Review on Tokenization based Code Clone Detection

[1]Sandeep kaur, [2]Dhiraj kapila,
[1]Research Scholar, [2]Assistant Professor,
School of Computer Science and Engineering
Lovely Professional university, Phagwara(Punjab),India
Email – satgunsandhu@gmail.com

Abstract: *This paper will explain first the code clone detection brief introduction, terminologies, types and process of code clone detection. It will also explain the disadvantages and reasons of code cloning and advantages of code clone detection. Then it will give the detail review of tokenization based code clone detection, its different detection algorithms with examples and the tools which used these algorithms. At the end, the paper will conclude all techniques based on their advantages and weaknesses. So that, this would be helpful in future research.*

Keyword: *code clone; clone detection; code fragment;*

1. INTRODUCTION:

When a programmer or developer uses the existing software code (by minor or extensive editing) for his software development, this procedure is known cloning and the code piece that is copied or reuse is called clone of real code[1]. The code fragments which are similar in their variables, literals or functions, are called clone pairs[2]. More than one pairs make a clone class. In post development phase, it is difficult to identify which codes are clones. Only the similarity or equality of their contents proved them as clones. Several studies of software clone detection represent the software with code clones are burdensome to handle as compared to without code clone. The tendency of code cloning not only produce maintenance issues, But also produce subtle errors. Because, if any error is introduced in one code segment then the entire identical fragments have to be changed. That is very time consuming and costly process in big software projects. It is also increases protection cost. The open source software and its variation also enhanced code reuse. There is no clear definition exist exactly what is clone till now. There are so many definitions according to different authors. Clone is also known as a duplicate. But not every identical code is a clone. Sometimes, two code segments can be same by chance. They are not copied by each other. According to Baxter et al. “Clones are portions of code which are identical according to some definition of sameness” [3]. According to this definition similarity is to be based on lexical, text, syntax or semantic representation.
Software clone detection is appears as a vital research area. These studies are suggested that 20-30% of large software systems consist of cloned code. Kamiya et al.[4] has reported 29% cloned code in JDK .Code clone detection can be useful in clone maintenance , copyright infringement detection, plagiarism detection, code simplification and detection of bug report[1].

1.1. CLONE TERMINOLOGIES

Clone detection process output clones in the sort of clone pair, clone classes or both. Exactly what is the meaning of these terminologies in clone detection process? Now we discuss here:
Clone fragment (CF): A code fragment is a method or function or sequence of statements that are needed to run a program.
Clone pair (CP): when two fragments are similar in syntactically or semantically or both are clones of each other then called them clone pair[1].

Clone class (CC): Many code fragments that contain two or more clone pairs are called clone class

2. TYPES OF CODE CLONE:

Types of code clone available in software systems are based upon their similarity between textual representation and functionality. So we can categorize them into following four types. First three types like Type1, Type2 and Type3 are textually or syntactically similar and Type4 is similar in its functionality [2][3][5][1].
Type1 (Exact clone): Program fragments which are similar but slight different in white spaces and comments as shown in Table1

Table1 Type 1 clone

FRAGMENT 1	FRAGMENT 2

//printing
for(h=1;h<=4;h++)
	for(h=1;h<=4;h++)
	{

{
print h;}
	Print h;
	}

Type 2(Renamed clone): When two code fragments are syntactically identical but different in comments, layout, literals and identifiers. Reserved words and sentence formation are identical like real source code as shown in Table 2. We can see two code fragments are changed in their appearance, variables name and values.

Table2 Type 2 clone

FRAGMENT 1	FRAGMENT 2

if(m>= n){
q= n+m;
else
q= n-m;
	if(n>=o)
	z=n+o;
	else
	z=n-o;

Type 3(Near Miss clone or Gapped clone): In this type of clone, there is a insertion and deletion gap between similar statements within two code fragments as shown in below Table3

Table3 Type 3 clone

FRAGMENT1	FRAGMENT 2

If(k>l)
{
l++;
k=1;
}
	if(o>p)
	{
	o=o/2; //statement inserted
	o++;
	}

Type 4 (Semantic clone): Semantic code clones are two segments that are similar in their functionality or semantically not syntactically or textually. As shown in below Table4.

Table4 -Type 4 clone

FRAGMENT1	FRAGMENT 2

If(s>l)
{
u=s*l;}
else
u=s/l;
	switch(true)
	{
	case s>l:
	u=s*l;
	case s!=l;
	u=s/l;
	}

3. REASONS FOR CLONING:

There are so many reasons for copying the code by programmers. Some of the reasons have been listed below [3]

- A programmer reuses the existing code, logic and design of a system by copy and paste operations. It is called code cloning or duplication.
- When a programmer merges two similar systems to make a new system.
- Clones are frequently occurred in the financial products. Because companies do not wants to take risks of new product's high rate of errors. So they reuse the already existed well tested codes for adapting to the new product [6].
- Cloned fragments in the systems may improve maintenance. If all fragments will be independent, then we have to maintain them separately. This process will take more time.
- Complexity of code: Programmer sometime find hard to understand large and complex code so they just copy the code.
- Time limit: Time limit that is assigned to programmers to complete a product is very less. So they have to copy the existing clone.
- Accidently: Sometime unintentionally code may be copied by the programmers for the solution of similar types of problems.

4. WEAKNESSES OF CLONING:

Software cloning is widely used by all programmers. But it has so many adverse effects on software engineering. Some harmful effects of cloning are listed below [5] :

- **More maintenance cost:** Software cloning increases the efforts of maintenance by duplicating multiple fragments.
- **Bug propagation:** When an error is presented in a one code fragment that is copied at multiple places in a system. Thus the code cloning elevate to the bug propagation.
- **Difficult to understand by maintainer**: Because the maintainers have no information regarding duplicate fragments.

5. ADVANAGES OF CODE CLONE DETECTION:

As there are so many flaws in code clone, yet so many benefits are existed. Some of the advantages of software cloning are discussed below [3] :

- Helps in maintenance of software system by detecting code clones.
- Detects Library functions by detecting codes that are reused again and again.
- Detects plagiarism and copyright infringement.
- Helps in code compactness by reducing the size of source code.

6. STEPS OF CODE CLONE DETECTION:

The task of clone detector is to search code fragments which have high sameness in a software real code. The major problem is searching of code segments that are duplicated. So the detector has to compare every code segment with every other possible segment. Such activity to estimate the similarities and dissimilarities is very costly from computation view. So before performing the actual comparisons, there are so many calculations are used to lessen the domain of comparison. In this section, this report provides a net summary of fundamental paces in a clone detection procedure. The following figure 1 [2] shows the all paces that a classic clone detector may follow. These steps are [2][3]

Pre-processing: In this step, the source code is filtered by removing uninteresting parts and then subdivided and the field of the comparison is fixed.

Remove uninteresting source code fragment: In this step, the unwanted code which have no importance in comparison process are to be removed.

Fix comparison unit: In this pace, the Source code is to be separated into small scale units based upon the detection method. Source unit may be begin-end blocks, classes, files, functions or statements. This step determines the source units of code.

Transformation: Transformation in clone detection process is a step that used to transformed source code into a suitable middle delineation for comparison. This is also called extraction. This extraction transformed target code to the form appropriate for the resource to the actual clone detection algorithm. Depending upon the technique, it can have following steps:

Token or lexical form: Here, each line of target code is chopped up into token values according to the lexical guidelines of the programming language of heed. The whitespaces and comments are detached from the sequences of tokens. CC-Finder and Dup are the prime techniques that utilize this kind of tokenization.

AST (Abstract syntax tree) form: In this, the entire source code is converted or parsed into Abstract syntax tree. The source units to be compared are then shown as sub-trees of the AST and comparison algorithm look for similar sub-trees to mark as clones.

Program dependence graph (PDG) analysis: Semantic based methods induce program dependence graph from the source file. The vertices of a PDG act as a substitute for the lines and conditions of codes, edges indicate data and control dependencies. Source units are represented as sub-graphs of these PDG.

Normalization: This step is a voluntary pace knowing to remove specious difference such as removal of whitespaces, comments, formatting or structural transformation.

Clone Detection After transformation, code is then input into a comparison function. Then it is differentiate to other code fragments employing a comparison tool to identify equal code portions.

Formatting In this step of clone detection process, the clone code obtained as a result of previous step is converted into its original source code.

Filtering This step is not performed by all clone detectors. Here, code clone are took out and a human specialist refine out the false positives clones. This manual analysis process is to be defined based on length, range or frequency.

Aggregation While some tool results clone pairs not classes. Then to decrease the bulk of data, the clone pairs can be collected into mass, bunches or clone classes.

7. LITERATURE REVIEW:

The clone detection in coding has become a very major concept of research these days. So many techniques have been proposed for code clone detection by different researchers. In this section, the literature review about papers or topics related to software cloning will be given.

Chanchal kumar roy and James R.Cordy (2007)[3]**:** In this survey of code clone detection, the terms of clone commonly used in the literature like clone pairs, clone class and clone fragment are described. It is also discussed clone types which are commonly used. Second, this paper provides a review of detection techniques, different clone taxonomies and experimental evaluations of different tools of detections.

Roy, James and Rainer Koschke (2009)[2]**:**This paper gives us the very basic information regarding code cloning like clone types, clone detection steps and all-inclusive of recent techniques and tools. There are two different dimensions used for comparing, classifying and evaluating all tools and techniques. First, it divide and compare methods based on a number of aspects based on usages, interaction, language, clone ,technical, adjustment etc. and second a predictive scenario-based approach is followed that estimate maximal potential of each clone detection technique.

Dhavleesh Rattan et al.(2013)[5]**:** This survey is a systematic literature review of software clone detection that is based on 213 articles, 37 premier conferences and workshops. In this review, 9 different types of clone, model based and semantic clone detection description, 13 intermediate representations are reported.

Abdullah sheneamer, Jugal kalita (2016)[1]**:**This study gives us the benefits and flaws of all available clone detection techniques and tools by employing measurement based on precision, F-measure and recall metrics, scalability and portability. The goal of this study is to compare the recent status of the tools and techniques and highlight the future scope of them.

Toshihiro kamiya et al. (2002)[4]**:**In this paper, Kamiya et al. has proposed a tool named cc-finder which detects code clone in so many languages like java, c++, COBAL and other source files. It first transform the input source text files into token sequences and applies rule-based transformation to the sequence and then perform token by token comparison with suffix tree matching algorithm for extracting clones.

Rainer Koschke et. al(2006)[7]**:** In this paper, the author has proposed a method for finding syntactic clones in linear time and space by using combined approach of tokenization and AST(Abstract syntax tree). Here, first source files are parsed and formed AST. Then serialize AST and employ Suffix tree method for detection of syntactic clones.

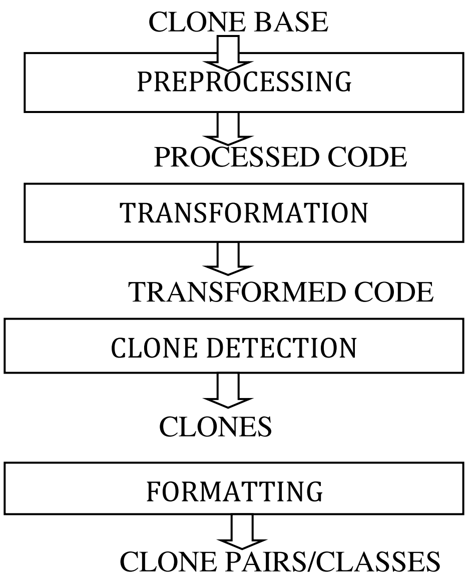

Figure 1. Basics phases of code clone detection

Hamid Abdul Basit(2007)[8]**:** In this research, author has proposed a clone detection tool called RTF(Repeated Tokens Finder). The working of this tool is as:

1) First the source program is transformed into a string of token values. 2). A suffix array based algorithm is used for computing clones in the string of tokens 3 Then, to get rid of the probable false positive, the pruning that is based upon a heuristic is used [8].

Warren Toomey(2012)[9]**:** The author has developed a tool called CT-Compare. It uses a noval tokenization approach. Each source code file is first parsed into tokens with lexical analyzer and then divided into tuples of N successive tokens. The tuples are then hashed and the hashed sequences are used to detect type 1 and type 2 clone pairs[9].

Yang Yuan and Yao Guo [10]**:**In this report, Boreas, a scalable and accurate token based approach for detecting code clone[10]. Here, variables are used instead of matching sequences or structures. Using Counting-Based characteristics matrix, the similarity of two code fragments is identified by using the characteristics of proportion variables.

Rajnish kumar [11]**:** This research paper has proposed a clone detection technique using program slicing with a token based matching algorithm. This method detects type1, 2 and type 3 code clone and also best for detecting non contiguous code clone.

Benjamin Hummel et al. [12]**:** The author has proposed a index based method for detection of exact and renamed clones. This method is both scalable and incremental to very big code base and deal with real-time detection in large system.

Kodhai et al.(2014) [13]**:**In this paper, author has proposed a clone manager named tool based upon metrics and textual analysis for detection of procedure level semantic and syntactic clones in c and java projects.

Vera Wahler et al.[14]**:**In this paper, first the source files from java, c++ or prolog are parsed into XML and then frequent data mining technique is applied for extraction of clones.

Zhenmin Li et al(2006)[15]**:**The author gives us a tool called CP-miner. It uses "clospan" frequent sub sequence mining algorithm for code clone detection. The procedure of this tool as:

1) First parse the source code into tokens. 2) Then do mining for basic duplicate segments. 3) Then prune false positive. 4) Then compose larger copy-pasted segments.

Hamid Abdul Basit(2009)[16]**:**In this method, the author has represented a technique that is used for structural or higher level clone detection with the application of data mining techniques.

Hiroaki Murakami(2016)[17]**:**In this dissertation, the author has proposed two clone detection techniques that upgrade the existing weaknesses. The first technique is a token based technique that folds every repeated instruction for reducing uninteresting clones and then apply token based detection techniques. This tool is called FRISC. Second technique detects gapped clone applying a local sequence alignment method named Smith Waterman (SW). This algorithm is faster and more accurate than previous algorithms like LCS, suffix-tree, suffix-array, PDG-based etc.

Hiroaki Murakami(2013)[18]**:**In this paper, author has proposed a tool named CDSW. This tokenized based tool uses the smith waterman algorithm for clone detection. This is very efficient algorithm and takes less time as compared to previous algorithms.

8. TOKENIZATION BASED CODE CLONE DETECTION:

Today, so many code clone identification methods are available. Some are based upon string based methods or some others are token based, PDG or AST based. Here, we will discuss token based code clone detection in detail. A token based approach in which the whole source code is parsed to a series of token values. Then the examination of sequence is to be performed for detecting duplicated subsequence of tokens and ultimately converts the resulted code clone into original code portions. As compared to text based approach, it is more durable averse to code variations such as formation and arrangement. There are so many algorithms or methods are available for token values comparison of two source files to detect clone in code fragments. Here we will explain some very famous or commonly used algorithms used in tokenization based code clone detection. These are:

a) Suffix tree algorithm

b) Suffix array algorithm

c) Frequent itemset mining techniques

d)Smith waterman algorithm

a) SUFFIX TREE ALGORITHM: In this, first a suffix tree is build of the text after preprocessing of the text. Then we can search any pattern from this suffix tree in o(m) time where m is length of pattern that is very lesser than entire text length. So it takes very less time as compared to so many other existing algorithms like KMP and Rabin Karp etc. Here, we will understand the suffix tree structure with following example:

Suppose we have a string "BANANA" , the following figure shows how to make the suffix tree of BANANA.

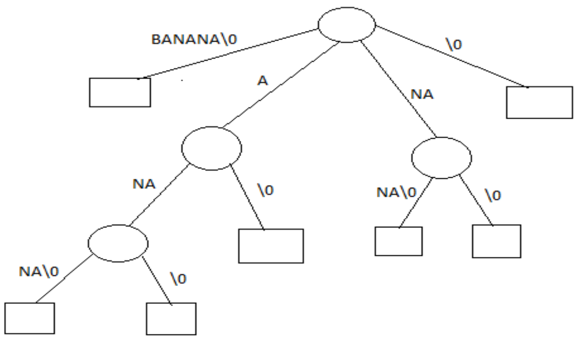

Figure2: suffix tree of BANANA string

Now, question is how to find a pattern in the build suffix tree. There are following steps:

1. First take the first character of pattern and compare it with the root of suffix tree. Then follow below steps for

every character:

a) If any edge is existed from the current node of suffix tree for the current character of pattern then follow it.
b) Otherwise print "Pattern not existed "and return.

2. If all characters of pattern are matched and there is path from root then print "Pattern Found".

b) SUFFIX ARRAY ALGORITHM: When all suffixes of a given word are converted into a sorted array is called a suffix array. A suffix array is made from a suffix tree by using DFS traversal of suffix tree. A algorithm that is based upon suffix tree can be replaced with suffix array. Both suffix tree and suffix array have same time complexity, but suffix array takes less space as compared to suffix tree. For example, we can convert a suffix tree in figure 2 into suffix array as shown in below figure:

0 BANANA		5 A
1 ANANA		3 ANA
2 NANA	sorted string -	1 ANANA
3 ANA		0 BANANA
4 NA		4 NA
5 A		2 NANA

c) FREQUENT ITEMSET MINING TECHNIQUES: Frequent item set mining is used for discovering recurring relationships in a given database. It helps us to find a pattern that is frequently occurred in a dataset. So this type of algorithms has been used in code clone detection to find frequent occurring code fragments. Apriori algorithm is very famous frequent itemset mining method which is employed for searching frequent pattern by using prior knowledge of frequent itemset properties. It uses an iterative method also called a level wise search where k-itemsets are used to explore k+1 itemsets. Following example shows the whole processing steps of Apriori algorithm:

Suppose we have a transactional data for a company as shown in Figure4.

Tid	Item list
1	A1,A2,A5
2	A2,A4
3	A2,A3
4	A1,A2,A4
5	A1,A3
6	A2,A3
7	A1,A3
8	A1,A2,A3,A5
9	A1,A2,A3

Figure4. Transaction database

C1

itemset	Sup.Count
{A1}	6
{A2}	7
{A3}	6
{A4}	2
{A5}	2

L2

itemset	Sup.count
{A1}	6
{A2}	7
{A3}	6
{A4}	2
{A5}	2

C2: generate from L1

Itemset	Sup.count
{A1,A2}	4
{A1,A3}	4
{A1,A4}	1
{A1,A5}	2
{A2,A3}	4
{A2,A4}	2
{A2,A5}	2
{A3,A4}	0
{A3,A5}	1
{A4,A5}	0

L2: after scanning database

Itemset	Sup.count
{A1,A2}	4
{A1,A3}	4
{A1,A5}	2
{A2,A3}	4
{A2,A4}	2
{A2,A5}	2

And so on.

Figure5. Apriori algorithm for generating frequent and candidate item sets with minimum support count 2

d) SMITH WATERMAN ALGORITHM: Smith waterman is a local alignment method that is used for identifying similar alignments between two given sequences. There are following steps which are followed to perform sequence alignment:

Step1: first create a table

Step2: Then initialize the table by filling first row and column with two base sequences and second row and column initialize to zero.

Step3: Now perform scoring of all cells by using following formula:

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

$$S(A_i, B_j)=\{ \text{ match } (a_i = b_j) \quad ,\text{mismatch } (a_i \neq b_j)\}$$

Where match, mismatch and gap have some fixed value.[18]

Step4: Perform tracebacking from maximum score cell towards the zero value cell.

Step5: Identifying similar alignments.

Thus all above algorithms have been used for code clone Detection with slight changes time to time for improving the efficiency and accuracy of code clone detection. The Table5 shows us the different code clone detection techniques based on tokenization with these above detection algorithms by different authors:

Table5: Tokenization based code clone detection tools and detection algorithms.

Tool name/ auther name	Algorithm For comparison
Dup	Suffix tree
CC-Finder	Suffix tree
RTF by Basit	Suffix array
CP-miner	Frequent itemset mining(FIM)
FC-Finder	LSH(Locality Sensitive Hashing)
Jian-lin	Suffix array
Rainer Koschke et al.	AST,Suffix tree
Ctcompare	Hashing
Hiroaki Murakami	Smith Waterman

9. CONCLUSION AND FUTURE SCOPE:

Thus this review gives us the basic knowledge of code clone detection, its types, its different phases for smoothly detecting duplicate fragments in software source code, disadvantages, reasons for cloning and advantages of code clone detection in brief. It will also help the future readers in understanding tokenization based code clone detection by providing detail explanation of its comparison algorithms with examples and tools by different authors in form of figure. Even tokenization based detection is very popular method for clone detection, but it has also so many weaknesses. It is based on the order of program lines. If order is changed in duplicated fragment then code will not be detected. This approach cannot detect code clone with swapped lines or even added or removed tokens.

These techniques are very expensive in time and space complexity. So in future, we can make changes existing detection algorithms to improve the efficiency and speed of code clone detection. There are so many algorithms are available for frequent itemset mining and for sequence alignments in different fields which can be used for code clone detection in better way than previous.

REFERENCES:

1. A. Sheneamer and J. Kalita, “A Survey of Software Clone Detection Techniques,” *Int. J. Comput. Appl.*, vol. 137, no. 10, pp. 975–8887, 2016.
2. 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.
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. 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.
5. D. Rattan, R. Bhatia, and M. Singh, *Software clone detection: A systematic review*, vol. 55, no. 7. Elsevier

B.V., 2013.

6. J. R. Cordy, “Comprehending Reality: Practical Challenges to Software Maintenance Automation,” *Int. Work. Progr. Compr.*, pp. 196–206, 2003.
7. R. Koschke, R. Falke, and P. Frenzel, “Clone detection using abstract syntax suffix trees,” *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 253–262, 2006.
8. H. A. Basit, S. J. Puglisi, W. F. Smyth, A. Turpin, and S. Jarzabek, “Efficient token based clone detection with flexible tokenization,” *6th Jt. Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng. companion Pap. - ESEC-FSE companion ’07*, p. 513, 2007.
9. W. Toomey, “Ctcompare: Code clone detection using hashed token sequences,” *2012 6th Int. Work. Softw. Clones, IWSC 2012 - Proc.*, pp. 92–93, 2012.
10. 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.
11. R. Kumar, “Token based clone detection using program slicing,” vol. 5, no. August, pp. 1537–1541, 2014.
12. 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.
13. E. Kodhai and S. Kanmani, “Method-level code clone detection through LWH (Light Weight Hybrid) approach,” *J. Softw. Eng. Res. Dev.*, vol. 2, no. 1, p. 12, 2014.
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. 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.
17. I. Science, “Fast and Precise Token-Based Code Clone Detection January 2016 Hiroaki Murakami.”
18. 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.