Multi-Agent based Sequence Algorithm for Detecting Plagiarism and Clones in Java Source Code using Abstract Syntax Tree

D. Poongodi
Research Scholar
Manonmaniam Sundaranar University
Tirunelveli, Tamilnadu

G.Tholkkappia Arasu
Principal
AVS Engineering College
Ammapet, Salem, Tamilnadu

ABSTRACT
Plagiarism and clone detection plays an important role in software security protection, software maintenance and license issues. Source-code similarity detection method can be classified as string-based, token-based, parse-tree-based and program-dependency-based. All of these approaches have certain limitations and can not meet the requirements when the source code is large and may produce false positives. But, parse-tree based detection improves the detection ability and efficiency. This paper describes method and statement based source code similarity detection, which detects the simple plagiarized code like exact match, near exact match and longest common sequence using multi-agent based detection which will perform the detection automatically. Automatic plagiarism detection will be helpful for code clone detection in software industry and plagiarism detection in projects.

Keywords
Abstract syntax tree, plagiarism detection, source code plagiarism detection, parse tree, code clone.

1. INTRODUCTION
This Research investigates the application of plagiarism technology that derives avoidance of duplicate files using Artificial Intelligence and Abstract Syntax Tree. The performance of this technology depends on parameters that can influence its performance and this research aims to investigate its performance using the time as major parameter. This technology has Sequence Algorithm which is a similar source code detection when the parameters are optimized will be evaluated. A technique for searching duplicate source codes using hierarchical technology is proposed in order to find the plagiarism and clone detection is proposed. This tool also will improve the plagiarism and clone detection with effective investigation process.

This proposed work is used to avoid the source-code plagiarism and clone increases in the source code of program due to the plenty of resources available in the electronic form. The easy access to the internet has also been increased. Manual detection of source code similarity is not very easy and it is time consuming due to the vast amount of contents available. As the amount of programming code created is increasing, different techniques are available to detect plagiarism in source code.

The proposed system is designed using Abstract Syntax tree and multi-agent for source code similarity detection in java source code with the help of sequence algorithm. As a result, the framework detects the source code similarity in java source code based on method level and statement level using the Multi-Agent. It is useful for the IT industry to detect cloning from one version to the next version and also to find plagiarism in source code of the projects.

2. LITERATURE SURVEY
People can gather source code using various technologies available such as internet, text books, sources including large database available in electronic form etc.,

According to the Study of source code similarity detection approaches and its algorithms [1], some of the Detection algorithms are classified based on the approach and comparison method. They are sequence, finger print, hashing, suffix tree and so on. Each algorithm has produced some false positives.

At Present, People are aware of so many algorithms using various technologies for detecting duplications that are not giving complete results, and it also have so many disadvantages, which are mentioned below.

1. Sequence algorithm [2], [9] detects sub-tree clones and it is used essentially to detect statement and declaration sequence clones. In this algorithm Clone removal is not carried out.

2. Finger Print Algorithm[3] compares nm sub-trees of m projects of size n (in terms of nodes) for exact equality detection that would require $O((nm)^2)$ comparisons with a native approach, all sub-trees are rather fingerprinted and put in buckets according to their hash value. This algorithm takes much time to compare.

3. In mapping algorithm [4], traverse the ASTs of the function bodies of old and new versions in parallel, adding entries to a LocalNameMap and GlobalNameMap to form mapping between local variable names and global variable names respectively. LocalNameMap will help to detect functions which are identical up to a renaming of local and formal variables, and GlobalNameMap is used to detect renaming of global variables and functions. This algorithm cannot proceed without mapping.

4. Hashing algorithm [5], [10] makes the main idea of the algorithm to compute certain characteristic vectors to approximate structural information within ASTs and then adapt Locality Sensitive Hashing (LSH) to efficiently cluster similar vectors. This algorithm will not work efficiently without LSH.

5. In Greedy String Tiled Algorithm [6] the whole parse tree need not be converted to a string. An intermediate stage in algorithm could transform the original parse tree into a "degenerate" parse-tree by removing nodes.

6. The pattern matching algorithm [7] will process in the format of statement, if the statement order has minor change this algorithm will not detect the duplicate codes.

7. In the suffix tree algorithm [8], it will convert the tree into string and compare the source code. This algorithm is not the statement structure based comparison.

So the various algorithms that were discussed could not meet as per the user expectation. Hence the implementing proposed Algorithm user expectation can be met and the false positives can be reduced in order to increase the response time by reducing its comparison time.

3. PROBLEM STATEMENT

In the current literature review, Source-code similarity detection algorithms can be classified based on the following:

- String Based Detection
- Token Based Detection
- Parse Tree(AST) Based Detection
- PDG (Program Dependency Graph) Based Detection
- Metric Based Detection
- Hybrid Based Detection

All of the above said detection methods have certain limitations and cannot meet the requirements when the source code is large and may produce false positives. The Abstract syntax tree based linear representations is efficient than the other comparison algorithms, because it uses the structure of the source code for comparison. Multi-Agent system is a computerized system composed of multiple interacting intelligent agents within an environment. Multi-agent systems can also be used to solve problems that are difficult or impossible for an individual agent to solve. The main purpose of the research work is to detect the source code similarity to reduce the software maintenance in IT industry in the form of cloning and to increase the software security in the form of plagiarism done by the user. This detection technique is easy and efficient using the intelligent agent and parallel detection with the help of multi-agent system.

Though there are so many technologies used to develop the algorithm for finding the duplication method, it is not successfully processing the need of user requirement. The existing method works for minimum task given by the user, which is not giving the user to get the result as step by step comparison result nor the statement level comparison method, the same can be done using this method for plagiarism and clone detection in the source code.

4. OBJECTIVES OF THE RESEARCH

The objective of the research work is to design the architecture which will detect the source code similarities available in the form of plagiarism and clone. Automatic plagiarism detection will be helpful for the multi-tasking and automatic code clone detection which helps the software industry to detect the clones parallel with development.

Scope of the Work

- This component helps the user to detect the plagiarism in the java source code, either by using in method level or in the statement level.
- Various comparisons can be done simultaneously using this tool and at the same time, it can be executed / compared in different levels.
- This component can use to detect the clone in the java source code, either using in the method level or in the statement level.
- The sequence algorithm used in this component, increase the response time and takes the minimum comparison time in the order of n. (O(n)).
- This Algorithm also reduces the false positive in both plagiarism and clone detection.
- Detection of code that gives the same result, promises decreased software maintenance cost corresponding to the reduction in code size.
- If the plagiarisms are detected, then the code will not get any problem to get copyrights.
- This component is developed based on multi agent system, which uses agents with their own actions and behaviors. The main characteristic is to control their own behavior and interact with the environments and other agents. Some properties of agents are
    i. The agents are able to decide on their own without the human or other interventions.
    ii. The agents perceive their environments and respond the change that occurs.
    iii. The agent has initiative and they do not act in response to their environment.

5. PROPOSED WORK

5.1 Multi Agent based Sequence Algorithm for Plagiarism and Clone Detection

In tree-based approach, a program is parsed to a parse tree or an Abstract Syntax Tree (AST) with a parser of the language of interest. Similar sub trees are then searched in the tree with the proposed tree matching technique and the corresponding source code of the similar sub trees are returned as plagiarism classes.

The programming languages are defined by their grammars, which describe the set of all possible strings that represent programs (called a language). During the compilation process, a compiler builds a parse tree which represents the program and uses this tree to guide compilation.

Traverse the parse tree of different parts of source code to identify the plagiarism between the programs. Steps of algorithm are given below:

1. Parse the source code into a AST using AST Parser

2. ***Compare the Parse trees, based on the methods as follows***

    a) Count the number of child nodes that matches for both the methods.

b) If the number of child nodes matches with two different methods and at the same time if it is greater than or equal to three then do the comparison

c) If both node matches then find the number of child.

d) Find the threshold value using the following formula.

$$\text{Ratio}=\frac{nm}{\text{Min}(\sum m1,\sum m2)!}\ x\ \frac{\text{Min}(\sum_{i=0}^{nm} nmc(m1),\sum_{i=0}^{nm} nmc(m2))}{\text{Max}(\sum_{i=0}^{nm} nmc(m1),\sum_{i=0}^{nm} nmc(m2))}$$

Where nm is number of node matches in between method1 and method2.

Where nmc is number of children count for the node matched.

Ratio of threshold can be configured with 0.75, 0.9 or any value greater than 0.5

Compare the tree, based on statements as follows

a. Count the number of child node matches between two different Statements.

b. Find the number of child nodes which matches for both the statements.

c. Find the threshold value using the following formula.

$$\text{Ratio}=\frac{nm}{\text{Min}(\sum sl1,\sum sl2)!}\ x\ \frac{\text{Min}(\sum_{i=0}^{nm} nmc(sl1),\sum_{i=0}^{nm} nmc(sl2))}{\text{Max}(\sum_{i=0}^{nm} nmc(sl1),\sum_{i=0}^{nm} nmc(sl2))}$$

Where nm is number of node matches in between statement list1 and statement list2.

Where nmc is number of child count for the node matched.

Ratio of threshold can be configured as 0.75 , 0.9 or any value greater than 0.5

5.1.1 Method of Comparison

The proposed approach of comparison is different from the existing algorithms. After parsing the source code into parse tree, if the comparison is method level, then it compares using the following method

1. Collecting all the methods and its child node up to leaf node.
2. Count the number of nodes in each method
3. Based on the number of nodes compare with other source code, if the count difference is less than or equal to 3 then do the node matching as follows
    a. Take the first node or statement from the given list of code1
    b. Compare to the first node or statement of the other list of code
    c. If both the nodes matching then compare the next statement in both the list of code, else compare the first node of list1 to the second node of list2 until to find the matching node or compare with all other nodes in the list.
    d. While matching continuously (that is 3 nodes are matched continuously), if the next node is not matching, then that node will be compared from the first node including the node which is matched.
    e. Steps c and d will repeat until the end of both the list or all nodes are compared.
4. Depending upon the number of child node matches, the threshold value can be calculated using ratio formula.
5. If the threshold value is between 0.1 and 0.9 then there is a similarity between both the codes.

Statement level comparison is as follows

1. Collecting all the statements from different source codes.
2. Take the first statement node from the given source code and compare the same with similar program in first statement node.
3. If both the statements are equal or match with each other then take the second node of both the source code, if it matches then compare the next as same until there is a match.
4. In the continuous matching, if found a mismatch then start comparing the similar program list mismatch node with the first statement node of the given source code.
5. If first statement node is not matched, take second node and compare with the mismatch node.
6. If matched then next node of both the source codes will compare as mention in step 3.
7. Matching nodes are stored in file and this report is the output of the algorithm
8. This algorithm finds the exact match and near exact match like longest common sequence.

For Example,

i) *Exact match or no change Comparison*.

The exact match codes are as follows

int i;	int k;
int j;	int m;
for(i=0;i<10;i++)	for(k=0;k<10;k++)
for (j=0;j<10;j++)	for (m=0;m<10;m++)
System.out.println(i+j);	System.out.println(k+ m);

Table1. Example for Exact Match Source Code

In the above source code both the list are same, the only difference is that the identifier name is changed. This type of plagiarism or clone is exact match or no change.

The proposed algorithm compares the given source code with similar type of program and it is represented in the below given diagram for the exact match. First it takes the statement node1 from both the parsed source code and compares. If it is matches then it compares the next node of both the lists. The process continues until it reaches at the end of both the parsed source code

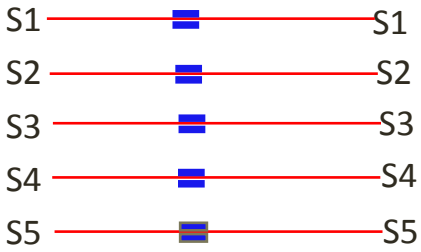

Figure1. Comparison Method of Exact Match Source Code

Finally, it gives the report as text/html file which contains matched node in the given source code and the similar program source code.

ii) *Near exact match or Longest Common Sequence (LCS) Comparison*

"Near exact match" is like copying part of the source code from others and adding own code or including unnecessary codes. If the plagiarizer includes some code then the source code might look different from the original code.

Some of the plagiarizer may divide the copied code and paste in different manner without affecting the final result of the source code. That is by changing the order of the program like first line as third or fourth line, fourth line as first or second. Example for near exact match as follows

i=f=1;
for(i=1;i<=n;i++)
f=f*i;
System.out.println ("Factorial");
System.out.println(f);
	System.out.println("Factorial");
	i=f=1;
	for(i=1;i<=n;i++)
	f=f*i;
	i=s=1;
	for(i=1;i<=n;i++);
	System.out.println(f);

Table2. Example for Near Exact Match Source Code

In the above source code, first and second line is repeated and the fourth line is pasted as first line of other program. This example code contains the "Near exact match" and the "Longest common sequence". Comparison of this kind of source code is mentioned below.

1. Compare the first node with the all other nodes until there is a match.
2. Once matched then compare the next node which is given in source code and similar program source code.
3. If mismatch occurred, it has to start comparing from the first node until matches. If first node does not match then second node will be compared until the match occurs.
4. Once matched then repeat step2 and step3 until source code ends.

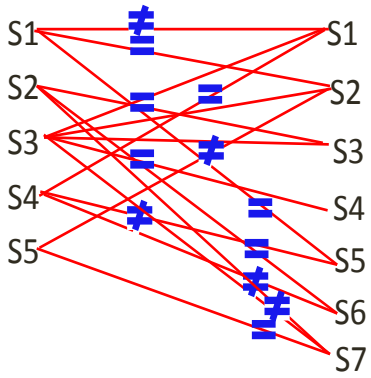

Figure2. Comparison Method of Near Exact Match Source Code

Finally, it gives the report as matched node as longest common sequence and repeated node in the given source code and similar program source code as text/html file.

This algorithm reduces the time of comparison and detect the maximum possible plagiarized or cloned code in the given source code and the similar source code of various programs.

5.2 Flow Diagram of the Tool

The flow diagram explains the flow of execution. Once the tool is started, first it starts the JADE (Java Agent Development Environment), then it starts the Generic Agent. From the Generic Agent, Generic GUI will be started. Then the user has to select the type of source code detection and give the input files depends on the type. Once input is received then it will translate the source code into the AST using Java parser. If the detection type is plagiarism, collects the list of relevant logic source and translates into the AST.

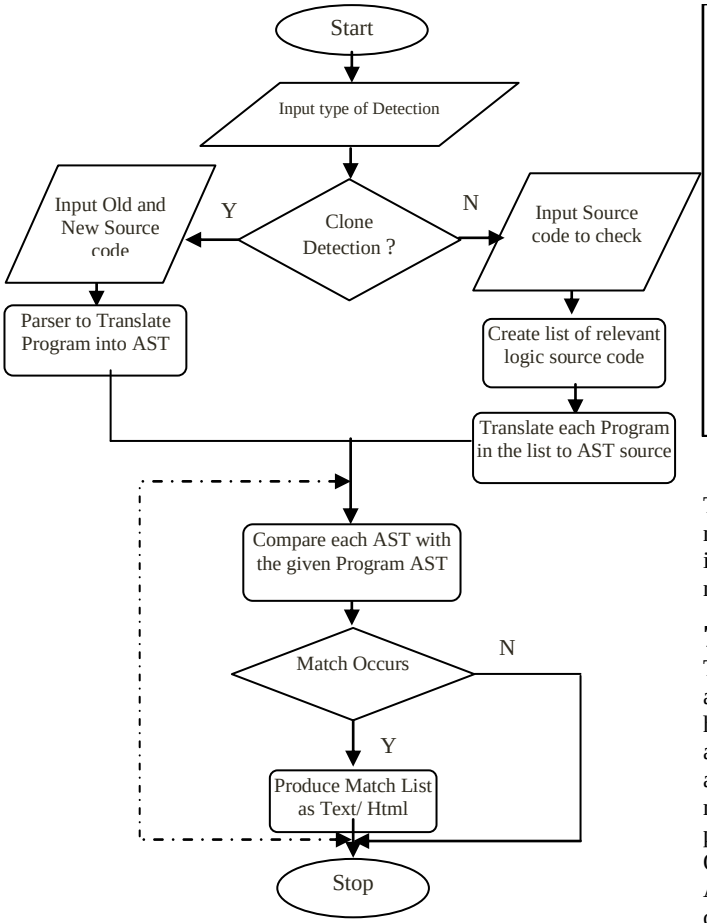

Figure3. Flow Sequence of Plagiarism and Clone Detection Tool

After Translating the AST's, Compare using the sequence algorithm. If match occurs within the threshold it will produces the match list as text or html file or else it gives the message as "No Match" and stops the Agent.

6. RFORMANCE EVALUATION

The proposed system is based on multi-agent system using Abstract Syntax tree. It is implemented with the help of JADE framework and Eclipse.

6.1 Evaluation

To evaluate the effectiveness of proposed algorithm, various similar programs are collected and compared. Once the source code is converted into the parse file, comparison process is easy for this algorithm approach. Java was used to parse the source code into abstract syntax tree. Each statement of source codes is converted into AST based node and each node contains full information about the statement. Then the number of node matches is found based on program level or method level of the source code. The output file will generate a report about the statement or method matches in various similar programs.

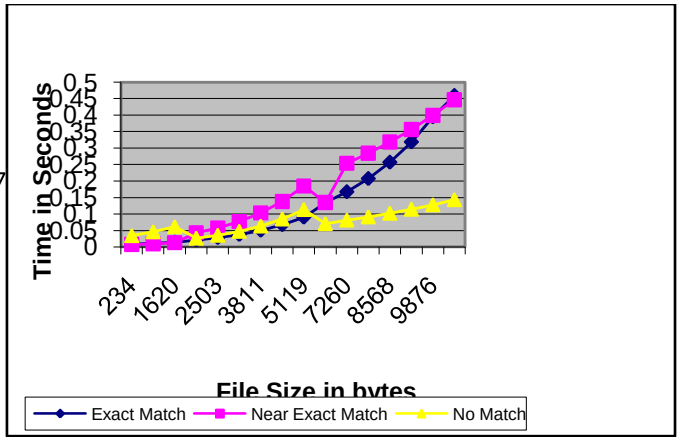

Figure4. Performance Evaluation of Algorithm

The above chart displays the response time for exact match, near exact match and no match based on the file size. If there is a similarity in the source code, it takes more time than the not similar source code.

7. CONCLUSION AND FUTURE WORK

Today, Plagiarism and clone detection in source code is an active research area. In this research work it was presented how the plagiarism detection can be handled using the new algorithm based on the Abstract Syntax Tree. The proposed algorithm reduces the time of comparison. This might take minimum order of N (O(n)) comparison time to detect the plagiarism in source code. It is developed using agent Oriented Programming, which also reduces the man power. Agent can control their own behaviors, actions and communicate with other agents. The component is based on multi-agent system, so it is helpful to control their own behavior and interact with the environment and other agents. This study may help the plagiarism and clone detection users to detect the similarity of the source code.

This proposed approach support only for the java based source code and the same approach may be used to compare with cross programming language, which is language independent comparison.

This algorithm helps to detect the plagiarism and cloning in source code in an effective manner. However, still some of the algorithms lacking to avoid the false positives. In future these algorithms may be improved to avoid false positives and detect all type of plagiarism to affect success plagiarism detection using AST. So it may enrich to avoid false positive with efficient manner.

General approaches are used like Meta data to find the similar program or logic of source code. In future this can also be found the similar logic of source code without using Meta data.

8. REFERENCES

[1] Roy, ChanchalKumar;Cordy, James R.."*A Survey on Software Clone Detection Research*". School of Computing , Queen's University, Canada. Vol 115, TR2007-541, September 26, 2007.

[2] Baxter,I.D, Yahin,A.; Moura, L.; Sant'Anna,M; Bier, L. "Clone detection using abstract syntax trees", International conference on software maintenance 1998, 598, ISBN:0-8186-8779-7.

[3] Michel Chilowicz, Etienne Duris and Gilles Roussel"Syntax tree fingerprinting: a foundation for

source code similarity detection", 17th IEEE International Conference on Program Comprehension (ICPC'09). Vancouver, BC, Canada. May 2009. pp. 243–247. IEEE Computer Society.

[4] IulianNeamtiu;Jeffrey S. Foster;Michael Hicks "Understanding Source Code Evolution Using Abstract Syntax Tree Matching" MSR '05 , Volume 30 Issue 4, Pages 1-5, ISBN:1-59593-123-6, July 2005.

[5] Lingxiao Jiang Ghassan andSt´ephaneGlondu*"DECKARD: Scalable and Accurate Tree-based Detection of Code Clones"* 29th International Conference on Software Engineering 2007, 96-105, May 2007.

[6] Matt G. Ellis, Claude W. Anderson *"Plagiarism Detection in Computer Code", http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.8027&rep=rep1&type=pdf ,* March 23, 2005.

[7] William S. Evans Christopher W. Fraser Fei Ma "Clone Detection via Structural Abstraction", Journal of Software QualityControl, Vol 17, Issue 4, 309-330, Dec 2009.

[8] Rainer Koschke, RaimarFalke, Pierre Frenzel "Clone Detection Using Abstract Syntax Suffix Trees" 13th Working Conference on Reverse Engineering (WCRE 2006), 253-262, ISBN:0-7695-2719-1, October 2006.

[9] Kevin Greenan, *"Method-Level Code Clone Detection on Transformed Abstract Syntax Trees Using Sequence Matching Algorithms"*University of California - Santa Cruz , 2005

[10] Baojiang Cui, Jun Guan, Tao Guo, Lifang Han, Jianxin Wang and Yupeng J *"Code Syntax-Comparison Algorithm based on Type-Redefinition-Preprocessing and Rehash Classification" , Journal of Multimedia*, Vol 6, No 4 (2011), 320-328, Aug 2011

[11] Young-Chul Kim and Jaeyoung Choi *"A Program Plagiarism Evaluation System"*, ICCSA 2005 http://link.springer.com/bookseries/558Volume 3483, 2005.

[12] A.S. Bin-Habtoor and M.A.Zaher, "A Survey on Plagiarism Detection Systems", International Journal of Computer Theory and Engineering, Vol 4. No.2, April 2012.

[13] Christian Arwin and S.M.M.Tahaghoghi, "Plagiarism Detection across Programming Lanugages", ACSC'06, Vol. 48, 277-286 , 2006, ACM.

[14] Tahira Khatoon, Priyansha Singh and Shikha shukla, "Abstract Syntax Tree Based Clone Detection for Java Projects", IOSR '12, Vol.2, 45-47, Issue 12, Dec 2012.