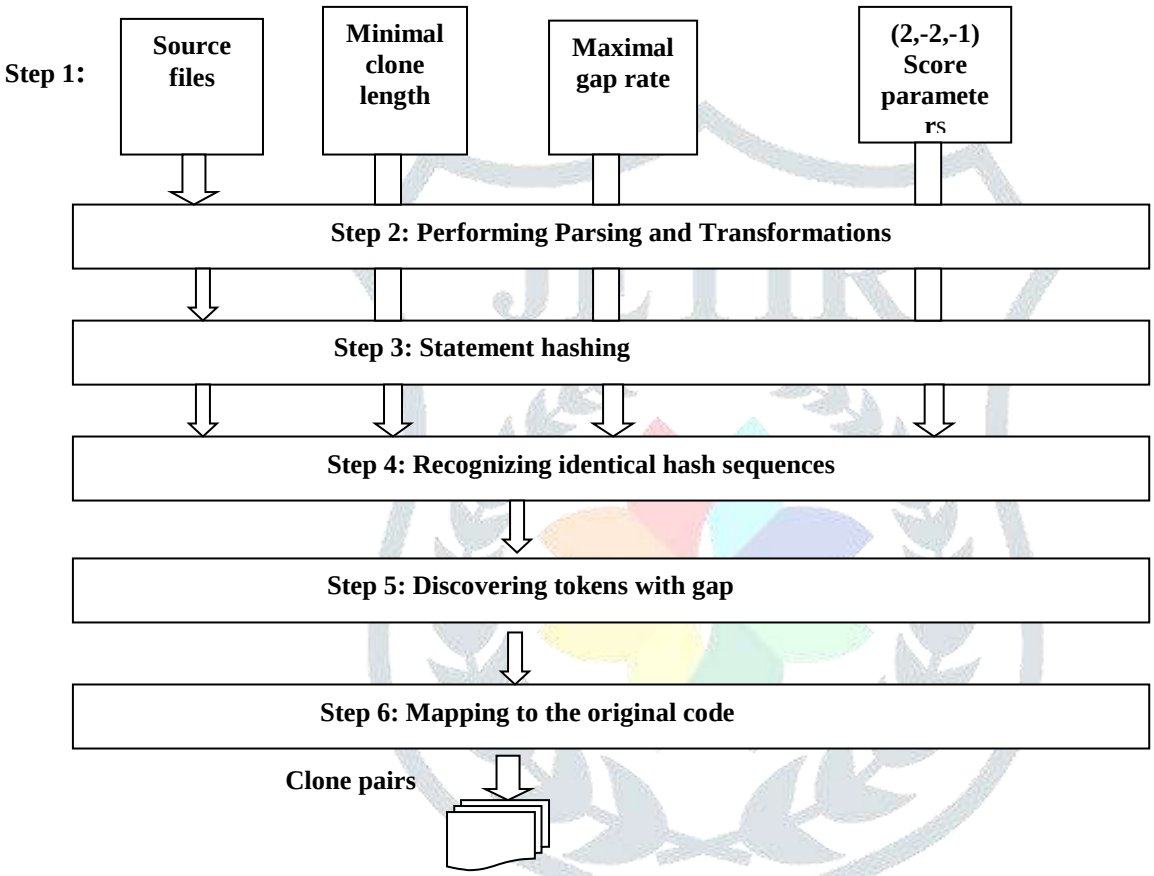


**Figure1 Paces of proposed method**

**IV. Recognizing identical hash sequences**

The Needleman Wunsch algorithm is used for recognizing similar hash sequences. Trace back is done from lower right corner towards upward by following maximal values and end at 0 value.

**V. Discovering tokens with gaps**

Discovering how many gaps in sequences

**VI. Mapping to the source code**

This step is used for mapping of code clone detected in step 4 and 5 into original source code.

**A EXAMPLE SHOWS PROPOSED METHOD**

First take following two source files code as input:

```
1 if(flag){
2 for( int m = 0; m< str . length; m++) {
3 buff. append(token[i]);}
4 str res = buff. tostring ( );}
```

```
1 strbuff buff = new stringbuffer( );
2 for(int m = 0; m < tok . length; m++) {
3 buff. append(tok[i]);}
4 buff. append(getcomma( ));
5 str res = buffer. tostring( );
```

After lexical analysis and normalization

```
1 if($){
2 for($$ = $ ; $<$.$ ; $++ ) {
3 $.$( $ [$]) ; )
4 $$= $.$( ) ; }
```

```
1 $$=new$( ) ;
2 for($$=$ ; $<$.$ ; $++){
3 $.$($[$]) ; }
4 $.$($( )) ;
5 $$=$. $( ) ;
```

Generating statement hash

```
1 0(4) {
2 10(6) ; 20(5) ; 30(4) {
3  40(9) ; }
4 50(8) ;
```

```
1 60(7) ;
2 10(6) ; 20(5) ; 30(4) ; {
3 40(9) ; }
4 65(8) ;
5 50(8) ;
```

**0 10 20 30 40 50**

**60 10 20 30 40 65 50**

Creating and initializing table and scoring

| _ | _ | 60 | 10 | 20 | 30 | 40 | 65 | 50 |
|---|---|---|---|---|---|---|---|---|
| _ | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 |
| 0 | -1 | | | | | | | |
| 10 | -2 | | | | | | | |
| 20 | -3 | | | | | | | |
| 30 | -4 | | | | | | | |
| 40 | -5 | | | | | | | |
| 50 | -6 | | | | | | | |

| _ | _ | 60 | 10 | 20 | 30 | 40 | 65 | 50 |
|---|---|---|---|---|---|---|---|---|
| _ | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 |
| 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 |
| 10 | -2 | -3 | 0 | -1 | -2 | -3 | -4 | -5 |
| 20 | -3 | -4 | -1 | 2 | 1 | 0 | -1 | -2 |
| 30 | -4 | -5 | -2 | 1 | 4 | 3 | 2 | 1 |
| 40 | -5 | -6 | -3 | 0 | 3 | 6 | 5 | 4 |
| 50 | -6 | -7 | -4 | -1 | 2 | 5 | 4 | 7 |

Trace backing

| _ | _ | 60 | 10 | 20 | 30 | 40 | 65 | 50 |
|---|---|---|---|---|---|---|---|---|
| _ | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 |
| 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 |
| 10 | -2 | -3 | **0** | -1 | -2 | -3 | -4 | -5 |
| 20 | -3 | -4 | -1 | **2** | 1 | 0 | -1 | -2 |
| 30 | -4 | -5 | -2 | 1 | **4** | 3 | 2 | 1 |
| 40 | -5 | -6 | -3 | 0 | 3 | **6** | **5** | 4 |
| 50 | -6 | -7 | -4 | -1 | 2 | 5 | 4 | **7** |

gap

| **0 10 20 30 40 - 50** | **60 10 20 30 40 65 50** |
|---|---|

Mapping to source code

```
1 if(flag){
2 for( int m = 0; m< str . length; m++) {
3 buff. append(token[i]);}
4 str res = buff. tostring ( );}
```

```
1 stringbuffer buffer = new stringbuffer( );
2 for(int m = 0; m < tok . length; m++) {
3 buff. append(tok[i]);}
4 buff. append(getcomma( ));
5 str res = buff. tostring( );
```

Gapped Tokens

**Figure2 Example of proposed method**

## V EXPERIMENTAL RESULTS

This proposed work is successfully completed with implementation of it. We have developed a tool of proposed work for detection of clones in two files of objected oriented language like java. The key outcomes of this proposed technique are as follows:

I. Proposed Technique is able to detect Type1, 2 and 3 of code clone.

II. Proposed Technique gives better result than base paper technique in precision, time, accuracy, recall and F-measure.

III. Tool of proposed technique is GUI based and very user friendly.

To prove our research that says our proposed work is able to detect type1, 2 and 3 clone very efficient and fast way. We have done trials with the base paper technique and proposed technique.

**Our proposed technique is fast**

First we will proved that our proposed work has taken very less time to calculate code clones than previous technique. We have been taken 9 pairs of different sample files of java and performed trials with both previous and proposed technique. The outcome reveals that our proposed method takes less time as shown in below Table4. Thus this result has proved our technique fast than previous technique.

**Table4: Time Comparison Result**

| Filenames (in pairs) | Time taken by Previous Technique(in seconds) | Time taken by Proposed Technique(in seconds) |
|---|---|---|
| Add.java<br>Add1.java | 1.03527 | 0.02479 |
| binarysearchByteArray.java<br>BinarySearchCharArray.java | 3.53880 | 0.04268 |
| CRC32-EXTRACTZIP.java<br>CRC-32extractzipfile.java | 40.19790 | 0.07422 |
| deflater-32.java<br>Deflater-compressbytearray.java | 31.06248 | 0.10045 |
| fact-for.java<br>fact-while.java | 1.09601 | 0.02909 |
| insertionsort.java<br>selection-sort.java | 8.05765 | 0.03285 |
| ResourceUtils.java<br>StdDocletSettings.java | 6.06230 | 0.05278 |
| MyBubbleSort.java<br>Sorting.java | 25.77521 | 0.04970 |
| My selection sort.java, mysort.java | 6.16941 | 0.05518 |

**Our Technique is efficient**

We have proved our proposed work efficient based upon the following parameters:

Precision

Recall

Accuracy

F-measure

First we will discuss that what are these parameters and how these are calculated. Calculation of these parameters is represented by the understanding of confusion matrix. Confusion matrix includes how many files are detect as clones and not clones[18][19][20]. The confusion matrix is shown as Table5.

**Table5: Confusion Matrix**

| | | True Label | |
|---|---|---|---|
| | | Positive | Negative |
| Categorize Label | Clones | True Positive (TP) | False Positive (FP) |
| | Not Clones | False Negative (FN) | True Negative (TN) |

In Table 5

**I True Positive (TP):** All files which are actually clones detected as clone by technique.

**II True Negative (TN):** The whole files which are not clone and also detected as not clone by technique.

**III False Positive (FP):** Files that are actually not clone but detected as clone by technique.

**IV False Negative (FN):** Files that are cloned files but detected as not clone.

By using above terms of confusion matrix, we will calculate following parameters:

**PRECISION:** Precision gives us number of results that are pertinent[18]. The formula of precision is

**Precision= TP/TP+FP**

**RECALL:** Recall is the number of true result detected from all possible outcomes of a technique[21]. The formula of recall is

**Recall= TP/TP+FN**

**ACCURACY:** Accuracy is the percentage of effectively identified files as clone.

**Accuracy=TP+TN/TP+FP+FN+TN**

**F-MEASURE:** F-measure is weighted harmonic mean of precision and recall[20].

**F-measure= 2*precision*recall / precision+recall**

We have taken total number of 30 sample files of java for testing or comparing proposed method with previous technique. From which 20 files are cloned files and 10 files are non cloned files. The confusion matrix of previous technique and proposed technique is as shown in below tables 6 and 7 respectively after performing trials with cloned and non cloned files.

**Table6: Results of Previous technique**

| | | True Label | |
|---|---|---|---|
| | | Positive | Negative |
| Categorize Label | Clones | 18 (TP) | 0 (FP) |
| | Not Clones | 2 (FN) | 10 (TN) |

**Table7: Results of Proposed technique**

| | | True Label | |
|---|---|---|---|
| | | Positive | Negative |
| Categorize Label | Clones | 20 (TP) | 0 (FP) |
| | Not Clones | 0 (FN) | 10 (TN) |

We have computed precision, recall, Accuracy and F-measure of both techniques by using above given formulas of them. The results are as given in table8 and figuree3 which proved our proposed method more efficient than previous.

**Table8: Comparison Results**

| Parameters to compare | Results of Previous Technique | Results of Proposed Technique |
|---|---|---|
| **Precision** | 1 | 1 |
| **Recall** | 0.9 | 1 |
| **Accuracy** | 0.94 | 1 |
| **F-measure** | 0.95 | 1 |

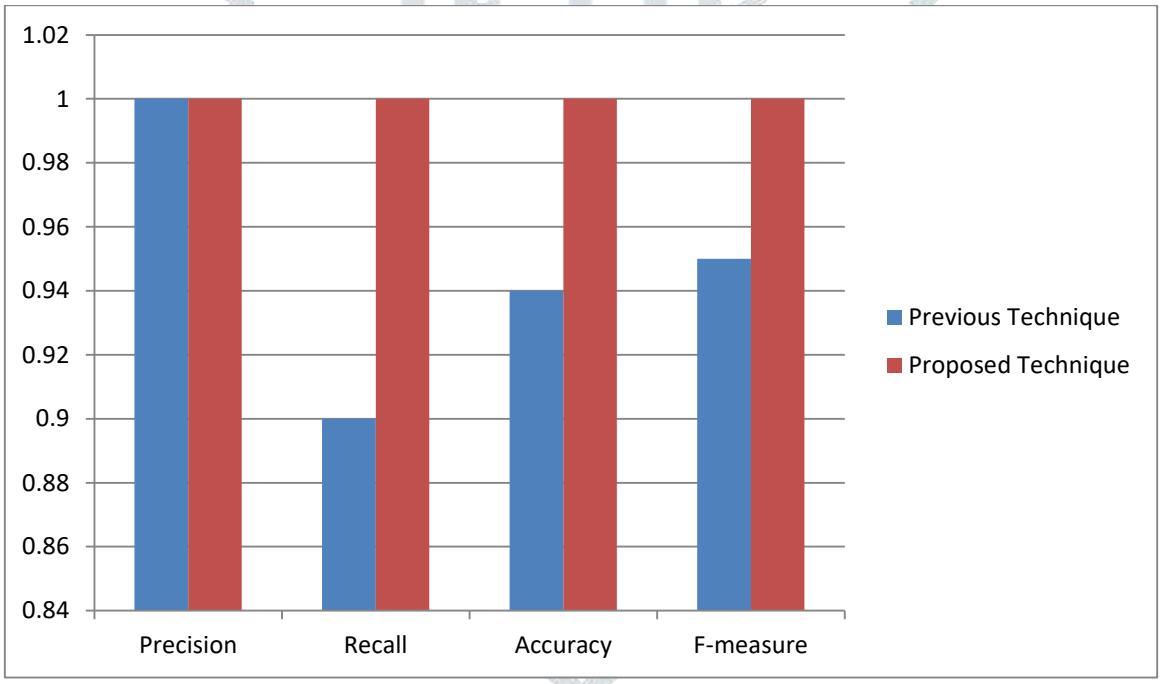


**Figure3 Comparison graph**

**CONCLUSION**

This paper has presented a tokenization based technique for code clone detection in which a DNA sequence alignment algorithm called Needleman Wunsch is to be used for comparing two file of object oriented languages like java after converting them into token values. The proposed method is a technique which detects code clone in fast and efficient way. Needleman Wunsch algorithm is used instead of Smith Waterman algorithm used in our base paper technique of code clone detection. Needleman Wunsch algorithm is better in speed and efficiency than Smith Waterman. It is a global sequence alignment method which performs end to end comparison and more accurate also.

We have also compared the results of proposed technique with base paper technique in which Smith Waterman algorithm have used. The whole results are based upon time, accuracy, precision, recall and F-measure parameters comparison that is represented with tables. This results show us how our proposed method is better than previous in time and efficiency.

This proposed technique is also able to detect Type1, 2 and 3 of code clone in fast and efficient way than previous.

## REFERENCES


[1] D. Rattan, R. Bhatia, and M. Singh, *Software clone detection: A systematic review*, vol. 55, no. 7. Elsevier B.V., 2013.

[2] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, "Gapped code clone detection with lightweight source code analysis," *IEEE Int. Conf. Progr. Compr.*, pp. 93–102, 2013.

[3] C. K. Roy and J. R. Cordy, "A Survey on Software Clone Detection Research," *Queen's Sch. Comput. TR*, vol. 115, p. 115, 2007.

[4] S. Baichoo and C. A. Ouzounis, "Computational complexity of algorithms for sequence comparison, short-read assembly and genome alignment," *BioSystems*, vol. 156–157, pp. 72–85, 2017.

[5] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. Comput. Program.*, vol. 74, pp. 470–495, 2009.

[6] A. Sheneamer and J. Kalita, "A Survey of Software Clone Detection Techniques," *Int. J. Comput. Appl.*, vol. 137, no. 10, pp. 975–8887, 2016.

[7] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. IEEE Trans Softw Eng," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 654–670, 2002.

[8] R. Koschke, R. Falke, and P. Frenzel, "Clone detection using abstract syntax suffix trees," *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 253–262, 2006.

[9] H. A. Basit, S. Jarzabek, and I. C. Society, "A Data Mining Approach for Detecting Higher-Level Clones in Software," vol. 35, no. 4, pp. 497–514, 2009.

[10] W. Toomey, "Ctcompare: Code clone detection using hashed token sequences," *2012 6th Int. Work. Softw. Clones, IWSC 2012 - Proc.*, pp. 92–93, 2012.

[11] Y. Yuan and Y. Guo, "Boreas: an accurate and scalable token-based approach to code clone detection," *Proc. 27th IEEE/ACM Int. Conf. Autom. Softw. Eng. - ASE 2012*, p. 286, 2012.

[12] R. Kumar, "Token based clone detection using program slicing," vol. 5, no. August, pp. 1537–1541, 2014.

[13] B. Hummel, E. Juergens, L. Heinemann, and M. Conradt, "Index-based code clone detection: incremental, distributed, scalable," *Softw. Maint. (ICSM), 2010 IEEE Int. Conf.*, pp. 1–9, 2010.

[14] V. Wahler, D. Seipel, J. Wolff, and G. Fischer, "Clone Detection in Source Code by Frequent Itemset Techniques."

[15] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner : Finding Copy-Paste and Related Bugs in Large-Scale Software Code," vol. 32, no. 3, pp. 176–192, 2006.

[16] A. C. B, "A Comparative Study on Global and Local Alignment Algorithm Methods," *Int. J. Emerg. Technol. Adv. Eng. Website www.ijetae.com ISO Certif. J.*, vol. 9001, no. 1, pp. 34–43, 2250.

[17] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–453, 1970.

[18] "Model Evaluation I_ Precision And Recall – Towards Data Science." .

[19] R. Joshi, "Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures." 2016.

[20] "How to calculate Precision, recall and F-measure in NLP relation extraction - Quora." .

[21] Koo Ping Shung, "Accuracy, Precision, Recall or F1? – Towards Data Science." 2018.