COMPARATIVE STUDY OF EXISTING PLAGIARISM DETECTION SYSTEMS AND DESIGN OF A NEW WEB BASED SYSTEM

By

NEERAJA JOSHUA SAMUEL

Bachelor of Commerce

University of Delhi

New Delhi, India

1985

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2006

COMPARATIVE STUDY OF EXISTING PLAGIARISM

DETECTION SYSTEMS AND DESIGN OF A

NEW WEB-BASED SYSTEM

Thesis Approved:

Dr. Istvan Jonyer

Thesis Adviser

Dr. Johnson Thomas

Dr. Debao Chen

Dr. A. Gordon Emslie

Dean of the Graduate College

Acknowledgment

I would like to express my sincere thanks to my advisor Dr. Istvan Jonyer for his advice, guidance and support in this research and throughout my master's studies. I am also thankful to Dr. Johnson Thomas and Dr. Debao Chen for being on my committee to examine my thesis. Further, I would like to thank my husband, Joshua Samuel, and my children, Akshay and Anisha, for their encouragement and support during this time. Finally, I would like to express my gratitude to my parents for the inspiration and support they provided me in all my endeavors.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

II. RELATED WORK 4

2.1 MOSS 4
2.2 YAP 5
2.3 SID 6
2.4 JPlag 7
2.5 DFGGI 8

III. WEB-BASED PLAGIARISM DETECTION USING GRAMMAR-BASED FINGERPRINTING 10

3.1 Overview 10
3.2 Parser 12
3.3 SubdueGL 13
3.4 Submatch 18
3.5 Web Interface 20
3.5.1 Interactive HTML Form for File Submission 21
3.5.2 CGI Script 22
3.5.3 Shell Script 22

IV. A COMPARITIVE STUDY OF PLAGIARISM DETECTION SYSTEMS 24

4.1 Methodology 24
4.2 Results and Analysis of Tests Using Real World data 26
4.3 Results and Analysis of Additional Tests 39

V. SUMMARY AND CONCLUSION 46

REFERENCES 49

APPENDIXES 50
APPENDIX A--CGI SCRIPT 51
APPENDIX B--SHELL SCRIPT 54

LIST OF TABLES

Table Page

1. Features of existing plagiarism detection systems..9

2. Assignment 1 – with –exhaust option...27

3. Assignment 1 – without –exhaust option..28

4. Assignment 1 - comparison of results...29

5. Assignment 2 – with –exhaust option...32

6. Assignment 2 – without –exhaust option..32

7. Assignment 2 - comparison of results...33

8. Assignment 3 34

9. Assignment 3 - comparison of results...34

10. Assignment 4 39

11. Assignment 4 - comparison of results...39

12. Comparison of results with modification to file 9.c 43

13. Comparison of the system developed under this research with existing plagiarism systems...48

LIST OF FIGURES

Figure Page

1. Information flow in our system....11
2. Graph file generated by the parser....13
3. Graph of graph file presented in Figure 2....13
4. Input graph....15
5. First production by SubdueGL....16
6. Input graph after first production by SubdueGL....17
7. Second and third production rule by SubdueGL....17
8. Input graph after second and third productions by SubdueGL....18
9. Comparison of two finger prints....20
10. File submission form....21
11. Email sent to user....23
12. Example of code comparison between files 11 and 17....36
13. Example of sample sub-structures for files 11 and 17....37
14. Code identified as Match in SID but not in JPlag....41
15. Some minor modification made to 9.c to test the results in JPlag....42
16. Code identified as Match in JPlag but not in SID....44

CHAPTER I

INTRODUCTION

The problem of determining similarity between two documents has been a topic of ongoing research in the field of computing. However, in recent years it has become more critical to have a reliable, accurate and easy to use system, to determine if two documents are similar and the extent of their similarity. Both professional and academic plagiarism is easier to commit in the information age. In the academic world students share their work and collaborate with other students through email or copy material from published works or the internet without proper citation. In computer programming classes, the students can easily share their source code for programming assignments.

Determining similarity between two documents, with particular reference to plagiarism detection, has been addressed in the past, and several systems have been proposed and published. Some of the systems that are being used to find plagiarism are MOSS, YAP, SID and JPlag. These systems use different algorithms to detect plagiarism among source code files. Generally, depending upon the algorithms used, a system may have inherent weaknesses that can be exploited. In this research we conducted an extensive study with an objective to compare and contrast the existing plagiarism detection systems and identify their strengths and weaknesses. As a part of this study several experiments were conducted and the results obtained from these experiments are discussed in this thesis.

In addition to this comparative study, a new web based plagiarism detection system, with an easy to use interface, was also developed. This system is an enhancement of a previously developed system named DFGGI. This system was also evaluated along with other existing plagiarism detection systems mentioned above in the comparative study.

The hypothesis of this research is that the comparative study will reveal inherent weaknesses that the existing plagiarism detection systems might have, which can be exploited to circumvent them. The attempt in the new system is to see if using a graph grammar data mining technique can produce a more reliable system that can overcome some of the weaknesses discovered in the existing systems. The system developed in this research uses a graph based data-mining algorithm for generating graph grammars and document fingerprints. The data mining algorithm used in this solution is SubdueGL. The fingerprints generated by SubdueGL are compared to determine similarities and detect plagiarism.

The system developed in this research gives the user a simple form on the internet, where they can fill in the contact information and submit the source code files compressed into a single zip file. The source files can be written in C, C++, Java and FORTRAN programming languages. The system acknowledges submission of the compressed file as soon as the user submits the file. After the file is processed, the results are communicated to the user via email.

This work is organized as follows. Chapter II gives an overview of the related work. It gives a description of the existing plagiarism detection systems. Chapter III discusses the new web based plagiarism detection system and its components. Chapter IV

gives a detailed comparison of the existing plagiarism detection systems. Chapter V describes the conclusions of this research along with some recommendations for future work.

CHAPTER II

RELATED WORK

This chapter describes currently available systems for detecting plagiarism in software source code. Table 1, at the end of this chapter, highlights some of the main features of the plagiarism systems studied in this research.

2.1 MOSS

MOSS (Schleimer, Wilkerson, and Aiken, 2002) stands for "Measure Of Software Similarity" and was developed by Alex Aiken I at UC Berkeley in 1994. It is accessible on the Internet at http://theory.stanford.edu/~aiken/moss/. MOSS employs a document fingerprinting technique to detect textual similarity. It first extracts significant words or phrases from the documents under scrutiny, by applying whitespace sensitivity and noise suppression. This is done by ignoring noise data such as comments, whitespaces, capitalization and punctuation marks. Noise suppression also removes short or common words that are likely to complicate the comparison, such as "the", "a". Whitespace characters are hidden control characters, such as blanks, tabs, newline, carriage-return. Whitespace sensitivity and noise suppression leaves the strings that are used for comparison unaffected.

After the documents are clean of noise, MOSS combines all text in the document together and divides them into small sub-strings, or k-grams. The length of k-gram is the number of alphabets in each sub-string and is individually defined by each user. Next an

index number representing each sub-string is added to each document using a hashing function. Finally, the sequences of index numbers of the two documents are compared to find similarity between the two documents.

To use MOSS, a Perl script needs to be downloaded by the user. This script is used for submitting files to the server and for displaying some server responses at command prompt. MOSS displays results via a web page. Files are submitted by listing them as parameters to the command line. After submission of the files is complete, MOSS gives a response at the command prompt giving the URL of the web page where results can be viewed. MOSS presents the results as a list of ordered pairs with the matching percentage of each file in the pair and the number of lines matched. The matched results are followed by a list of errors that the program encountered during processing. For both measures, higher numbers mean more overlap and a higher probability of plagiarism.

2.2 YAP

YAP (Wise 1996), which stands for Yet Another Plague, tries to find a maximal set of common contiguous substrings to detect plagiarism. It has three different versions - YAP1, YAP2 and YAP3. All three versions of YAP work as follows. In the first phase, source texts are used to generate token sequences. This phase involves several operations, such as, removal of comments and string-constants, translation from upper-case letters to lower case, mapping of synonyms to a common form, reordering the function into their calling order, and removal of all tokens that are not from the lexicon of the target language.

In the second phase, which is a comparison phase, different versions of YAP use different algorithms. The original version of YAP is based on the UNIX utility “sdiff”.

YAP2, which was implemented in Perl, uses Heckel's algorithm. Yap3 is the latest version in YAP series, and uses an algorithm called Running Karp-Rabin, Greedy String Tiling (RKR_GST).

2.3 SID

SID (Chen et al., 2004), which stands for Shared Information Distance or Software Integrity Detection, detects similarity between programs by computing the shared information between them. It was originally an algorithm developed for comparing how similar or dissimilar genomes are. It was later extended to other applications like finding plagiarism. SID finds plagiarism by computing the amount of shared information between two programs as follows:

$$D(x,y) = 1 - \frac{K(x) - K(x|y)}{K(xy)}$$

where K(x|y) is the Kolmogorov complexity of x given y. However, since the Kolmogorov complexity is not computable, SID uses a compression algorithm to approximate Kolmogorov complexity.

SID also works in two phases. The first phase involves parsing the source programs to generate tokens. In the second phase an algorithm named TokenCompress is used which computes heuristically the shared information metric D(x,y) between each program pair submitted. Then, all the program pairs are ranked by their similarity distances. SID can detect plagiarism in source code written in Java and C/C++.

SID is a web based service. Users can submit the files in a compressed (zip) format. The user has to make separate zip files for source code files written in different programming languages. After processing the files, SID sends an email to the users to

inform them that the results are ready to be viewed on the internet. Users need to log in to the SID web site to see the results.

2.4 JPlag

JPlag (Prechelt, Malpohl, and Phlippsen, 2000) can find plagiarism in source code written in Java, C, C++ and Scheme. JPlag, also, works in two phases. In the first phase programs to be compared are parsed, depending on the input language and converted into token strings. In the second phase, these token strings are compared in pairs for determining the similarity of each pair. During each such comparison, JPlag attempts to cover one token stream with substrings ("tiles") taken from the other as well as possible. The percentage of the token streams that can be covered is the similarity value.

The matching step (phase 2) consists of two more phases. In phase 1, the strings are searched for biggest contiguous matches using three nested loops. The first one iterates over all the tokens in the first string. These nested loops collect the set of all longest common substrings. The second one compares this token with every token in the second string. If they are identical, the innermost loop tries to extend the match as far as possible. In phase 2, all matches of maximal length found in phase 1 are marked. This means that all the tokens are marked and thus may not be used for further matches in phase 1 of subsequent iteration. The two phases of the matching step are repeated until no further matches are found or a lower bound for length, called "Minimum Match Length" is met.

JPlag requires download of a Java program to the client. It requires the Java virtual machine (runtime environment) to be present for the client application to work. The application is a Java applet that provides a Graphical User Interface (GUI), which allows

the users to browse their file system to submit the files. JPlag supports files written in C, C++ and Java.

2.5 Document Fingerprinting Using Graph Grammar Induction (DFGGI)

Document Fingerprinting Using Graph Grammar Induction (DFGGI) (Apiratikul 2004) uses a graph-based data mining technique to find fingerprints in the source code. The system first converts the source code to a linear graph based on the textual relationship. It then applies the SubdueGL data mining algorithm to find graph grammars, which are the fingerprints of the source file. Finally, it compares the fingerprints to detect similarities and returns the percentage of similarity between two source files.

Table 1: Features of existing plagiarism detection systems

	MOSS	JPlag	SID	YAP
Algorithms Used	Fingerprinting technique	Greedy String Tiling	Computes shared amount of information using Kolmogorov's complexity	Running-Karp_Rabin Greedy-String-Tiling
Language Supported	[1]C, C++, Java. C#, Fortran, etc.	C, C++, Java. C#, scheme	C, C++, Java	Pascal, C, LISP
Platforms for Client	Unix	Java program. Requires Java run time applicable to the platform	Web-based	Unix
File Submission Mode	Command line	Java application	Web form	Command line
File format for submission	Files as parameter to Perl executable	Files by specifying folder in UI	Zip file	By specifying folder as parameter to Perl executable
Language Specification	Required	Required	Required	Use appropriate tokenizer
Results	Email with URL	HTML page	SID site	Written to file

[1] Python, Visual Basic, JavaScript, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2.

CHAPTER III

WEB-BASED PLAGIARISM DETECTION USING GRAMMAR-BASED FINGERPRINTING

This chapter discusses the concepts and algorithms used in the web-based plagiarism detection system developed under this research. Section 3.1 of this chapter gives an overview of the system. Section 3.2 discusses a converter program that converts source code into a graph file. Section 3.3 discusses SubdueGL, a graph based data mining algorithm to generate graph grammars. Section 3.4 discusses Submatch, an algorithm that computes the similarity between two graph grammars generated by SubdueGL. Lastly, section 3.5 discusses the web-based interface developed for this system.

3.1 Overview

The system developed under this research is a web-based system to detect plagiarism in software source code. This system provides a web interface through which several files can be submitted in batches to detect similarity among them. When files are submitted, a converter program first parses all the source code files and converts them into graph files. Files written in C, C++, Java, and Fortran programming languages can be converted into graph files. The next step is to find the graph grammar using a graph based data mining algorithm, called SubdueGL. Next, an algorithm named Submatch is used to find the degree of similarity between the graph grammars generated by

SubdueGL. The system then emails the results to the users. The results show the degree of similarity between different source code files and the suspected plagiarism cases.

This system, unlike other plagiarism detection methods discussed in Chapter 2, can compare source code files written in different programming languages, and detect plagiarism among them. For example, it can compare code written in C and Java. This is a unique feature of this system. Another advantage is that only a single zip file containing all the source code files needs to be submitted. In other words, the user need not spend their time creating separate zip files for source files written in different programming languages.

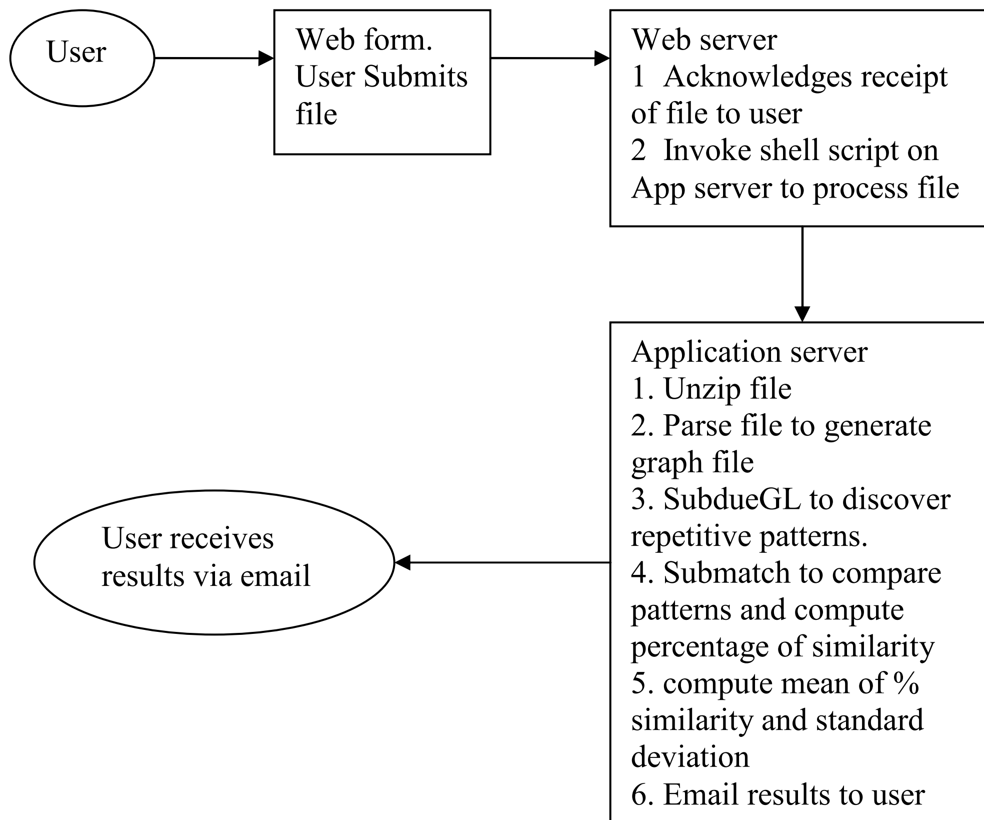

Figure 1: Information flow in our system

This system was designed on a Linux platform and consists of two major parts. The first part is the web server, where files are received, and an acknowledgement is sent to the sender. A shell script is also invoked to start the second step. The second step is the processing of the files on the application server. After the processing is completed an email is sent to the user containing the results. The conceptual flow of information in this system is shown in Figure 1. The rest of this chapter discusses the components that make up this system.

3.2 Parser

Parser is a program developed to convert the source code files into graph files with standard features – labeled vertices and edges. The goal of the parser is to create a graph file with the vertices and edges without checking the functionality/validity of the source code. This was decided with an assumption that for finding the similarity between documents, it is not essential to know whether the source code can be compiled. Another assumption made was that plagiarism can be committed between code files written in two different programming languages. Thus, the parser is designed to extract tokens from the source code files written in different programming languages. The first step the parser does is to find which language the source code is written in. It then concatenates all the source code files written by the same person into a single file. Next, it parses the source code file to find all the tokens that represent the vertices in the graph file. Each word or symbol is considered a single token. All the insignificant data such as single lines of comments and blocks of comments are ignored for tokenizing. Strings are removed and replaced with “String” in the graph file. This is done with a view that text strings have the same structural meaning in the language and very easily changed by plagiarizers, thus

they would make it harder to match documents (Jonyer, Apiratikul, and Thomas, 2005). Figure 2 shows a graph file generated by the parser. A graph, as a network of nodes and relationships, for the graph file shown in Figure 2 is illustrated in Figure 3.

```
% File Name: test.c
s
%
% VERTICES
%
v 1 for
v 2 (
v 3 j
v 4 =
v 5 2.0
%
% EDGES
%
d 1 2 NEXT
d 2 3 NEXT
d 3 4 NEXT
d 4 5 NEXT
```

Figure 2: Graph file generated by the parser

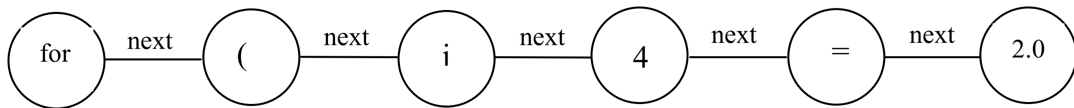

Figure 3: Graph of graph file presented in Figure 2

3.3 SubdueGL

SubdueGL (Jonyer, Holder, and Cook 2002) algorithm is based on a graph-based data mining system named Subdue (Cook and Holder 2000) that can extract common

substructures from graphs. SubdueGL takes its input in the form of a graph with standard features: labeled vertices and labeled edges. Edges can be directed or undirected. When converting data to a graph representation, objects and values are mapped to vertices, and relationships and attributes are mapped to edges. SubdueGL performs its search on a graph in an iterative manner, such that, each iteration results in a grammar production. When a production is found, it is abstracted away from the input graph by replacing each occurrence of it by the non-terminal graph. In each iteration, SubdueGL searches for the best substructure which is used in the next production rule.

The search starts by finding all uniquely labeled vertices and their instances in the input graph. Then, SubdueGL applies the ExtendSubstructure search operator, which extends each of these single-vertex substructures to produce 2-vertex substructures. The ExtendSubstructure search operator extends the substructures in all possible directions to find instances that match to form new substructures. However, only the best substructures are used for further extension. Best substructures are identified using the principle of minimum description length (MDL). MDL principle was introduced by Rissanen (1989) and states that the best theory is the one that minimizes the description length of the entire data set. The iterations continue until the entire graph is abstracted into a single non-terminal, or a user-defined condition is reached. For example, a limit on the number of production rules to be found can be defined by the user or the user can select one or more options provided by SubdueGL.

SubdueGL can also discover recursive productions which are done through the Recursify-Substructure search operator. The Recursify-Substructure search operator is applied to each substructure and checks each instance of the substructure to find if it is

connected to any other instance of the same substructure by an edge. If so, a recursive production can be produced. The operator adds the connecting edge to the substructure and collects all possible chains of instances. Then, the chain of subgraphs is abstracted away and is replaced with a single vertex.

Another feature provided by SubdueGL is that if any of the commonly occurring substructures are found connected to different vertices, those vertices can be turned in to variables. The variables are discovered inside the ExtendSubstructure search operator. After collecting all the instances that match, SubdueGL collects all instances that were extended by the same edge, regardless of which vertex they point to (except the vertex which is already in the substructure). This new vertex is replaced with a variable (non-terminal) vertex and a new substructure is formed. This new substructure is also evaluated along with the other substructures found, to find the best substructure.

An illustrative example is given from Figure 4 to Figure 8 to explain how SubdueGL works. Figure 4 is the graph representation of an artificially generated domain showing vertices and edges.

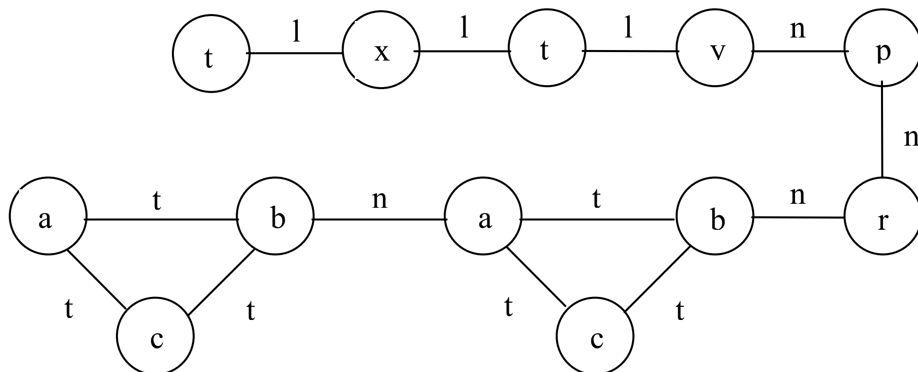

Figure 4: Input graph

SubdueGL, will start by collecting all the unique vertices in the graph and then expand them in all possible directions by applying the ExtendSubstructure search operator. For example, in Figure 4, expanding vertex “a” in all directions will produce 2-vertex substructures which are (a, t, b), (a, t, c), and (b, n, a). Since the first two substructures have two instances each and last one has only one instance, the first two substructures will be selected for further extension. After applying ExtendSubstructure search operator few more times, substructure will have (a, b, c) vertices and as it can be seen from Figure 4, this substructure with the vertices {a, b, c} is the biggest and most common substructure. Thus, this substructure will get selected and SubdueGL will execute RecursifySubstructure operator which will result in the recursive grammar rule shown in Figure 5. Figure 6 shows the input graph after abstracting away both the instances of the substructure using the production results of Figure 5. SubdueGL uses this input graph to learn the next grammar rule.

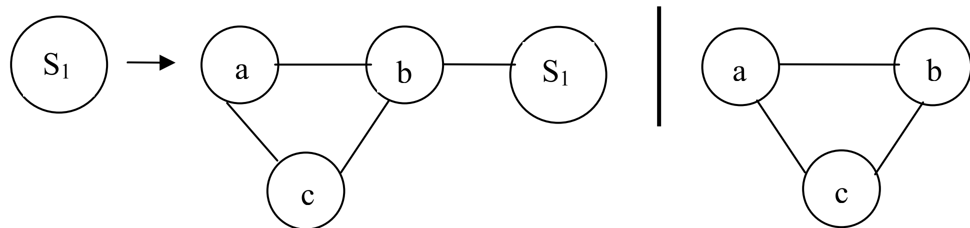

Figure 5: First production by SubdueGL

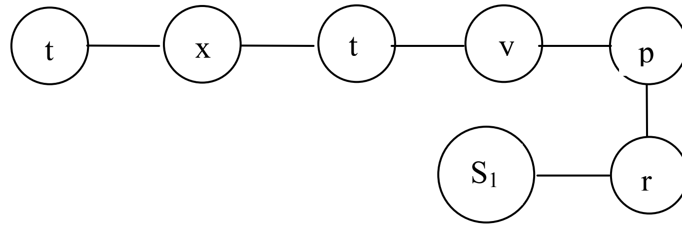

Figure 6: Input graph after first production by SubdueGL

In the next iteration, SubdueGL will find single instances of substructures with the edges {t, l, x}, and {t, l, y}. At this point, SubdueGL will create a variable by replacing the vertices 'x', and 'y' with a non-terminal vertex (S_3), thereby, generating a new substructure with two instances. Since, this is now the biggest and common substructure, Recursify-Substructure operator will be applied to see if any instances are connected. Since, both of them are connected with an edge a recursive substructure will be created as shown in Figure 7. The input graph after abstracting both instances of the substructure using the results of Figure 7 is shown in Figure 8. As it can be seen, there are no more recurring substructures that can be abstracted out, therefore, graph in Figure 8 becomes the final production rule.

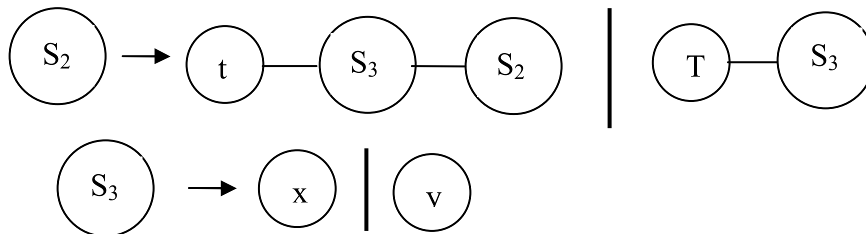

Figure 7: Second and third production rule by SubdueGL

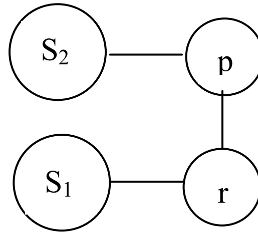

Figure 8: Input graph after second and third productions from SubdueGL

3.4 Submatch

Submatch is an algorithm that can compare graph grammars generated by SubdueGL. It takes two graph grammar files as its input and compares them to see if the graphs are isomorphic. If any two graphs are not found to match exactly, Submatch transforms them to match each other and computes a minimum cost of transformation of a graph to an isomorphism of another graph. Transformation cost is one unit for every change made in any vertex or an edge.

Before the comparison, Submatch determines which of the two grammars is bigger. It then begins comparison by comparing the first substructure of the bigger file to every substructure in the smaller file. As soon as a matching substructure is found, the substructure from the bigger file and the matched substructure from the smaller are removed and not used for any further comparisons. If an exact match for a substructure is not found, a transformation cost is computed to identify a substructure in the smaller file that requires minimum cost of transformation to match the substructure in the larger file. This substructure is then removed and not used for further comparisons. The comparison

continues until all the substructures of the bigger file are matched/transformed with the smaller file. An illustrative example to explain the above procedure follows.

Figure 9 shows two graph grammar files G1 and G2 generated by SubdueGL. Each of them has three substructures S1, S2, and S3. Since, G1 is a bigger file with 11 vertices and edges, as compared to 9 vertices and edges in G2, Submatch will start by trying to find an exact match for its substructure S1. As can be seen from Figure 9, substructure S3 of G2 is an exact match to S1 in G1. Therefore, S3 in G2 will be removed so that it will not be compared with any other substructures. S1 in G1 will also be removed since an exact match has been found. Next, S2 of G1 will be compared to the remaining substructures in G2. Since, no exact match will be found, a minimum transformation cost will be computed, which is equal to 4 transformations. Similarly, a transformation cost of 2 transformations will be incurred to match S3 of G1. Thus, the total cost to match G1 to G2 will be 6 transformations.

The final step of Submatch is to compute the similarity between the two graphs. The percentage of similarity is computed as follows:

$$\text{Similarity} = ((|V+E| - \text{total transformation cost}) / |V+E|) * 100$$

where |V+E| is the total number of vertices and edges in the bigger graph. Thus the similarity between G1 and G2 is equal to ((11 – 6) / 11) * 100 = 45.44%.

G1	G2
% Result of 1. iteration: s 2 v 1 while v 2 (e 1 2 NEXT **S1**	% Result of 1. iteration: s 2 v 1 int v 2 m e 1 2 NEXT **S1**
% Result of 2. iteration: s 2 v 1 j v 2 + v 3 k e 1 2 NEXT e 2 3 NEXT **S2**	% Result of 2. iteration: s 2 v 1 > v 2 0 e 1 2 NEXT **S2**
% Result of 3. iteration: s 2 v 1 2 v 2 ; e 1 2 NEXT **S3**	% Result of 3. iteration: s 2 v 1 while v 2 (e 1 2 NEXT **S3**

Figure 9: Comparison of two fingerprints

3.5 Web Interface

The system developed under this research is an easy to use and yet powerful web-based system to detect plagiarism in software source code. This section describes the interface developed for file submission and processing integrating the components described above. The interface consists of web-based file submission, CGI-Perl script to

receive the files on the server and a shell script to process the files. A discussion of these parts is given next.

3.5.1 Interactive HTML Form for File Submission

An interactive HTML page is developed for users to submit the files. The web page is an easy to use form as shown in Figure 10. The URL for this form is "http://www.agentlab.net/njs/file_submit.html". It requires the users to enter their name, email address and submit the source code file in a compressed format as a single zip file. Users can submit a zip file containing source code files in the following two formats:

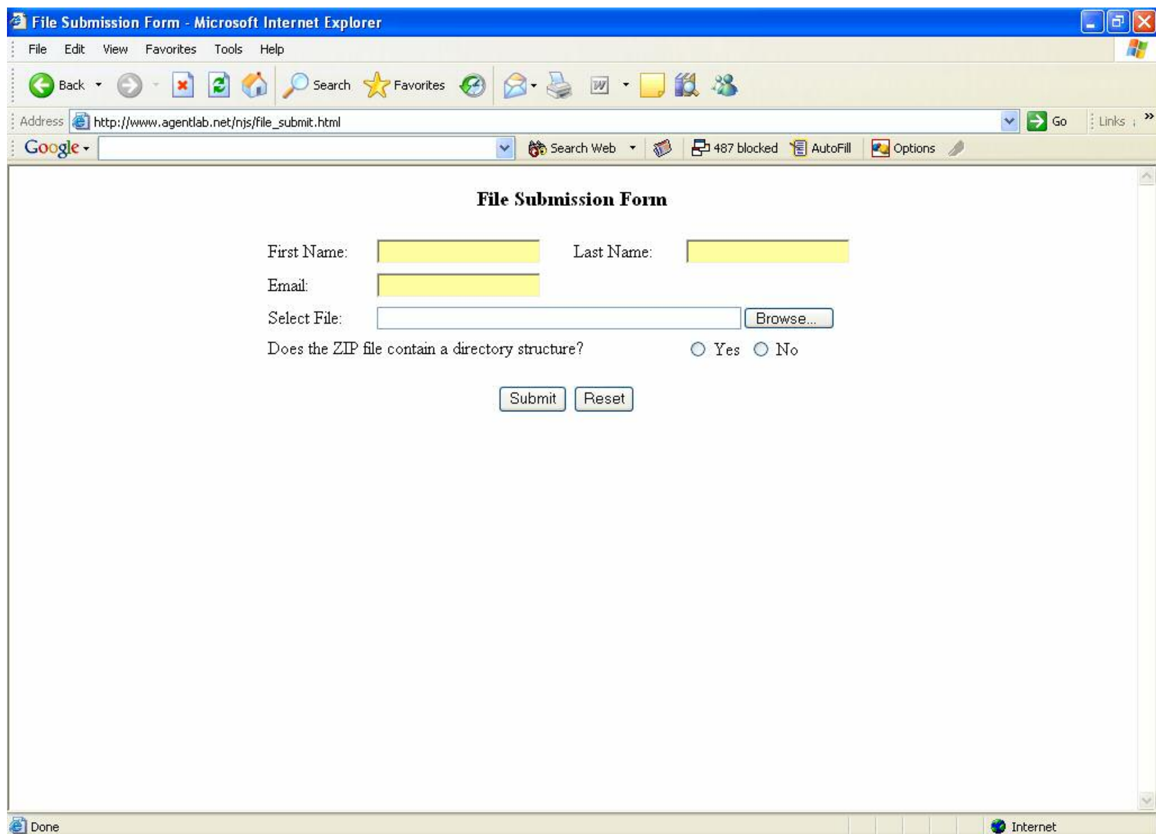

Figure 10: File submission form

- A zip file containing any number of source code files. However each source code file should belong to a different student. For example:
    - Zipfile.zip/1.c
    - Zipfile.zip/2.c
    - Zipfile.zip/3.cpp
    - Zipfile.zip/4.java
    - Zipfile.zip/5.f
- A zip file containing subdirectories/subfolders, where each subdirectory should belong to a different student. In other words, each student's files will be under a subfolder belonging to that student. For example:

    Zipfile.zip/John/1.c
    Zipfile.zip/John/1.h
    Zipfile.zip/Paul/2.cpp
    Zipfile.zip/Paul/2.h
    Zipfile.zip/Tim/4.java
    Zipfile.zip/Alice/5.f

The form requires the users to specify, using a check box, which one of the above two formats are being used.

3.5.2 CGI Script

The data and file submitted through the HTML form are processed by a server based CGI Script written in the Perl scripting language. The information about the user received through the web page is saved in a file on the server for later use to communicate the results via email. The script invokes a fork where the parent process returns an acknowledgement of file submission as an HTML page and terminates, while the child process starts a Korn shell script to process the submitted file.

3.5.3 Shell Script

The processing of submitted files is started by a Shell script. The processing starts by unzipping all the files and checking if the second option on the submission form was

selected. Second option is selected by the users in the case where the submitted zip file contains subfolders and the files within the subfolder belong to one student. In this case, the files in each folder are merged into a single file. Next, the source code files are converted into graph files using the parser. The graph files are used as an input to SubdueGL, which then generates the fingerprints from the source code. The output files from SubdueGL are saved in a folder for comparison. The next step involves applying the matching algorithm, Submatch, to find the percentage of similarity between each pair of files submitted by the users. The input to Submatch are graph grammar fingerprints produced by SubdueGL. Once all the files in pairs are processed by Submatch, the average and standard deviation are calculated and an email is sent to the users with the results as shown in Figure 11. Since we are looking for outliers, the files with the percentages of greater than average plus two standard deviations are treated as the suspected cases of plagiarism (Jonyer, Apiratikul, and Thomas, 2005).

```
Neeraja Samuel,

Results
1.c   10.c  48.32%
1.c   2.c   48.66%
1.c   3.c   45.74%
1.c   8.c   47.54%
1.c   9.c   47.13%
10.c  2.c   43.10%
10.c  3.c   39.74%
10.c  8.c   39.93%
10.c  9.c   47.76%
2.c   3.c   47.20%
2.c   8.c   40.62%
2.c   9.c   46.53%
3.c   8.c   40.18%
3.c   9.c   41.19%
8.c   9.c   63.76%

No. of Data        =    15
Mean Value         = 45.83
Standard Deviation =  6.01
```

Figure 11: Email sent to user

CHAPTER IV

A COMPARATIVE STUDY OF PLAGIARISM DETECTION SYSTEMS

Several studies were conducted to test the strengths and limitations of the plagiarism detection systems studied under this research and to compare their performance. The systems included in this study were MOSS, JPlag, SID, and the new web-based plagiarism detection system using graph grammars developed under this research. Yap could not be in this study, because, despite our best effort, the application could not be configured to work on our UNIX or Linux platforms for lack of clear documentation. This chapter will discuss all the studies conducted along with the results and findings. Section 4.1 of this chapter discusses the methodology adopted for this study. Section 4.2 presents an analysis of the results from real world data. Lastly, section 4.3 discusses results of additional tests conducted to further explain the differences observed between systems in this study.

4.1 *Methodology*

The first step in conducting the comparative study was to select a few real world data sets of source code files written in different programming languages. One of the objectives of this study was to find out if all the systems will find the same cases of plagiarism for a given set of data. The next objective was to study the reasons for differences in results, if any. The last objective was to study their performance, strengths and weaknesses. The real world data was expected to highlight some differences between

the functioning of the systems under study, which may not be evident from the data in controlled domain.

Accordingly, four data sets from the real world were chosen. All of these data sets were collections of source code files submitted by students as programming assignments. The data sets are named Assignment 1, 2, 3 and 4, for identification in the discussion. The first data set consists of 19 source code files written in C and C++ programming languages. The second and third data sets each consists of 20 source code files written in C/C++. The fourth data set consists of 26 source code files written in C, C++ and Java.

The assignments were processed using the system developed in this research and then in MOSS, JPlag and SID. This step was done in two parts. In the first part data sets Assignment 1 and Assignment 2 were processed in the system developed under this research with and without the "-exhaust" option in SubdueGL. The purpose of this variation was to study the effect of these options on the results, and to determine the suitable option to compress the data using graph grammars and generate the fingerprints. As mentioned earlier in chapter III, SubdueGL provides some options for graph data mining. One of the options is the "exhaust" option, which prevents SubdueGL from stopping after discovering all the sub-graphs that can compress the graph, and have it continue until the input graph is compressed into a single vertex. In other words, with the "exhaust" option the entire graph file is compressed, whereas without "exhaust" option, only the portions of the graph that can be compressed are included in the graph grammar.

In the second part of this step, Assignment 3 and Assignment 4 were processed in SubdueGL using the option selected in the previous part. After gathering results from all of the above tests, a detailed analysis was performed to compare the data and identifying

the similarities and differences observed in the results. A detailed discussion of the results and its analysis is given in section 4.2.

The next step was to further investigate the results obtained from the various systems, especially in the areas where differences were observed among the systems. This study was mainly conducted using the visual tools provided in the graphical user interface (GUI) by MOSS, JPlag and SID. This step of the study involved conducting additional tests to study the changes in results given by the systems due to changes introduced in the data files. The details of these tests are presented in section 4.3.

During the course of the study SID became unavailable, and therefore, some of the comparative results do not include data from SID. This has impacted the comparison presented in Section 4.2 for Assignment 3 and Assignment 4, and some results in Sections 4.3.

4.2 *Results and Analysis of Tests Using Real World Data*

This section discusses the results obtained from tests conducted with the four selected data sets that were mentioned in section 4.1. The results in Tables 2 and 3 were obtained by processing Assignment 1 with and without the exhaust option, respectively. The results with the exhaust option identified three suspected cases of plagiarism. They are files 4 and 8, 8 and 9, and 8 and 19. The percent similarity between files 9 and 19 was also very close to the detection value of 56.80% for suspected plagiarism cases. However, since, file 8 is common to the three cases identified in this test, results of files 4 and 9, and 4 and 19, were also included in the comparison with results obtained from other systems. Results from this test for these files are 54.28% and 53.67%, respectively, which is below the detection value of 56.80%, but nevertheless high. As mentioned previously,

the detection value used for this study is the sum of mean of percent similarities and two standard deviations. This formula is merely a heuristic to point to possible cases of plagiarism.

The results without the exhaust option identified only one suspected case of plagiarism, which was between files 8 and 9. The files 5 and 7, and 8 and 19, were very close to the detection value of 61.25% for suspected plagiarism cases. Again, since, file 8 was common between two of the results in this test, results of files 9 and 19 was considered for comparison with results obtained from other systems. The result for files 9 and 19 was 57.23%.

Table 2: Assignment 1 - with –exhaust option

	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
19	40.36	42.47	42.34	53.67	47.07	34.87	50.81	57.46	56.06	41.60	49.06	37.73	43.14	35.03	46.45	48.32	37.72	45.95
18	41.33	45.23	44.68	45.64	52.20	40.12	45.70	41.97	44.30	39.12	40.94	38.30	40.32	29.26	46.29	47.72	35.05	
17	42.41	40.02	30.82	40.48	36.52	25.76	39.28	41.31	40.11	44.25	43.42	28.43	48.67	43.65	37.53	34.77		
16	43.39	45.62	48.40	44.76	54.91	37.44	50.26	43.70	47.27	41.26	46.00	36.38	39.68	32.53	48.62			
15	43.15	48.50	42.21	47.63	47.80	35.66	49.22	43.01	46.44	40.36	44.94	38.43	40.54	32.69				
14	40.05	37.21	29.10	34.36	31.35	22.83	38.63	38.21	39.21	42.47	37.71	24.33	42.31					
13	51.14	45.08	37.95	48.32	41.95	26.81	50.49	49.73	49.41	51.14	47.68	31.03						
12	30.67	35.16	41.64	32.79	41.25	40.14	37.63	36.07	35.51	28.41	37.65							
11	40.24	49.41	41.06	48.59	44.94	32.82	51.06	50.52	52.71	47.35								
10	48.37	46.56	37.77	43.29	38.67	26.27	47.13	47.80	50.51									
9	49.76	48.10	39.79	54.28	50.00	28.38	50.59	63.93										
8	51.79	42.31	37.80	57.69	48.21	30.17	51.33											
7	52.36	45.05	42.97	53.74	47.92	33.85												
6	28.12	31.76	40.78	29.05	34.99													
5	39.15	47.06	47.51	47.63											No. of Data			171
4	39.64	43.52	41.40												Mean Value			42.16
3	39.27	41.31													Standard Deviation			7.32
2	41.33														M + 2SD			56.80

Results from MOSS, JPlag, and SID, along with the results obtained from our system for Assignment 1 are presented in Table 4. Table 4 only presents the suspected cases of plagiarism detected by the various systems. The values in the bold type indicate

that they are cases detected as suspected of plagiarism in that system. Table 4 shows that the three cases detected using the exhaust option were also detected by MOSS. JPlag detected files 8 and 9, and 4 and 8. However, SID and our system, without the exhaust option, detected similarity between files 8 and 9 only. The distance measure for files 4 and 8, and 8 and 19 in SID was 82 and 83, respectively, which seems to be rather high to indicate any plagiarism. As stated previously, SID ranks the files by distance measured and the lower the distance measure the higher the probability of plagiarism. Plagiarism in files 9 and 19, for which the percent similarity was close to detection value with the exhaust option was also detected by MOSS. Results for files 4 and 9, and 4 and 19 from MOSS, JPlag and SID did not detect them as suspected cases of plagiarism, as was the case witth our system.

Table 3: Assignment 1 - without –exhaust option

	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
19	39.86	44.70	39.17	57.37	46.77	35.02	47.70	60.94	57.23	37.31	49.02	29.95	46.80	33.27	47.24	54.38	35.84	43.78
18	45.26	44.53	50.94	50.75	54.97	29.38	53.87	40.85	41.58	37.69	37.04	38.44	43.71	34.03	51.36	45.30	33.92	
17	42.13	42.31	33.39	37.59	40.03	23.43	36.36	38.81	42.83	48.78	42.31	29.37	49.30	50.87	38.81	39.51		
16	42.58	47.69	44.31	52.72	44.80	31.93	49.01	50.45	48.91	40.49	51.85	29.46	44.95	36.69	48.02			
15	42.58	44.28	48.91	48.49	51.05	30.43	52.80	44.87	47.52	38.99	47.49	30.71	43.30	36.12				
14	47.53	46.58	38.40	41.44	39.35	16.54	41.63	37.64	43.92	54.29	42.78	20.72	50.38					
13	51.13	44.95	41.86	52.58	51.13	23.71	50.72	48.45	50.10	52.43	49.90	34.43						
12	24.82	31.63	23.81	32.66	36.91	34.65	38.40	29.91	28.71	23.13	30.50							
11	46.19	47.28	42.27	50.33	46.19	28.76	47.49	47.06	50.10	43.47								
10	48.32	43.10	39.74	41.98	40.49	17.72	43.28	39.93	47.76									
9	47.13	46.53	41.19	53.66	48.12	23.76	47.13	63.76										
8	47.54	40.62	40.18	57.37	52.23	24.55	50.22											
7	48.18	54.50	48.27	56.78	60.99	25.07												
6	16.55	23.36	19.37	27.14	29.84													
5	44.04	51.58	46.07	57.54											No. of Data			171
4	48.91	44.28	43.97												Mean Value			42.33
3	45.74	47.20													Standard Deviation			9.46
2	48.66														M + 2SD			61.25

Table 4: Assignment 1 - Comparison of Results

Programs	This System with Exhaust (%)	This System without Exhaust (%)	MOSS			JPlag (%)	SID
			File 1 %	File 2 %	LM		
8 & 9	**63.93**	**63.76**	**59**	**58**	**231**	**67.1**	**65**
4 & 8	**57.69**	57.37	**36**	**35**	**133**	**53.5**	82
8 & 19	**57.46**	**60.94**	**32**	**35**	**110**	41.8	83
9 & 19	**56.06**	57.23	**32**	**35**	**117**	37.4	87
4 & 9	54.28	53.66	32	31	100	36.1	86
4 & 19	53.67	57.37	30	32	105	29.8	86
5 & 7	47.95	**60.99**	16	10	59	13.8	99

LM = Lines Matched

However, since files 4 and 9, and 4 and 19 were not detected by any of the systems as suspected cases of plagiarism, a physical examination was conducted for those files. The physical examination of these files showed that the degree of similarity observed in the files is proportional to the results obtained from the tests. But it is hard to say just by looking at the matched blocks of code if they are plagiarized. They could be some routine code constructs that are typically written that way. The teacher who gave the assignment would be the best judge to make that evaluation based on the requirements of the programming assignment. This brings out the point that these detection solutions are tools to aid the evaluator to identify potential cases of plagiarism, but the final determination can only be made with human intervention by a person possessing knowledge of the subject domain.

In fact, a physical examination was conducted for all the files listed in Table 4. The examination shows that the degree of similarity observed in the files is proportional to the

results obtained from the tests. This is particularly true for the results obtained in the tests with the system developed under this research and MOSS. Generally, a higher percentage of similarity is an indication of a greater probability of plagiarism.

Comparing the results obtained with and without the exhaust option, it was observed that the results without the exhaust option did not show any consistency. The results did not indicate files 4 and 8 as suspected cases, unlike all the other systems. Also, the results detected files 5 and 7 as close to a suspected case, which was not the case with results from any other system. One factor that might contribute to this is the fact that, without the exhaust option only the portions of the graph that can be compressed are included in the graph grammar. Whereas, with the exhaust option, the entire graph file is compressed. These results from tests without the exhaust option seem to suggest that this option is less reliable than the exhaust option. This conclusion will be put to further test in tests with Assignment 2. There were no additional suspected cases of plagiarism detected by MOSS, JPlag or SID for Assignment 1, other than the ones presented in Table 4.

The results in Tables 5 and 6 were obtained by processing Assignment 2 with and without the exhaust option, respectively. The results with the exhaust option identified two suspected cases of plagiarism. They are files 4 and 9, and 9 and 10. The percent similarity between files 4 and 10 was also very close to the detection value for suspected cases of 61.08%. The results without the exhaust option identified the three above mentioned sets of files as suspected cases of plagiarism. However, in the no exhaust option files 1 and 8 were also identified as suspected cases of plagiarism. The results from MOSS, JPlag and SID for Assignment 2 are presented in Table 7. Table 7 shows

that the three cases detected using the exhaust option were also detected by MOSS. However, JPlag did not detect files 4 and 10. The percent match for files 4 and 10 in JPlag was 34.9. Viewing the files with the visualization tools provided by MOSS and JPlag revealed that code blocks identified by MOSS and JPlag differed in one place. This difference is due to the detection of matches at maximal token length and the value selected for minimum match length threshold used in JPlag. These aspects of JPlag algorithm are explained in detail in section 4.3 using a similar example from Assignment 2. SID only detected similarity between files 4 and 9. The distance measure for files 4 and 10, and 9 and 10 in SID was 88 and 83, respectively, which again seems to be rather high to indicate any plagiarism. The results from the no exhaust option detected one case of files 1 and 8 which was not detected by any other system. This again stood out as an inconsistency in results with this option. There were no additional suspected cases of plagiarism detected by MOSS, JPlag or SID in Assignment 2, other than the ones presented in Table 7.

Table 5: Assignment 2 - with –exhaust option

	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
20	54.50	39.30	50.50	43.10	49.00	18.70	44.90	52.10	42.80	47.20	34.70	55.00	45.81	43.85	43.10	47.40	46.80	52.90	39.20
19	42.00	46.14	40.29	46.75	40.96	23.95	45.29	46.33	49.08	44.09	42.90	41.25	38.72	35.56	48.54	43.39	40.90	40.95	
18	55.44	38.30	49.23	46.36	55.12	16.11	49.12	55.72	46.36	52.10	39.40	53.12	46.21	44.83	47.46	50.49	46.75		
17	48.83	39.14	43.88	42.45	46.84	20.73	44.65	47.19	45.87	44.87	36.60	45.75	45.32	41.80	44.76	48.52			
16	52.77	39.78	49.29	44.26	50.33	17.38	49.73	53.24	44.04	48.09	44.81	49.95	46.01	44.56	43.93				
15	46.06	42.54	47.96	52.65	47.82	22.59	49.36	48.38	54.37	50.98	42.06	44.73	44.14	37.34					
14	43.40	33.24	41.53	39.30	45.01	14.80	40.82	43.05	37.70	41.53	32.09	48.40	46.35						
13	46.70	36.85	46.50	41.18	49.56	18.72	41.87	44.04	42.66	44.63	33.10	51.23							
12	52.71	37.97	46.26	44.42	54.96	15.66	45.85	49.95	43.71	48.11	36.13								
11	40.19	41.26	40.29	42.47	35.40	31.23	44.02	39.52	39.24	38.12									
10	49.15	42.94	51.44	61.08	51.42	19.63	49.83	49.14	61.88										
9	43.92	46.25	44.00	67.90	49.02	23.25	49.32	47.19											
8	56.50	40.39	46.76	44.38	47.19	18.25	47.52												
7	50.32	40.46	50.72	48.09	46.73	21.36													
6	20.79	27.53	16.31	26.79	18.74														
5	52.88	39.65	51.09	49.24												No. of Data			190
4	43.82	46.27	45.44													Mean Value			43.15
3	47.55	39.57														Standard Deviation			9.25
2	38.81															M + 2SD			61.65

Table 6: Assignment 2 without –exhaust option

	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	48.77	33.68	52.28	38.42	46.49	29.82	40.18	45.26	35.79	39.65	28.95	48.60	47.19	45.44	41.58	46.49	37.19	54.56	34.74
19	41.28	35.24	36.23	39.94	31.95	38.58	37.43	40.34	41.19	36.59	47.81	38.29	36.27	31.93	43.57	37.32	37.34	32.88	
18	48.53	35.81	47.75	38.75	50.10	29.16	43.25	50.29	37.96	41.10	32.68	52.45	47.55	45.80	44.42	41.68	43.25		
17	44.81	50.00	37.25	45.78	39.83	42.13	50.38	42.23	47.31	51.66	39.13	40.09	42.42	37.77	51.15	40.57			
16	54.97	35.50	47.98	38.13	48.07	31.44	44.22	51.52	38.34	39.15	37.93	51.72	47.87	47.08	42.39				
15	50.55	41.90	46.36	47.77	44.19	32.49	56.43	49.37	43.31	49.61	41.73	50.23	44.26	41.79					
14	44.89	34.31	53.65	35.58	46.72	29.38	41.24	46.72	33.76	37.04	30.29	44.34	49.09						
13	47.34	35.25	51.82	40.37	51.84	32.79	45.29	45.49	35.66	41.80	32.58	47.75							
12	52.10	31.98	47.98	46.62	53.94	26.35	49.77	56.30	45.50	44.59	33.33								
11	37.31	34.29	33.60	43.84	32.99	33.25	42.51	35.71	43.28	40.11									
10	45.25	47.38	40.28	62.06	45.23	39.09	48.40	42.02	62.33										
9	39.36	39.29	34.21	69.55	41.91	43.65	45.72	38.66											
8	59.87	37.82	47.17	38.24	48.13	32.77	44.54												
7	52.32	37.86	47.17	47.06	43.98	29.70													
6	29.58	42.86	27.94	41.62	34.02														
5	46.89	36.10	51.21	43.15												No. of Data			190
4	43.93	42.86	40.48													Mean Value			42.44
3	52.63	34.01														Standard Deviation			7.26
2	35.98															M + 2SD			56.96

Table 7: Assignment 2 - Comparison of Results

Programs	This System with Exhaust (%)	This System without Exhaust (%)	MOSS			JPlag	SID
			File 1 %	File 2 %	LM	(%)	
4 & 9	**67.90**	**69.55**	**78**	**77**	**238**	**81.6**	**49**
9 & 10	**61.88**	**62.33**	**43**	**39**	**129**	**51.7**	83
4 & 10	**61.08**	**62.06**	**40**	**36**	**121**	34.9	88
1 & 8	56.50	**59.87**					

LM = Lines Matched

The result from tests with and without exhaust option for Assignment 1 and Assignment 2 show that the tests with the exhaust option are producing better results. As stated previously, one of the factors that contribute to this is the fact that, without the exhaust option only the portions of the graph that can be compressed are included in the graph grammar. Whereas, with the exhaust option, the entire graph file is compressed into a single vertex. In other words, the fingerprints generated from the no-exhaust option may not represent the entire file. Therefore, the comparison may be taking place using only a portion of the file. Accordingly, the exhaust was selected for the tests with Assignment 3 and Assignment 4, which are discussed next. I would like to point out that SID stopped functioning at this point and remainder of Section 4.2 does not include any results from SID.

The results for Assignment 3 using the system developed under this research are presented in Table 8. The results indicate that there are two cases of plagiarism. They are files 2 and 3, and 11 and 17. The results from MOSS and JPlag for Assignment 3 are presented in Table 9. MOSS and JPlag both detected files 2 and 3 as suspected cases of plagiarism. However, they did not indicate that files 11 and 17 are suspected cases. The

reasons for files 11 and 17 not being detected by MOSS and JPlag were further studied and a discussion on the observations made follows. There were no additional suspected cases of plagiarism detected by MOSS or JPlag in Assignment 3, other than the ones presented in Table 9.

Table 8: Assignment 3

	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
19	36.12	43.77	49.41	41.49	37.12	24.97	18.14	41.44	28.68	18.63	41.74	34.53	15.03	28.45	44.97	22.36	42.11	41.61
18	31.66	36.94	42.39	43.90	29.63	27.38	19.71	44.63	26.69	19.71	45.90	41.01	19.44	33.86	45.11	24.07	45.37	
17	29.67	37.70	43.56	44.40	28.97	30.20	20.96	40.07	26.29	23.88	53.07	47.91	19.72	36.06	43.01	29.28		
16	17.62	20.80	24.82	23.14	14.65	42.70	37.06	23.40	13.71	35.29	27.61	25.21	29.71	28.31	26.19			
15	33.97	44.85	43.09	46.04	32.51	26.46	19.27	42.58	28.74	20.76	41.79	37.72	13.03	28.90				
14	21.61	27.63	26.93	28.93	18.93	34.64	21.23	30.71	20.07	26.44	36.81	34.12	22.35					
13	10.37	13.65	17.33	15.22	9.14	30.00	26.57	16.10	9.87	36.86	20.40	18.52						
12	27.67	40.30	36.30	37.23	23.79	27.58	21.17	36.99	23.77	19.78	41.92							
11	29.51	39.00	41.22	43.77	28.64	28.83	22.39	39.73	24.44	23.93								
10	12.92	17.98	18.62	18.49	11.85	33.24	41.35	17.81	11.32									
9	43.38	28.01	29.87	29.80	43.44	13.25	9.80	27.09										
8	34.37	38.35	44.98	46.80	31.03	26.71	17.69											
7	11.88	15.28	17.21	19.62	12.35	33.24												
6	17.86	22.10	22.37	26.29	16.63													
5	48.33	26.34	36.54	35.47											No. of Data			171
4	36.20	45.94	44.96												Mean Value			29.75
3	39.79	52.00													Standard Deviation			10.66
2	34.21														M + 2SD			51.07

Table 9: Assignment 3 - Comparison of Results

Programs	This System with Exhaust (%)	MOSS			JPlag (%)
		File 1 %	File 2 %	LM	
2 & 3	**52.00**	**18**	**25**	**295**	**59.7**
11 & 17	**53.07**	11	11	36	18.2

LM = Lines Matched

To identify the cause for not detecting similarity in the case of files 11 and 17 in Assignment 3 by MOSS and JPlag, these two files were studied in detail in a text editor. One important observation was that although the code between the two files was

structurally different, there were quite a few common patterns among the two. An example of one such pattern is presented in Figure 12. It is apparent from Figure 12 that both codes are doing the same operation of trying to set the values of a two-dimensional array. The array is named programMemory in file 11 and progMem in file 17. There is a difference in the sequence in which the array is being populated between the two files, however, the code lines are similar. The lines are not identical, in that the variable names and function identifiers are different, yet there is considerable amount of similarity. This resulted in similar sub-structures in the graph grammar generated by SubdueGL for both files. SubMatch can match these sub-structures after incurring some transformation cost. Some sub-structures resulting out of code like the examples given below are presented in Figure 13:

(file 11)

```
if(action=="ADD") programMemory[pMemAddr][0]=1;
if(action=="SUB") programMemory[pMemAddr][0]=-1;
if(action=="MULT") programMemory[pMemAddr][0]=2;
if(action=="DIV") programMemory[pMemAddr][0]=-2;
if(action=="SQR") programMemory[pMemAddr][0]=3;
if(action=="SQRT") programMemory[pMemAddr][0]=-3;
remStr=strtok(NULL," ");
programMemory[pMemAddr][1]=search(remStr);
remStr=strtok(NULL," ");
programMemory[pMemAddr][2]=search(remStr);
remStr=strtok(NULL,"\n");
programMemory[pMemAddr][3]=search(remStr);
```

(file 17)

```
if(instruction=="ADD") progMem[progAddr][0]=1;
if(instruction=="SUB") progMem[progAddr][0]=-1;
if(instruction=="MULT") progMem[progAddr][0]=2;
if(instruction=="DIV") progMem[progAddr][0]=-2;
if(instruction=="SQR") progMem[progAddr][0]=3;
if(instruction=="SQRT") progMem[progAddr][0]=-3;
curVar=strtok(NULL," ");
progMem[progAddr][1]=addrof(curVar);
```

```
curVar=strtok(NULL," ");
progMem[progAddr][2]=addrof(curVar);
curVar=strtok(NULL,"\n");
progMem[progAddr][3]=addrof(curVar);
```

Code snippet: file 11

```
if(action=="ADD")
   programMemory[pMemAddr][0]=1;
if(action=="SUB")
   programMemory[pMemAddr][0]=-1;
if(action=="MULT")
   programMemory[pMemAddr][0]=2;
if(action=="DIV")
   programMemory[pMemAddr][0]=-2;
if(action=="SQR")
   programMemory[pMemAddr][0]=3;
if(action=="SQRT")
   programMemory[pMemAddr][0]=-3;
.
.
.
if((programMemory[pMemAddr][0]==1) ||
  (programMemory[pMemAddr][0]==-1) ||
  (programMemory[pMemAddr][0]==2) ||
  (programMemory[pMemAddr][0]==-2) ||
  (programMemory[pMemAddr][0]==6) ||
  (programMemory[pMemAddr][0]==-6))
{
   remStr=strtok(NULL," ");
   programMemory[pMemAddr][1]=search(remStr);
   remStr=strtok(NULL," ");
   programMemory[pMemAddr][2]=search(remStr);
   remStr=strtok(NULL,"\n");
   programMemory[pMemAddr][3]=search(remStr);
}

if((programMemory[pMemAddr][0]==0) ||
   (programMemory[pMemAddr][0]==3) ||
   (programMemory[pMemAddr][0]==-3))
{
   remStr=strtok(NULL," ");
   programMemory[pMemAddr][1]=search(remStr);
   remStr=strtok(NULL,"\n");
   programMemory[pMemAddr][3]=search(remStr);
}
```

Code snippet: file 17

```
if((instruction=="ADD") || (instruction=="SUB") ||
  (instruction=="MULT") || (instruction=="DIV"))
{
   if(instruction=="ADD")
      progMem[progAddr][0]=1;
   if(instruction=="SUB")
      progMem[progAddr][0]=-1;
   if(instruction=="MULT")
      progMem[progAddr][0]=2;
   if(instruction=="DIV")
      progMem[progAddr][0]=-2;

    curVar=strtok(NULL," ");
    progMem[progAddr][1]=addrof(curVar);
    curVar=strtok(NULL," ");
    progMem[progAddr][2]=addrof(curVar);
    curVar=strtok(NULL,"\n");
    progMem[progAddr][3]=addrof(curVar);
}

if((instruction=="SQR")||(instruction=="SQRT"))
{
   if(instruction=="SQR")
      progMem[progAddr][0]=3;
   if(instruction=="SQRT")
      progMem[progAddr][0]=-3;

   curVar=strtok(NULL," ");
   progMem[progAddr][1]=addrof(curVar);
   progMem[progAddr][2]=0;
   curVar=strtok(NULL,"\n");
   progMem[progAddr][3]=addrof(curVar);
}
```

Figure 12: Example of code comparison between files 11 and 17

Sample sub-structures: file 11	Sample sub-structures: file 17
% Result of 1. iteration: s 46 v 1 progMem v 2 [v 3 progAddr v 4] v 5 [e 1 2 NEXT e 2 3 NEXT e 3 4 NEXT e 4 5 NEXT	% Result of 8. iteration: s 10 v 1 programMemory v 2 [v 3 pMemAddr v 4] v 5 [e 1 2 NEXT e 2 3 NEXT e 3 4 NEXT e 4 5 NEXT
% Result of 2. iteration: s 32 v 1 (v 2 instruction v 3 = v 4 = v 5 string v 6) e 1 2 NEXT e 2 3 NEXT e 3 4 NEXT e 4 5 NEXT e 5 6 NEXT	% Result of 3. iteration: s 16 v 1 ; v 2 if v 3 (v 4 action v 5 = v 6 = v 7 string v 8) e 1 2 NEXT e 2 3 NEXT e 3 4 NEXT e 4 5 NEXT e 5 6 NEXT e 6 7 NEXT e 7 8 NEXT
% Result of 5. iteration: s 18 v 1 curVar v 2 = v 3 strtok v 4 (v 5 NULL v 6 , v 7 string e 1 2 NEXT e 2 3 NEXT e 3 4 NEXT e 4 5 NEXT e 5 6 NEXT e 6 7 NEXT	% Result of 4. iteration: s 10 v 1 remStr v 2 = v 3 strtok v 4 (v 5 NULL v 6 , v 7 string v 8) v 9 ; e 1 2 NEXT e 2 3 NEXT e 3 4 NEXT e 4 5 NEXT e 5 6 NEXT e 6 7 NEXT e 7 8 NEXT e 8 9 NEXT

Figure 13: Sample sub-structures for files 11 and 17

This shows that the system developed under this research can detect similarity even if the similarity is not in contiguous blocks of code and in situations where the source codes are structurally dissimilar. This may also explain the higher percent match obtained from this system compared to MOSS and JPlag in few cases in the tests performed in this research.

Before processing the files in Assignment 4, all the JAVA files were removed from this data set. This was done because the systems, other than the one developed under this research, do not have the capability to compare files written in different programming languages. This resulted in the data set being truncated to only contain 15 C/C++ files. The results for Assignment 4 using the system developed under this research are presented in Table 10. The results indicate that there is one case of plagiarism, files 8 and 9. The results from MOSS and JPlag for Assignment 4 are presented in Table 11. MOSS and JPlag both detected files 8 and 9. There were no additional suspected cases of plagiarism detected by MOSS or JPlag in Assignment 4, other than the ones presented in Table 11.

Table 10: Assignment 4

	%	%	%	%	%	%	%	%	%	%	%	%	%	%
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	19.77	12.95	30.75	18.96	26.47	38.40	32.30	36.88	32.83	27.62	30.64	28.10	27.00	33.06
14	29.69	8.96	23.19	18.50	32.07	41.20	42.19	35.77	35.44	37.99	30.76	51.89	46.16	
13	27.54	8.07	20.79	16.68	32.27	38.25	39.26	31.73	32.89	39.49	29.95	49.03		
12	20.50	8.37	21.69	14.69	30.40	39.97	46.80	34.50	31.51	44.32	30.74			
11	40.72	13.80	36.50	31.29	45.92	39.41	35.73	46.64	39.00	31.45				
10	21.60	9.02	22.79	15.22	33.45	38.38	43.48	35.73	33.18					
9	37.23	13.20	31.54	22.00	40.59	46.40	39.06	54.28						
8	37.11	13.03	32.65	23.05	44.77	44.71	43.99							
7	21.67	9.12	26.39	19.53	38.30	42.17								
6	34.80	11.94	30.52	23.99	42.23									
5	40.84	13.52	32.14	26.91							No. of Data			105.00
4	24.10	18.98	29.94								Mean Value			30.53
3	30.72	19.75									Standard Deviation			10.90
2	12.17										M + 2SD			52.33

Table 11: Assignment 4 - Comparison of Results

Programs	This System with Exhaust (%)	MOSS			JPlag (%)
		File 1 %	File 2 %	LM	
8 & 9	**54.28**	**22**	**22**	**197**	**56.6**

LM = Lines Matched

4.3 *Results and Analysis of Additional Tests*

Additional tests were performed to better understand the working of the systems compared in this research. These tests were conducted by viewing the files side by side in the visualization tool provided in the graphical user interface (GUI) by MOSS, JPlag and SID. This study focused mainly on JPlag and SID because the results obtained with MOSS in previous tests agreed with the results obtained from the system developed in this research in almost all the cases. The first observation made was, that the blocks of code that were shown as a match differed in all the systems. There was some overlap, but there were several places where the code blocks identified as matching were completely

different. This was especially true in the case of JPlag and SID. Some examples of the differences are presented in Figures 14 and 16. The blocks of code shown in Figure 14 were identified as matching in SID, but were not detected as matching in JPlag. These blocks of codes were taken from files 9.c and 10.c of Assignment 2. As explained previously, JPlag algorithm in the first phase searches for the biggest contiguous matches between two strings. In the second phase all matches of maximal length found in phase 1 are marked so that they are not used in further matches of subsequent iteration. This ensures that every token will only be used in one match. The iterations then continue, and at each step the maximal length is lowered by at least one, until the minimum match length threshold is met. Since the iterations stop when the minimum match length is met, any block of code that may be matching but does not possess tokens equal to or higher than the minimum match length, will not be identified as a matching block. The default value for minimum match length is 12. This test was run using the default value and JPlag did not detect the blocks of code as similar. The JPlag user interface does have an option which allows the user to change the value of minimum match length. When an experiment was run by setting this value to 8, blocks of code shown in Figure 14 were identified as a match. However, JPlag cautions against the use of a very low number for minimum match length, as this could increase the chances of a few tokens to frequently occur, thus rendering the results spurious.

9.c	10.c

void main(int argc,char *argv[])
{
if (strcmp(argv[1],"-k")==0) /* IF THE COMMAND LINE ARGUMENT argv[1] IS "-k" THEN THIS IS EVALUATED TO TRUE AND
 	
THE PROCEDURE TO GENERATE THE PUBLIC AND SECRET KEYS IS CALLED */
generatekeys();
else if (strcmp(argv[1],"-e")==0) /* IF THE COMMAND LINE ARGUMENT argv[1] IS "-e" THEN THIS IS EVALUATED TO TRUE
 	
AND KEYS ARE GENERATED AND THE MESSAGE IS ENCRYPTED */
{
generatekeys();
encryption();
}
else if (strcmp(argv[1],"-d")==0) /* IF THE COMMAND LINE ARGUMENT argv[1] IS "-d" THEN THIS IS EVALUATED TO TRUE
 	
AND KEYS ARE GENERATED AND THE MESSAGE IS ENCRYPTED AND THEN DECRYPTED */
{
generatekeys();
encryption();
decryption();
}
	void main(int agrc,char *argv[])
	{
	if(strcmp(argv[1],"-k")==0)
	{
	generatekeys();
	}
	else if(strcmp(argv[1],"-e")==0)
	{
	generatekeys();
	encryption();
	}
	else if(strcmp(argv[1],"-d")==0)
	{
	generatekeys();
	encryption();
	decryption();
	}

Figure 14: Code identified as a match in SID, but not in JPlag

Yet another test was performed to study the impact of minimum match length value on detecting matches. The encrypt() function of file 9.c was modified by adding some dummy variables and assigning values to them in the body of the function. This caused the code block not to be detected. The reason this happened was that, in the contiguous area where the tokens were being compared, the number of matching tokens fell below the minimum match length. This suggests that a student can circumvent detection by adding some extraneous code to a plagiarized block of code. It is very hard in this system to determine the optimum minimum match length for reliable results. As stated earlier, a very low minimum match length can render the results spurious. Modifications made to file 9.c discussed above are shown in Figure 15.

<table>
<tr><th>Original 9.c</th><th>Modified 9.c</th></tr>
<tr><td>

```
double encrypt(double m,unsigned long e1,double N1)

{

double ecr1=1,ecr2=0;

unsigned long i=0,po;
while(e1>0)

{

        po=e1%2;

        if(i++==0)

                ecr2=fmod(m,N1);

        else

                ecr2=fmod((ecr2*ecr2),N1);
        if(po==1)

        {

                ecr1=ecr1*ecr2;

                ecr1=fmod(ecr1,N1);

        }

        e1=e1/2;

}
return (ecr1);

}
```

</td><td>

```
double encrypt(double m,unsigned long e1,double N1)

{

double ecr1=1,ecr2=0;
double ecr3=1,ecr4=0;
unsigned long i=0,po;

while(e1>0)

{

        po=e1%2;
        ecr3=0;
        if(i++==0)

                ecr2=fmod(m,N1);
                ecr4=0;
        else

                ecr2=fmod((ecr2*ecr2),N1);
        if(po==1)

        {

                ecr1=ecr1*ecr2;

                ecr1=fmod(ecr1,N1);
                ecr4=0;

        }

        e1=e1/2;

}
return (ecr1);

}
```

</td></tr>
</table>

Figure 15: Some minor modifications made to 9.c to test the results in JPlag

The same test was performed in the system developed in this research. Our system successfully identified both the original and modified code as plagiarized with almost no change in the total percentage. The results obtained are summarized in Table 12.

Table 12: Comparison of results with modification to file 9.c

System	Before Modification			After Modification		
	Files		Results	Files		Results
JPlag	9.c	10.c	49.0%	Modified 9.c	10.c	38.7%
This system	9.c	10.c	61.9%	Modified 9.c	10.c	62.0%

Another example of where SID and JPlag did not identify a similar block of code as matching is shown in Figure 16. The blocks of code shown in Figure 16 were identified as a match in JPlag, but not in SID. These blocks of code are two different functions. JPlag matched the decrypt() function in file 9.c with the encrypt() function of file 10.c. The reason this happened was, because these two blocks of code had a total of 31 matching tokens during the iteration process. Even though, encrypt and decrypt blocks of both the files are similar, because a match was detected at the maximal length of 31 between decrypt function in file 9.c and the encrypt function of file 10.c, these tokens were used to identify the blocks as matching. Since these blocks were identified as matching, they were not used again in the subsequent iterations. Hence, the blocks with the same function identifier names were not shown as matching. Later in the iterations, a match was found with the encrypt function of file 9.c and the decrypt function of file 10.c. at the maximal length of 27.

On the other hand, SID matched the encrypt function of 9.c with encrypt function of 10.c, as expected. However, it did not match the code blocks named decrypt. The reason for this behavior is that, encrypt and decrypt functions were very similar and SID, after detecting the first pair, ignored the second pair as duplication. To confirm this, an experiment was conducted by sending files containing the duplicates to SID. The results of this experiment did not come back due to problems with functioning of SID site. The

turnaround time for results from the SID site has been inconsistent. Sometimes the results are returned within five minutes, and at other times the results can get delayed for a long period of time, even for several days.

```
return (ecr1);

}
/* FUNCTION TO DECRYPT THE ENCRYPTED MESSAGE
  THE  DECRYPTION REQUIRES THE SECRET KEY AND
THE VALUE OF M
  IT IS CALCULATED BY c^d MOD N  BY HE METHOD OF
REPEATED SQUARING *
  c - the encrypted text, d - the modulo inverse of e mod n and
N=p*q IS PASSED TO
  PROCEDURE TO DECRYPT THE TEXT .THE FORMULA
USED IS m^e mod n
  THE DECRYPTED VALUE IS RETURNED */

double decrypt(double c,unsigned long d1,double N1)

{

double dec1=1,dec2=0;

unsigned long i=0,po;

while(d1>0) //DO TILL D1 > 0

{

        po=d1%2; // STORES THE VALUE OF D1 MOD 2

        if(i++==0)

                dec2=fmod(c,N1); // IF I++ > 0 FIND
THE C MOD N1 ELSE SQUARE OF DEC2 MOD N1

        else

                dec2=fmod((dec2*dec2),N1);

        if(po==1)

        {

                dec1=dec1*dec2;

                dec1=fmod(dec1,N1);

        }

        d1=d1/2;

}

return(dec1);

}
```

```
return s;
}

/*This function takes the arguments M,e,N and calculates
P(M)=M pow e (mod N), which is nothing
 but the cipher text.This function is called by the function
encryption for every block of message
 that is to be encrypted and the encrypted message P is
returned*/

double encrypt(double m1,unsigned long e1,double N1)
{
   double x1=1,x2=0;
        unsigned long r=0,p1;
        printf("\nMessage to be encrypted:%0.0f",m1);
        while(e1>0)
        {
                p1=e1%2;
                if(r++==0)
                        x2=fmod(m1,N1);
                else
                        x2=fmod((x2*x2),N1);
                if(p1==1)
                {
     x1=x1*x2;
                        x1=fmod(x1,N1);
                }
                e1=e1/2;
        }
        return x1;
}
```

Figure 16: Code identified as a match in JPlag, but not in SID

Some additional experiments were also conducted with the files after changing comments, white spaces and tabs etc. in them but these changes did not show any change in the results from any of the systems. This indicates that all the systems handle differences caused because of comments, tabs and white spaces etc. consistently and do not allow them to bias the results.

Another observation that was made while running JPlag was that the users cannot make multiple submissions to the system. When an attempt was made to make multiple submissions, one after another, JPlag displayed an error massage, stating that a submission was already being processed and that a second submission could not be made until the processing of the first submission was complete. This seems to be a drawback, however, since the processing happens very quickly the user does not have to wait for long periods of time before making the next submission. The results are usually received in less than five minutes. In the case of SID and the system developed in this research, multiple submissions can be made without waiting for the results to come back. However, in the case if SID, the results are not presented by zip file names, and this causes some ambiguity if more than one zip file is submitted at the same time.

CHAPTER V

SUMMARY AND CONCLUSIONS

The need for a good plagiarism detection system cannot be overemphasized due to increasing availability of information in the electronic medium which can easily be plagiarized. This research compared some of the popular plagiarism detection systems and identified their strengths and weaknesses. All of these systems can be used for plagiarism detection. However, they all have some limitations.

MOSS is a command line tool and is not easy to use. JPlag application runs only on platforms that have Java run-time present. The use of minimal match length in JPlag can open up the algorithm to miss some matches, as the algorithm considers matches of string of minimal length only. However, a very low minimum match length can render the results spurious. Further, a plagiarist can circumvent detection by adding some extraneous code to lower the number of matching tokens to fall below the minimum match length. In MOSS smaller k-grams increase the sensitivity but increase the execution time also. To make the program run efficiently MOSS makes a trade-off by setting the value of k to a suitable number. However, many programming languages may have key words/statements that have a length of less than the value of k which can be missed. The study did observe some inconsistencies with the detection method of SID, but the tests could not be completed to arrive at any conclusions. Table 13 presents a comparison of all the systems, including the system developed under this research.

The system developed under this research uses document fingerprinting technique using graph grammars. This algorithm compresses the source code files and generates the graph grammars. The graph grammars are matched for detecting similarities. The results of this research show that this method for detection of similarity between documents yields reliable results when compared with other systems. The method is superior because it can also detect similar patterns between documents even if there is some structural dissimilarity.

The interface provided in this new system is easy to use and is platform independent, being a web-based interface. It does not require specification of language and all files written in languages that are supported can be submitted in a single zip file. The system can be extended to find plagiarism in more languages by enhancing the capabilities of parser. The program allows submission of multiple files without waiting for the results to come back. This is not possible in JPlag.

In the present design of the system developed under this research, the processing of all the files takes place on a single machine. This is causing the processing to take a long time. For a job consisting of 19 files with an average of 450 lines of code per file, it takes about 40 minutes to an hour to complete the processing and return the results. The equipment used for the experiments was a Pentium III 733 MHz machine with 512 MB RAM. When the experiment was run on a Pentium IV 2.8 GHz machine with 512 MB RAM, the result for the same set of files was returned in about 15 to 20 minutes. This processing time can be significantly reduced by implementing the application in a distributed system, where the graph files are processed in parallel on multiple machines simultaneously to generate the grammar fingerprints. The results from these machines

can then be collected on one machine to be fed to the next step. This integration with a distributed system can be investigated in the future.

After completion of this research, the system developed under this research was made available on the internet. Users can submit files for plagiarism detection using this system by going to web site "http://eaton.agentlab.net/dfggi".

Table 13: Comparison of the system developed under this research with existing plagiarism systems

	Our System	MOSS	JPlag	SID	YAP
Algorithms Used	SubdueGL, SubMatch	Fingerprinting technique	Greedy String Tiling	Computes shared amount of information using Kolmogorov's complexity	Running-Karp_Rabin Greedy-String-Tiling
Language Supported	C, C++, Java. Fortran	[1]C, C++, Java. C#, Fortran, etc.	C, C++, Java. C#, scheme	C, C++, Java	Pascal, C, LISP
Platforms for Client	Web-based	Unix	Java program. Requires Java run time applicable to the platform	Web-based	Unix
File Submission Mode	Web form	Command line	Java application	Web form	Command line
File format for submission	Zip file	Files as parameter to Perl executable	Files by specifying the folder in UI	Zip file	By specifying folder as parameter to Perl executable
Language Specificatio n	Not required	Required	Required	Required	Use appropriate tokenizer
Results	Email	Email with URL	HTML page	SID site	Written to file

[1] Python, Visual Basic, JavaScript, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2.

REFERENCES

Apiratikul, P., "*Document Fingerprinting Using Graph Grammar Induction*", Masters Thesis submitted to the Department of Computer Sciences, Oklahoma State University, 2004

Cook, D. J. and L. B. Holder, **"***Graph-Based Data Mining***"**, IEEE Intelligent Systems, 15(2), 32-41, 2000.

Chen, X., B. Francia, M. Li, B. Mckinnon and A. Seker, "*Shared Information and Program Plagiarism Detection***",** IEEE Transactions on Information Theory, vol. 50, pp. 1545-1551, 2004

Jonyer, I., P. Apiratikul, and J. Thomas, "*Source Code Fingerprinting Using Graph Grammar Induction*", Proceedings of the Eighteenth Annual Florida AI Research Society, May 2005.

Jonyer, I., L.B. Holder, and D.J. Cook, "*Concept Formation Using Graph Grammars*", Proceedings of the KDD Workshop on Multi-Relational Data Mining, pages 71-79, 2002.

Jonyer, I., L.B. Holder, and D.J. Cook, "*MDL-Based Context-Free Graph Grammar Induction*", Proceedings of the Sixteenth Annual Florida AI Research Society, 2003.

Prechelt, Lutz, Guido Malpohl, Michael Phlippsen, "*JPlag: Finding plagiarisms among set of programs*", Technical Report 2000-1, March 28, 2000

Schleimer, Saul, Daniel Shawcross Wilkerson, and Alexander Aiken, "*Winnowing: Local Algorithms for Document Fingerprinting*," SIGMOD 2002, 76-85.

Wise, M., "*YAP3: improved detection of similarities in computer program and other texts*", Proceedings of twenty seventh SIGCSE technical symposium on computer science education, Philadelphia, USA. 130-134, 1996.

APPENDIXES

APPENDIX A

CGI SCRIPT

```
#!/usr/bin/perl -w

#------------------------------------------------------------------------------
#  File name    : file_submit.cgi
#
#  Function     : This script processes data submitted by users in web form
#
#  Called by    : called by html page "/srv/www/htdocs/njs/file_submit.html" on
#                  web server
#
#  Author       : Neeraja J Samuel for Master's thesis research
#  Date         : 2005 - 2006
#------------------------------------------------------------------------------

use strict;
use File::Basename;
use CGI ':standard';

#------------------
# declare variables
#------------------
my $f_name;
my $l_name;
my $e_mail;
my $file_name;
my $file_name_ex;
#my $file_type;
my $sub_dir;
my $pid;

#----------------------------------------
# assign variables with data from web form
#----------------------------------------
$f_name = param('f_name');
$l_name = param('l_name');
$e_mail = param('emailAddr');
$file_name = param('up_file');
```

```
$file_name_ex = Get_File_Name($file_name);
#$file_type = param('filetype');
$sub_dir = param('subdir');


print "Content-type: text/html\n\n";


#-----------------------------------------------------------------
# fork the process
#
# parent will sleep for 10 seconds and then call
# /home/sneeraj/bin/process_files process on application server
#
# child will dynamically create an acknowledge page
# upload file attachement to web server
# create a dat file with web form data to add in processing of files
# then terminate
#-----------------------------------------------------------------


if (!defined ($pid = fork)) {
    die "Unable to fork: $!\n";
}
elsif (! $pid){
   # this is branch for child process

   close(STDIN);
   close(STDOUT);
   close(STDERR);
   sleep(10);
   exec ("/home/sneeraj/bin/process_files>/dev/null 2>&1");
   #print "I am the child, and my PID is $$\n";
}
else {

   # this is branch for parent

   #print "<font size=+2>Your name is          <i>$f_name $l_name</i></font><br>";
   #print "<font size=+2>Your email address is <i>$e_mail</i></font><br><br>";
   print "<font size=+2>    <i>$f_name $l_name</i></font><br>";
   if ($file_name_ex) {
     # print "<font size=+2>The file name is $file_name</font><br>";
     print "<font size=+2>Thanks for submitting <i>$file_name_ex</i></font><br>";
     print "<font size=+2>The results will be emailed at <i>$e_mail</i>
     shortly</font><br><br>";
```

```
    open (UPLOAD, ">../htdocs/njs/upload/$file_name_ex") || Error();

    my ($data, $length, $chunk);
    while ($chunk = read($file_name, $data, 1024)) {
      print UPLOAD $data;
    }
    close (UPLOAD);
  } else {
    print "<font size=+2>No file was submited</font>";
  }

  open (OFD, ">../htdocs/njs/upload/$file_name_ex.dat") || Error();
    print OFD "$f_name:$l_name:$e_mail:$file_name_ex:$sub_dir\n";
  close(OFD);

 #  exec ("/home/sneeraj/bin/process_files>/dev/null 2>&1");
  exit;
}

#--------------------------------------------------------
# this sub function returns the name of file attachment
#--------------------------------------------------------
sub Get_File_Name {
   if($ENV{HTTP_USER_AGENT} =~ /win/i){
      fileparse_set_fstype("MSDOS");
   }
   elsif($ENV{HTTP_USER_AGENT} =~ /mac/i) {
      fileparse_set_fstype("MacOS");
   }

   my $full_name =  shift;
   $full_name    =  basename($full_name);
   $full_name    =~ s!\s!\_!g;         # Replace whitespace with _

   return($full_name);
}


sub Error {
  print "Couldn't open temporary file: $!";
  exit;
}
```

APPENDIX B

SHELL SCRIPT

```
#!/bin/bash

#------------------------------------------------------------------------------
#  File name    : process_files
#
#  Function     : This script process zip files submitted by users via web form
#                  This script performs the following actions
#                  1. copies the zip file and the associated dat file on web
#                      server to my home directory
#                  2.
#
#  Called by    : called by cgi script "/srv/www/cgi-bin/file_submit.cgi" on
#                  web server
#
#  Author       : Neeraja J Samuel for Master's thesis research
#  Date         : 2005 - 2006
#------------------------------------------------------------------------------

#--------------------------------------
# some path variables
#--------------------------------------
njsHome="/home/sneeraj"
wwwUpload="/srv/www/htdocs/njs/upload"

#--------------------------------------
# copy file from webserver
#--------------------------------------
mv -f $wwwUpload/* $njsHome/webfiles/
chmod 777 $njsHome/webfiles/*

#------------------------------------------------------------------------------
# create a separate directory for each zip file under /home/sneeraj/webfiles
# create "workspace" and "archive" sub-directories to process files
# extract file type and directory structure information in zip file from dat file
# unzip .zip files in workspace sub-directory
# move zip file from home to /home/sneeraj/webfiles/archive
#------------------------------------------------------------------------------
```

```
for zipfile in $(ls $njsHome/webfiles/*.zip); do
  if [ -f $zipfile ]; then
    zipfileName=$(basename "$zipfile")

    index=`cat $njsHome/bin/next_index.dat`
    next_index=`expr $index + 1`
    echo $next_index > $njsHome/bin/next_index.dat
    mkdir $njsHome/webfiles/zip$index
    mkdir $njsHome/webfiles/zip$index/archive
    mkdir $njsHome/webfiles/zip$index/workspace

    #------------------------------------------
    # next two lines commented out because
    # file type not being asked in web form
    #------------------------------------------
    #fileType=`cut -d":" -f5 $zipfile.dat`
    #echo $fileType

    subDir=`cut -d":" -f5 $zipfile.dat`
    echo $subDir

    #unzip if the zipfile is there

    /usr/bin/unzip -C $zipfile "*.[fch]" "*.cpp" "*.java" -d
    $njsHome/webfiles/zip$index/workspace/

    mv -f $zipfile $njsHome/webfiles/archive
    mv -f $zipfile.dat $njsHome/webfiles/zip$index/.
  fi

  if test $subDir -eq 1
  then
    subfolders="yes"
  else
    subfolders="no"
  fi

  #------------------------------------------------------
  # rename and concatenate files if mutiple files submitted
  # by a student (all files should be in one subfolders)
  #------------------------------------------------------
  if test $subfolders = "yes"
  then
    prevUser="   "

    #--------------------------------------------------------------------
```

```
  # next line commented out because we are now processing all files
  #------------------------------------------------------------------
  #for file in $(find $njsHome/webfiles/workspace/ -iregex '.*\.\([fch]\|cpp\|java\)'); do

  for file in $(find $njsHome/webfiles/zip$index/workspace/ -name '*.*'); do
    if [ -f $file ]; then
      fileName=$(basename "$file")
      pathName=$(dirname "$file")
      userName=$(basename "$pathName")
      if test $userName = $prevUser
      then
        cat $file >> "$pathName"/"$userName"."$fileExtn"
        rm -f $file
      else
        fileExtn=`echo $fileName | sed -e 's/.*[.]//g'`
        mv "$file" "$pathName"/"$userName"."$fileExtn"
        chmod 666 "$pathName"/"$userName"."$fileExtn"
      fi
      prevUser=$userName
    fi
  done
fi

#------------------------------------------------------------------
# process file with SubdueGL
# next two lines are commented out becuase all file are being processed
#------------------------------------------------------------------
#for file in $(find $njsHome/webfiles/workspace/ -iregex '.*\.\([fch]\|cpp\|java\)'); do
#for file in $(find $njsHome/webfiles/workspace/ \( -ipath '.*cpp'  -o -ipath '.*java' -o -
  ipath '.*[fch]' \)); do

 for file in $(find $njsHome/webfiles/zip$index/workspace/ -name '*.*'); do
  echo $file
  echo "file"
  if [ -f $file ]; then
    name=$(basename "$file")
    fn=`echo $fname | sed -e 's/.*[.]//g'`

    if text $fn = f
    then
      #---------------------------------------------------------------
      # this fileType variable is for Parser param 4=FORTRAN 0=C/C++/Java
      #---------------------------------------------------------------
      fileType=4
    else
      fileType=0
```

```
    fi

    #------------------------------------------------------------------
    # each file is copied to home directory for processing with Parser.
    # Graph files (.g) created by Parser are moved to workspace
    # sub-directory. Then processed with SubdueGL
    #------------------------------------------------------------------
    cp $file $njsHome/bin
    # $njsHome/bin/GenGo $njsHome/bin/$name
    # /usr/lib/java Parser $name $fileType
    $njsHome/bin/Parser $njsHome/bin/$name $fileType
    mv $njsHome/bin/$name* $njsHome/webfiles/zip$index/workspace/.
    $njsHome/bin/Subdue -gg -norecursion -novariables -save -exhaust
  $njsHome/webfiles/zip$index/workspace/$name.g
  fi
done

#---------------------------------------------------------------
# some variables declared and initialized for computing results.
# all files compared with Submatch in nested FOR loops below.
# results written to result.dat file and itmp file
#---------------------------------------------------------------
result="0"
count="0"
#echo "Results" > $njsHome/bin/result.dat
echo "   " > $njsHome/bin/result.dat

for firstFile in $(ls $njsHome/webfiles/zip$index/workspace/*.s); do
  if [ -f $firstFile ]; then
    fileName1=$(basename "$firstFile")echo "   " > $njsHome/bin/result.dat
    for secondFile in $(ls $njsHome/webfiles/zip$index/workspace/*.s); do
      if [ -f $secondFile ]; then
        fileName2=$(basename "$secondFile")
        if test $fileName1 != $fileName2
        then
          result=`$njsHome/bin/Submatch $firstFile $secondFile`

          #------------------------------
          # strip .g.s from file name
          # added 10/31/2006 by NJS
          #------------------------------

         file1=`echo $fileName1 | sed -e 's/.g.s//g'`
          file2=`echo $fileName2 | sed -e 's/.g.s//g'`

          echo "$file1  $file2  $result" >> $njsHome/bin/result.dat
```

```
            count=`expr $count + 1`
            if test $count -eq 1
            then
              echo $result > $njsHome/bin/itmp
            else
              echo $result >> $njsHome/bin/itmp
            fi
          fi
        fi
      done
      mv $firstFile $njsHome/webfiles/zip$index/archive/.
  fi
done

#-------------------------------------------------------------
  # sort the file result.dat based on percent match in reverse order
  # added 10/31/2006 by NJS
  #-------------------------------------------------------------
  cat $njsHome/bin/result.dat | sort -nr +2 -3 > $njsHome/bin/result_sort.dat
cat $njsHome/bin/result_sort.dat > $njsHome/bin/result.dat

#---------------------------------------------------------
# itmp file is striped of % sign using sed editor
# and output written to itmp_clean file
# itmp is overwriten with concatenation of count and itmp_clean
# itmp is processed by stdev_web for calculating
# mean and standard deviation
#---------------------------------------------------------
sed "s/%//" $njsHome/bin/itmp > $njsHome/bin/itmp_clean
echo $count > $njsHome/bin/icount
cat $njsHome/bin/icount $njsHome/bin/itmp_clean > $njsHome/bin/itmp
echo "   " >> $njsHome/bin/result.dat
$njsHome/bin/stdev_web >> $njsHome/bin/result.dat
cp $njsHome/bin/result.dat $njsHome/bin/result_"$zipfileName".dat
#mv result.dat resultExaust_"$zipfileName".dat

#-------------------------------------
# read user name and email from dat file
# and send email to user
#-------------------------------------
while IFS=: read f_name l_name e_mail file_name
do
  echo "$f_name $l_name," > $njsHome/bin/user_name.dat
  echo " " >> $njsHome/bin/user_name.dat
  cat $njsHome/bin/user_name.dat $njsHome/bin/result_"$zipfileName".dat >
  $njsHome/bin/result.dat
```

```
    mailx -s "Results of SubMatch" $e_mail < $njsHome/bin/result.dat
  done < $njsHome/webfiles/zip$index/$zipfileName.dat

  #----------------------------------------
  # cleanup. Archive dat file and
  # delete directories created at start
  #----------------------------------------
  mv -f $zipfile.dat $njsHome/webfiles/archive
  rm -rf $njsHome/webfiles/zip$index
  #rm -f $njsHome/webfiles/zip$index/archive/*
done
```

VITA

Neeraja Joshua Samuel

Candidate for the Degree of

Master of Science

Thesis: COMPARATIVE STUDY OF EXISTING PLAGIARISM DETECTION SYSTEMS AND DESIGN OF A NEW WEB BASED SYSTEM

Major Field: Computer Science

Biographical:

Personal Data: Born in New Delhi, India, the daughter of Mulkh Raj and Santosh Khurana

Education: Graduated from St. Mary's High School, New Delhi, India in March 1982; received Bachelor of Commerce degree from University of Delhi in March 1985. Completed requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December, 2006.

Experience: Worked as a Sr. Office Assistant in Indian Airlines, India, from 1985 to 1992. Worked as Computer Programmer/Analyst in healthcare and energy industries from 1996 to 1999. Worked as Teaching Assistant in Oklahoma State University from 2004 to 2005.