ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

Mümine KAYA

DEVELOPMENT OF A SOFTWARE ON DISTANCE EDUCATION APPLICATIONS FOR COMPILATION AND PLAGIARISM DETECTION OF C PROGRAMMING LANGUAGE ASSIGNMENTS

DEPARTMENT OF COMPUTER ENGINEERING

ADANA, 2011

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEVELOPMENT OF A SOFTWARE ON DISTANCE EDUCATION APPLICATIONS FOR COMPILATION AND PLAGIARISM DETECTION OF C PROGRAMMING LANGUAGES ASSIGNMENTS

Mümine KAYA

MSc THESIS

DEPARTMENT OF COMPUTER ENGINEERING

We certify that the thesis titled above was reviewed and approved for the award of degree of the Master of Science by the board of jury on 03/08/2011.

…….....	…….....	….…….....
Asst. Prof. Dr. S. Ayşe ÖZEL	Prof. Dr. Mehmet TÜMAY	Assoc. Prof. Dr. Zekeriya TÜFEKÇİ
SUPERVISOR	MEMBER	MEMBER

This MSc Thesis is written at the Department of Institute of Natural And Applied Sciences of Çukurova University.
Registration Number:

Prof. Dr. İlhami YEĞİNGİL
Director
Institute of Natural and Applied Sciences

ABSTRACT

MSc THESIS

DEVELOPMENT OF A SOFTWARE ON DISTANCE EDUCATION APPLICATIONS FOR COMPILATION AND PLAGIARISM DETECTION OF C PROGRAMMING LANGUAGE ASSIGNMENTS

Mümine KAYA

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF COMPUTER ENGINEERING

Supervisor : Asst. Prof. Dr. Selma Ayşe ÖZEL
Year: 2011, Pages: 145
Jury : Prof. Dr. Mehmet TÜMAY
: Assoc. Prof. Dr. Zekeriya TÜFEKÇİ
: Asst. Prof. Dr. Selma Ayşe ÖZEL

In this study, a module for compilation and plagiarism detection of C programming language assignments on Moodle distance education software for Çukurova University Department of Computer Engineering was developed. Rapid improvement of information technology has caused the distance learning types to vary and it has enabled the internet based distance learning to be popular.

This study uses Moodle, which has an utmost position to meet the information needs of students in distance learning programs and on virtual class which provides the participants with the information sources converted onto electronic format. In this study we have developed a software for Modle system that compiles and runs C programming assignments that are loaded to the Moodle system of our department. To compile and run C programming assignments, GCC compiler was employed. In this study, a module was included into the Moodle system to increase effectiveness of grading programming assignments, since programming assignments are important in Computer Engineering education. In addition to including online C compiler, we have developed a new plagiarism detection algorithm for determining similarities between the programming assignments that are submitted by the students. Our source code plagiarism algorithm is in fact a hybrid algorithm which employs Greedy String Tiling algorithm similar to JPlag source code plagiarism software and computes word similarity, source line similarity, and comment line similarity similar to CodeMatch plagiarism detection software. The experimental analysis made over the real programming assignments of our Computer Engineering Department showed that our hybrid system is successful in detecting source code similarities for C programming language assignments.

Key Words: Moodle, Plagiarism, Programming Language, Distance Education, GCC

ÖZ

YÜKSEK LİSANS TEZİ

UZAKTAN EĞİTİM UYGULAMALARINDA C PROGRAMLAMA DİLİ ÖDEVLERİNİN DERLENMESİ VE İNTİHAL TESPİTİ İÇİN BİR YAZILIM GELİŞTİRİLMESİ

Mümine KAYA

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Yrd. Doç. Dr. S. Ayşe ÖZEL
Yıl: 2011, Sayfa: 145
Jüri : Prof. Dr. Mehmet TÜMAY
: Doç. Dr. Zekeriya TÜFEKÇİ
: Yrd. Doç. Dr. S. Ayşe ÖZEL

Bu çalışmada Çukurova Üniversitesi Bilgisayar Mühendisliği Bölümü için Moodle Uzaktan Eğitim yazılımına C programlama dili ödevlerinin derlenmesi ve intihal tespiti için bir eklenti geliştirildi. Hızla gelişmekte olan bilgi teknolojisi, uzaktan öğretim türlerinin çeşitlenmesine ve internete dayalı uzaktan öğretimin yaygınlaşmasına neden olmuştur.

Bu çalışmada, öğretim sistemi içinde önemli bir yeri olan Moodle programının, uzaktan öğrenim gören bireylerin bilgi ihtiyaçlarını karşılamak için geliştirdikleri hizmet türleri üzerinde durulmakta ve bilgi kaynaklarının elektronik ortama aktarılarak, kullanıcılara sunulduğu sanal sınıflar anlatılmaktadır. Bu çalışmada biz Moodle yazılımı için bizim bölümümüzün Moodle sistemine yüklenen C programlama ödevlerini derleyen ve çalıştıran bir yazılım geliştirdik. C programlama ödevlerini derlemek ve çalıştırmak için GCC derleyici kullanıldı. Bu çalışmada, eklenti Moodle sistemine programlama ödevlerini notlandırmanın etkinliğini arttırmak için eklendi; çünkü programlama ödevlerinin Bilgisayar Mühendisliği eğitiminde önemli bir yeri ovardır. Online derleyici dâhil olmak üzere ek olarak, biz öğrenciler tarafından gönderilen programlama ödevleri arasındaki benzerliği belirleyen yeni bir intihal tespit algoritması geliştirdik. Bizim kaynak kod intihal algoritmamız gerçekte JPlag'a benzer Hırslı Metin Eşleme Algoritması, CodeMatch'e benzer Kelime Eşleme, Kaynak Satırı Eşleme, Yorum Satırı Eşleme ve Anlamsal Satır Eşleme Algoritmalarının kullanıldığı bir hibrit algoritmadır. Bilgisayar Mühendisliği Bölümümüzün gerçek programlama ödevleri üzerinde yapılan deneysel analiz hibrit algoritmamızın C programlama dili ödevleri için kaynak kodu benzerliklerini tespit etmede başarılı olduğunu gösterdi.

Anahtar Kelimeler: Moodle, İntihal, Programlama Dili, Uzaktan Eğitim, GCC

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Asst. Prof. Dr. Selma Ayşe ÖZEL, for her supervision guidance, encouragements, patience, motivation, constructive and lead -in and useful suggestions and her valuable time for this work. I am grateful to her for all the things that I have learned under her supervision. This thesis would not have been possible without the help of my advisor. She never deprived her help at any stage of my work. I would like to thank my eternal gratitude to her to support me for selecting the thesis about "Development Of A Software On Distance Education Applications For Compilation And Plagiarism Detection Of C Programming Language Assignments", to share her knowledge, suggestions, experience and opinion during all the time of my studies, to give me some guidance for my thesis. I will always take my estimable advisor, Asst. Prof. Dr. Selma Ayse ÖZEL as my model during my career.

I would like to thank members of MSc thesis jury, Prof. Dr. Mehmet TÜMAY for his experience and suggestions and for his lead-in and affirmative contribution at every stage of my work. I would like to thank members of MSc thesis jury, Assoc. Prof. Dr. Zekeriya TÜFEKÇİ, for his constructive and lead-in suggestions and corrections.

I would also like to thank Çukurova University Faculty of Engineering and Architecture Department of Computer Engineering Chair sincerely. Additionally I would like to thank post graduate students, Melis ÖZYILDIRIM, Sena ÖZTÜRK and Erhan TURAN, who I was supported by them during my studies and for their help and encouragement.

Last but not the least important, I would like to thank my family to stand by me since the beginning of my education life and my career, to vitalize to me with their presence, to support me with a huge patience always, not to deprive their physical and moral help from me. I would like to thank my dear father, Lawyer İsmet KAYA, and my dear mother, Cavidan KAYA, to be my lifeblood with their presence. I would like to thank to my sweetie sister, Agricultural Engineer CPD Emine KAYA, for her support and encouragements on my studies.

CONTENTS **PAGE**

LIST OF TABLES **PAGE**

LIST OF FIGURES **PAGE**

ABBREVIATIONS

CMS	: Course Management System
CNG	: Character N-Gram- Based
COSE	: Creation of Study Environments
CPD	: Copy/Paste Detector
EVE	: Essay Verification Engine
FLE3	: Future Learning Environment
GCC	: GNU Compiler Collection
GNU	: GNU's Not Unix!
GPL	: GNU General Public License
GBSCA	: Greedy String Tiling Based Comment Lines Matching Algorithm
GBSMA	: Greedy String Tiling Based Source Lines Matching Algorithm
GST	: Greedy String Tiling
HTML	: Hyper Text Markup Language
HTTP	: Hyper Text Transfer Protocol
ITFS	: Instructional Television Fixed Service
LCS	: Longest Common Subsequence
LMS	: Learning Management Systems
MOSS	: Measure Of Software Similarity
MOODLE	: Modular Object-Oriented Dynamic Learning Environment
OLAT	: Online Learning And Training
OS	: Operating System
PERL	: Practical Extraction and Report Language
PHP	: Personal Home Page
RKR-GST	: Running-Karp-Rabin Greedy-String-Tiling
SEM	: Semantic- Based
SID	: Shared Information Distance
SQL	: Structured Query Language

SSMA	: Semantic Sequence Matching Algorithm
STYLE	: Stylometric-Based
SVM	: Support Vector Machines
SYN	: Syntax-Based
WMA	: Word Matching Algorithm
VHDL	: Very High Speed Integrated Circuit Hardware Description Language
VLE	: Virtual Learning Environments
YAP	: Yet Another Plague

1. INTRODUCTION

The rapid development and the widespread use of the Internet and the information technology have made the Internet to become the world's largest source of information which is used by millions of people every day. Rapidly changing technology and evolving market conditions have changed the education system as well. More educational opportunities have emerged with fewer budgets. As a result of this, distance education tools have increased.

Distance education, which is also called distance learning, has come into being for centuries. Some recent definitions have focused on it as a new development, involving advanced technology. A few definitions have even looked for to define it in terms of a single technology (North, 1993). Others have displayed it simply as a recent development of the class into a remote location (Long-distance teaching, 1990). Such definitions have been too restrictive and failed to recognize the actual needs of distance education users. A better definition has been provided by Ian Mugridge (1991), as “it is a form of education in which there is normally a separation between teacher and learner and thus one in which other means - the printed and written word, the telephone, computer conferencing or teleconferencing, for example - are used to bridge the physical gap."

One of the most useful distance education tools is Moodle (http://moodle.org/), which is a free web application that educators can use it to create effective online learning sites, and students can use it to take all the information they need to be successful. It is a learning management system that enables lecturers to provide lecture notes, assignments, quizzes, discussion forums, etc. to students easily and effectively. It gives an opportunity to lecturers for online lesson delivery. Students use Moodle to participate classes from everywhere and they can submit their assignments easily to their lecturers.

On the other hands, some students use Internet and these distance education tools to make plagiarism easily. Plagiarism is described by the Honor Council (http://orgs.odu.edu/hc/pages/What_is_the_Honor_Council.shtml) as "the act of

passing off the ideas or writings of another as one's own." According to Hacker (2007) plagiarism is “using another author’s intellectual property - language, visuals, or ideas - in your own writing without giving proper credit”. Plagiarism can also be defined as turning in someone else's work as your own or changing words but copying the sentence structure of a source without giving credit (http://www.plagiarism.org/plag_article_what_is_plagiarism.html). To solve the problem of copying words, ideas or programs from someone else, a lot of new techniques have been developed. Plagiarism in computer code is also frequent since programming courses become popular in most of the undergraduate programs, and detection of source code plagiarism requires different tools than those found in textual document plagiarism. Significant research has been dedicated to find academic source-code plagiarism (Ahtiainen et al., 2007; Arwin and Tahaghoghi, 2006; Belkhouche et al., 2004; Cosma and Joy, 2006; Culwin et al., 2001; Clough, 2000; Gitchell and Tran, 1999; Verco and Wise, 1996; Niezgoda and Way, 2006; Mann and Frew, 2006; Prechelt et al., 2000; Schleimer et al., 2003; Vamplew and Dermoudy, 2005; Wiedemeier, 2002; Wise, 1993).

The aim of any plagiarism detection process is to ensure that copied material is detected, and that no student is unfairly accused of copying. Plagiarism detection of assignments submitted by students in most of the education programs is becoming harder, when the number of assignments submitted by students is large. Automated Plagiarism Detection Tools help to achieve this aim, but no tool is perfect, and no fully automatic plagiarism detection tools identify all cheaters. Some tools such as Moss (http://theory.stanford.edu/~aiken/moss/), JPlag (https://www.ipd.uni-karlsruhe.de/jplag/) and CodeMatch (http://www.safe-corp.biz/products_codematch.htm) are near perfect to detect source code plagiarism.

The plagiarism detection tools use different algorithms to detect similarities between source codes or documents. Some algorithms measure the similarity between token sequences such as Greedy String Tiling Algorithm (Prechelt et al., 2002), some of the other algorithms compute the metric based similarity between strings by constructing tiles (Yamamoto et al., 2002) such as YAP (Wise, 1993), some of them find similarity between Parse Trees (Son et al., 2006), and the others

calculate similarity between graphs such as GPLAG software (Mishne and Rijke, 2004).

One important algorithm that is employed for detecting similarities between source codes or documents is Greedy String Tiling Algorithm (Wise, 1993). It is based on tokenization of the source codes or documents. Tokenization is the process of breaking a stream of text up into words, identifiers, functions, phrases, symbols, or other meaningful elements called tokens. Tokenization is useful both in linguistics and in computer science, where it forms part of lexical analysis. Every programming language has different lexical structure and tokenization makes similarity comparison to be independent of programming languages. Greedy String Tiling has proposed for comparing pairs of files to determine the degree of their similarity. It is based on one-to-one matching of tokens, and is able to deal with transposition of substrings.

The reason for the selection of this topic in this thesis is the opinion that distance education is used by many educational institutions more frequently in the future. Day by day, the importance of distance education is understood better and the need for such software is increasing. Up till today, several methods and programs have been developed for detecting plagiarism and preventing from it. However, many of these methods are proposed for text document plagiarism (Clough, 2000; Chester, 2001; White and Joy, 2004; Monostori et al., 2001; Hoad and Zobel, 2003; Finkel et al., 2002; Bull et al., 2001). The algorithms of this type of plagiarism cannot be used directly for source code plagiarism.

The aim of this thesis is to design and develop an online system on which lecturers can load their lecture notes and assignments on the Internet, and they can check and grade the submitted assignments which are especially written in C programming language, students can follow the course content, they can upload their assignments and they can discuss the questions related to the course with their lecturers. The desired results of this study are incorporating an online compiler for C programming language assignments and detecting the similarities of the submitted assignments with a string matching algorithm within the Moodle distance education tool. In the current Moodle software, compilation of programming assignments and plagiarism detection modules do not exist. This study fills this gap in the literature.

There are several systems for source code plagiarism detection; such as YAP, YAP2, YAP3 (http://pam1.bcs.uwa.edu.au/~michaelw/YAP.html), Sherlock (http://sydney.edu.au/engineering/it/~scilect/sherlock/), CPD (http://pmd.sourceforge.net/cpd.html), SID (http://genome.math.uwaterloo.ca/SID/), Plague (Zeidman, 2010), Moss (http://theory.stanford.edu/~aiken/moss/), JPlag (https://www.ipd.uni-karlsruhe.de/jplag/) and CodeMatch (http://www.safe-corp.biz/products_codematch.htm). However these tools are standalone and most of them run on their own distant server which causes plagiarism detection to be vulnerable to network or server failures.

In this study, our aim is to design and implement an efficient source code plagiarism detection system that runs our own server and within our Moodle system. For this purpose, we examined the existing plagiarism detection software and observed that JPlag and CodeMatch perform quite well. So, our new algorithm is a combination of JPlag and CodeMatch such that, we employ Greedy String Tiling from JPlag and Source Line Matching, Comment Line Matching, Word Matching, and Semantic Sequence Matching algorithms from CodeMatch. Also, our algorithm can directly process the homeworks submitted to Moodle in our own local server and it is not affected from the network failures.

2. PRELIMINARY WORK

2.1. Distance Education

There is difficulty in finding a universal definition of distance education; “The ideas surrounding the educational endeavor are somewhat similar” (Keegan, 1996).

According to Rudolf Manfred Delling (1966), who is a German historian and bibliographer, distance education is a planned and systematic activity which is made up of the choice of teaching and it supports student learning. It is a bridge between student and teacher through at least one appropriate technical medium (Delling, 1966).

To Dohmen (1967), a former director of the German Distance Education Institute (DIFF) at Tubingen, distance education is a systematically organized form of self-study, which is made possible at a distance by means of media which can cover long distances. According to him, the opposite of ‘distance education’ is ‘direct education’ or ‘face-to-face education’ which is a type of education that takes place with direct contact between lecturers and students (Dohmen, 1967).

Peters (1973), who worked at DIFF in Tubingen, defines distance education as distance teaching which is a method of imparting knowledge, skills and attitudes. Distance education is an industrialized form of teaching and learning (Peters, 1973).

In the definition presented by Michael Moore (1973, 1977), distance teaching is described as the family of instructional methods in which the teaching behaviors are executed except for the learning behaviors, so that communication between the teacher and the student must be simplified by print, electronic, mechanical, or other devices (Moore, 1973; 1977).

Holmberg (1977) defined the term ‘distance education’ as the various forms of study at all levels which are not under the continuous, immediate supervision of lecturers present with their students in classes (Holmberg, 1977).

According to Garrison and Shale (1987), distance education refers that the majority of educational communication between teacher and students occurs non-

contiguously. It must involve two-way communication between teacher and students for the purpose of facilitating and supporting the educational process. It uses technology to mediate the necessary two-way communication (Garrison and Shale, 1987). With reference to his opinion distance education is "inexorably linked to the technology" (Garrison and Shale, 1987) and seems to be viewed as different from other forms of education, a factor which may contribute to course development and acceptance problems.

In 1988, Hilary Perraton defined the distance education as an educational process in which a significant proportion of the teaching is conducted by someone removed in space and/or time from the learner (Perraton, 1988).

In 1988, Shale states that "distance education is beset with a remarkable paradox - it has asserted its existence, but it cannot define itself" (Shale, 1988). According to Shale, "distance" is an incidental consideration and not a "defining criterion" for education.

In 1989, Barker et al. provided a definition of distance education that captured the emergence of telecommunication technologies. Telecommunications-based distance education approaches are an extension beyond the limits of correspondence study. The teaching and learning experiences for both instructor and students occur simultaneously. It is contiguous in time (Barker et al., 1989).

In 1990, Moore, the editor of *The American Journal of Distance Education*, provides another view of distance education, as arrangements for providing instruction through print or electronic communications media to person engaged in planned learning in a place or time different from that of the instructor or instructors (Moore, 1990).

Portway and Lane (1994) stated that the term ‘distance education’ refers to teaching and learning situations in which the instructor and the learners are geographically separated, and therefore, rely on electronic devices and print materials for instructional delivery. Distance education includes distance teaching – the instructor’s role in the process – and distance learning – the student’s role in the process (Portway and Lane, 1994).

According to Moore and Kearsley (1996), distance education is defined as planned learning that normally occurs in a different place and requires a well-defined system of delivery that includes modified teaching techniques, alternative modes for communication, including, but not limited to technology, as well as alternative administrative and organizational components (Birnbaum, 2001).

In a book entitled, *Distance Learning: Principles for Effective Design, Delivery and Evaluation,* Mehrotra, Hollister and McGahey (2001), defined distance education as: any formal approach to instruction in which the majority of the instruction occurs while educator and learner are not in each other's physical presence.

Lastly, Picciano's (2001) definition of distance education, as cited by Birnbaum, states, distance education uses three current and popular forms of media; broadcast television, two-way videoconferencing, and asynchronous learning networks (Birnbaum, 2001).

It should be apparent how one's definition of distance education could potentially shape an emerging theory of distance education. It is also important to remember that as technology advances it results in new ideas of distance education however the underlying concept of distance education remains the same, which is to educate individuals in a nontraditional environment through a variety of media.

2.1.1. History of Distance Education

Distance education needs a reliable means of communication between students and lecturers. Therefore, the history of distance education begins at the point where a reliable communication method is established. Most historians date distance education to the 18th century, when a few lecturers began to offer what were called correspondence courses. Distance education dates to at least as early as 1728, when "an advertisement in the Boston Gazette named 'Caleb Phillips, teacher of the new method of Short Hand" was searching students for lessons to be sent weekly (Holmberg, 2005). But technology-based distance education might be best linked to the introduction of devices, which are using both sight and sound, into the schools in

the early 1900s. The first catalog of instruction films appeared in 1910 (Reiser, 1987) and in 1913. Thomas Edison declared that, because of the invention of film, "Our school system will be completely changed in the next ten years" (Saettler, 1968). Instructional media were introduced into many extension programs by 1920 in the form of slides and motion pictures just as they were in the classroom. The introduction of television as an instructional medium appears as an important entry point. It marks parallel paths for correspondence study and instructional media. In 1932, seven years before television was introduced at the New York World's Fair, the State University of Iowa began experimenting with transmitting instructional courses. The certain success of audio-visual tools generated a renewed interest in using it in the schools (Reiser, 1987). Most of these studies were directed at understanding and generating theory on how instructional media affected classroom learning.

Although instructional television would never realize what many thought was its potential, it had limited success. It established a foothold in the minds of educators unlike instructional radio. In one of the earliest education, Gayle Childs concluded that television is not an instructional method, but an instrument for transmitting instruction in 1956.

In the late 1960s and early 1970s, microwave technology developed. So its costs went down, and universities began to set up microwave networks to take advantage of the Instructional Television Fixed Service (ITFS) authorized by the Federal Communications Commission (Carnegie Commission, 1972). The Carnegie Commission on Higher Education predicted that, by the year 2000, more than 80 percent of off-campus and 10 to 20 percent of on-campus instruction would take place through telecommunications (Carnegie Commission, 1972).

Distance education programs which exist today have a wide range of approaches (Jeffries, 2008). For example, independent study courses through computer networking, computer-delivered instruction, communication between students and instructors through electronic mail, class sessions, cluster groups, undergraduate and graduate degrees through cable networks, and video courses with texts and other collateral materials are these approaches.

In summary, the history of distance education shows a constant state of evolution. The historical view of distance education shows a stream of new ideas and technologies. History shows nontraditional education trying to blend with traditional education while meeting the changing learning theories and developing technologies.

2.1.2. Types of Distance Education

In general distance education is collected under two main headings: i) synchronous, and ii) asynchronous. Synchronous instruction requires that all students participate the classes at the same time. The method of delivery is usually interactive and includes telecourses, teleconferences, web conferencing, and internet chat sessions (Harrison, 2011). This form of distance education is less flexible than asynchronous because it requires students to be online at a specific time.

Asynchronous instructions do not require simultaneous participation of all students in the class, so it is more flexible. This form of instruction gives students the freedom to interact with the material and instructor at a time that is convenient for them (Harrison, 2011). However, the one drawback is that students are responsible for their own motivation in getting the work done. Some examples of asynchronous distance learning include online courses, videotaped courses, and correspondence courses. This tends to be more popular among students who are also working professionals.

Creating a virtual campus and allowing to asynchronous training are among the most important advantages of distance education (Gündüz M., 2000). Adults can continue to their higher education with these types of programs which eliminate time, distance and physical constraints. Employees have a second chance, an alternative to traditional education for in-service training (Gündüz M., 2000). Rapid improvement of information technology caused the distance learning types to vary and it enabled the Internet-based or Web-based distance learning to be common.

Internet-based distance learning model is described as transmitting educational content using text, image, video, and audio files over the Internet online or offline (Odabaş, 2004). Internet-based distance education has become a specific

focus for at least three reasons (The Institute For Higher Education Policy): First, Internet is quickly becoming the predominant technology in distance education, because of its increasing telecommunications bandwidth capabilities. Second, Internet-based distance education allows the teaching/learning process to occur "at any time and any place." The ability to provide interactive learning activities, which are occurring at different times, has become the signature characteristic of this technology. Third, Internet-based distance education is, in many ways, fundamentally different from traditional classroom-based education.

Internet-based distance education can offer a bright learning environment created through different teaching strategies, activities, and technologies: It is conducted exclusively in an online format. It is taken as part of an online certificate or degree-granting program. It is offered to students with limited or no access to physical classroom locations or comparative academic offerings. Internet-based distance education creates academic opportunities for students who may not have been able to otherwise access educational resources. It delivers education to students for whom convenience and flexibility are paramount considerations. Internet-based distance education must have the following features (Al and Madran, 2004):

- The identification and management of users,
- Preparation of online course contents,
- Managing courses,
- Monitoring and analysis of student behaviors,
- Assessment of students' achievement status,
- The creation and management of interactive communication media.

2.1.3. The Purposes of Distance Education

Many educational institutions have begun to create solutions to their increasing educational needs through the development of distance education programs. Since distance education allows educational path to be determined by educators and students that are separated with physical distance and technology (audio, video, data and written text). It is a form of education that students, teachers

and teaching materials in different geographies are brought together through communication technology (Gündüz et al., 2007).

The main goal of distance education is to overcome barriers of place and time. Learners may live in isolated, rural areas and have no access to education. Other learners may have ready access to a college, but that college might not offer the course of study needed by that learner. Distance learning allows education to reach those who are not able to physically attend courses in universities (http://education.stateuniversity.com/pages/1917/Distance-Learning-in-Higher-Education.html).

One of the important purposes of distance education is delivering teaching, often on an individual basis, to students who are not physically present in a traditional educational setting such as a classroom. Also it provides access to quality education and equity in educational opportunities for those who otherwise would have been denied (http://education.stateuniversity.com/pages/1917/Distance-Learning-in-Higher-Education.html).

Distance education increasingly allows learners to participate to the courses at a time that is most suitable for their schedule. Electronic education tools, formerly used only in distance learning, are increasingly being used in both on- and off-campus courses. Using video, audio, active learning, simulations, and electronic advances can overcome problems encountered by learners who do not adapt to just one learning style (http://education.stateuniversity.com/pages/1917/Distance-Learning-in-Higher-Education.html).

Distance education can help students advance toward a degree more quickly. It allows students to enter college with credits and obtain a higher education degree in less time (http://education.stateuniversity.com/pages/1917/Distance-Learning-in-Higher-Education.html).

2.1.4. The Failure of Distance Education

Besides the advantages of distance education, there are various disadvantages (http://web.firat.edu.tr/uzem/egitim.php):

- Lack of eye contact between the students and the lecturers,
- Teachers are unable to control their students,
- Costly and complex technology requirement,
- Immediate feedback is not offered,
- Social isolation,
- Needed more time to prepare course content,
- All the necessary courses cannot be offered online,
- Not enough knowledge about how effective distance education is,
- The difficulty of transferring for lecturers the course content to the online environment,
- Students are not given the opportunity to improve oral communication skills,
- Students and instructors need to take technical training and support.

2.1.5. Distance Education Tools

Computers and computer networks are rapidly becoming the preferred long-distance communication tool, and they are evolving as a major resource in distance education. The increasing communications capability of computers is being used in universities. There are many computer based distance education tools: Learnwise (http://www.learnwise.net/), ANGEL (http://www.angellearning.com/products/lms/), LON-CAPA (http://www.lon-capa.org/), Desire2Learn (www.Desire2Learn.com), Manhattan Virtual Classroom (http://manhattan.sourceforge.net/), MimerDesk (http://www.mimerdesk.org/), ATutor (http://atutor.ca/), Eduvo School (http://www.eduvo.com), CCNet (http://www.myccnet.com), Moodle (http://moodle.org/), eFront (http://www.efrontlearning.net), Edvance360 (www.edvance360.com), Sakai (http://sakaiproject.org/), Macromedia Breeze (http://www.adobe.com/resources/breeze/meeting/), Pearson LearningStudio (http://www.pearsonlearningsolutions.com/pearson-learning-studio/), Jusur (www.elc.edu.sa), BlackBoard (http://www.blackboard.com/), JoomlaLMS (http://www.joomlalms.com/), Scholar360 (www.scholar360.com), Fle3

(http://fle3.uiah.fi), OLAT (http://www.olat.org), SharePointLMS (http://www.sharepointlms.com), WebStudy Learning CMS (www.webstudy.com), ILIAS (http://www.ilias.de/docu/), Timecruiser Solution Suite (www.timecruiser.com), The rSmart Sakai CLE (http://www.sakaisandbox.com), BSCW (http://public.bscw.de/), WebCT (http://www.webct.com/content), Janison Toolbox (http://www.janison.com.au/JP/panels.aspx?Id=Home), IntraLearn SME (http://www.intralearn.com/), Claroline (http://www.claroline.net), Jenzabar Internet (http://www.jenzabar.com/), KEWL (http://kngforge.uwc.ac.za), Jones e-education (http://www.jones.com/companies/jones-e-education), Adobe Connect (http://www.adobe.com/products/adobeconnect.html), eTEA Learning Management System (www.etealms.com), COSE (http://www.staffs.ac.uk/COSE/), Dokeos (http://www.dokeos.com/). Among the others, WebCT (http://www.webct.com/content), Blackboard (http://www.blackboard.com/) and Moodle (http://moodle.org/) are the three well-known web-based learning management systems widely used in universities and higher education (Cheung, 2006).

WebCT (Zaina et al., 2001) is a Web-based distance education system with many teaching resources and with a simple user hierarchy. The system administrator has only some basic and essential tasks like initial course and teacher registration. Consequently, there is not excessive administrator dependence. Course management tasks, like registering students and course contents, checking test and statistics, are done by the teacher, called designer in WebCT. The system allows the teachers to change the page layouts, colors, font types, etc and they can choose texts or icon links. WebCT allows greater flexibility in designing course curriculum and study schedules, which is especially suitable for continuing education courses. With many communication and discussion features, WebCT facilitates active participation among instructors and students, and allows more varieties in designing learning materials and resources such as the use of multimedia.

The Blackboard Learning System (http://skillspark.ca/info/ MoodleandWebCTComparison.pdf) allows instructors to post course information and course materials, readings and assignments and provides functionality for basic

discussion and other collaborative tools. It is designed for teacher directed/centered delivery of content especially for lower level courses, and large classes.

Moodle is a course management system (CMS) - a free, open source software package designed by using sound pedagogical principles, to help educators to create effective online learning communities (Nozawa, 2011). Moodle tools focus on content delivery for course information.

For curriculum design, all three tools provide course templates that allow instructors to deliver course materials and to define study schedules and class activities, whereas WebCT and Blackboard allow more varieties in course contents and materials. WebCT also supports students to study off-line. For communication and discussion, all three tools provide discussion forums, chat rooms, electronic mail and file exchanges, whereas WebCT allows instructors to have control on discussions and chat rooms. WebCT and Blackboard also provide private folders and internal electronic mail for students, and allow students to make their own private notes. For performance assessment, all three tools provide assessment and grading functions, whereas WebCT and Blackboard support more varieties in assessment and grading. For course administration, all three tools allow uploading of student data and course data in batches, whereas Blackboard and WebCT also provide direct data interfaces with information systems. Moodle and WebCT have some similarities such as student enrolment in courses, discussion forums, quizzes, tests, uploading files (e.g. Word docs, PowerPoint, audio files), SCORM compliant courses, grades, course calendars, and monitoring student participation. The comparison of distance education tools is shown in detail in Table 2.1 (http://skillspark.ca/info/MoodleandWebCTComparison.pdf, 2011; Duzer and Munoz2005; Cole, 2005).

According to this comparison Moodle system was preferred in this thesis, because Moodle is free and open-sourced unlike WebCT and BlackBoard. The other reason is that Moodle has a very large community of people who develop new features and plugins. It has a self-assessment of submission and student peer review unlike the others.

Table 2.1. Comparison of Distance Education Tools (Duzer, Munoz; Cole, 2005)

Feature	**Moodle**	**WebCT**	**BlackBoard**
Bandwidth	All features can work on dial-up	Bandwidth hog, may time out	Bandwidth hog, may time out
Cost	Free	Cost	Cost
Learning Curve	Can use w/o manual or training. Provides excellent, easy to use "help" website.	Not as intuitive, many components need to configure initially. "Help" website is complicated.	Not intuitive, many components need to configure initially. "Help" website complicated.
Discussion	Photos	See posts at one time, not nested	See posts at one time, not nested
Tools	Blog, wiki, journal, glossary, workshop	Whiteboard	Whiteboard
Customization	Open Source, so can change locally	Need to request change from WebCT	Need to request change from BlackBoard
User Stats	Chart comparing students; number of visit per page	Time student spent on each page	Time student spent on each page
Discussion Forum	There are three types of Discussion Forums that you can choose depending on your needs.	One type of Discussion Forum where a student can reply to discussions without creating topics	One type of Discussion Forum where a student can reply to discussions without creating topics
Upload and share documents	Yes	Yes	Yes
Online Chat	Yes	Yes	Yes
Lessons with paths	Yes	Yes	Yes
Online Discussions	Yes	Yes	Yes
Create content online in HTML	Yes	Yes	No
Grade discussions / participation	Yes	Yes	No
Student peer review	Yes	No	No
Online Quizzes / Suveys	Yes	Yes	Yes
Online Gradebook	Yes	Yes	Yes
Student submission of documents	Yes	Yes	Yes
Self-assessment of submission	Yes	No	No
Student workgroups	Yes	Yes	Yes
Student Journals	Yes	No	No
Embedded glossary	Yes	No	No

2.1.6. Distance Education Applications in Turkey

Using Distance Education in universities has become widespread in Turkey recently. Some universities use distance education system for their all departments and faculties, while the others use it for only one department or a few departments which are needed. The universities and their departments using distance education systems in our country and their purposes of using distance education are given in Table 2.2. As shown in Table 2.2, the most widely used e-learning system is Moodle and it is mostly used to support formal education.

Table 2.2. Distance Education Applications In Turkey

UNIVERSITY	DEPARTMENT	TOOL	URL	PURPOSE
Atatürk University	Computer Engineering	Moodle	http://194.27.49.173/moodle/course/category.php?id=9	To Support Formal Education
Başkent University	Computer Engineering	Moodle	http://moodle.midas.baskent.edu.tr/course/category.php?id=22	To Support Formal Education
Bilkent University	Computer Engineering	Moodle	http://moodle.bilkent.edu.tr/ http://2010-2011-fall.moodle.bilkent.edu.tr/course/category.php?id=19	To Support Formal Education
Çankaya University	Computer Engineering	Moodle	http://webonline.cankaya.edu.tr/	To Support Formal Education
Çukurova University	Computer Engineering	Moodle	www.mmf.cu.edu.tr/mkaya/moodle	To Support Formal Education
Dokuz Eylül University	Computer Engineering	Moodle	http://lectures.cs.deu.edu.tr/	To Support Formal Education
Ege University	Computer Engineering	Moodle	http://sorubank.ege.edu.tr/~moodle/	To Support Formal Education
Eskişehir Osmangazi University	Computer Engineering	Moodle	http://dys.ogu.edu.tr/moodle/course/category.php?id=61	To Support Formal Education
Fatih University	Computer Engineering	Moodle	http://moodle.fatih.edu.tr/moodle/course/category.php?id=5	To Support Formal Education
Harran University	Computer Engineering	Moodle	http://eng.harran.edu.tr/moodle/moodle/course/category.php?id=3	To Support Formal Education
İstanbul Bilgi University	Computer Engineering	Moodle	http://courses.cs.bilgi.edu.tr/	To Support Formal Education
Middle East Technical University	Computer Engineering	Moodle	http://ocw.metu.edu.tr/course/category.php?id=3	To Support Formal Education
Pamukkale University	Computer Engineering	Moodle	http://cds.pau.edu.tr/course/category.php?id=177	To Support Formal Education
Sakarya University	Computer Engineering	Moodle	http://www.cs.sakarya.edu.tr/mc/	To Support Formal Education, Distance Education

Table 2.2. Con't.

UNIVERSITY	DEPARTMENT	TOOL	URL	PURPOSE
International Cyprus University	Computer Engineering	Moodle	http://ceng.ciu.edu.tr/moodle/	To Support Formal Education
Sabancı University	All Departments	WebCT	http://ab.org.tr/ab04/tammetin/138.doc	To Support Formal Education, Distance Education
Kadir Has University	All Departments	BlackBoard	http://blackboard.khas.edu.tr/index.html	To Support Formal Education, Distance Education
İstanbul University	All Departments	BlackBoard	http://uzem.istanbul.edu.tr	To Support Formal Education
Anadolu University	All Departments	WebCT, Adobe Breeze(Adobe Connect), BlackBoard	http://cevrimici.anadolu.edu.tr/ http://webct.anadolu.edu.tr/	To Support Formal Education, Distance Education
Hacettepe University	All Departments	BlackBoard	http://blackboard.hacettepe.edu.tr/	To Support Formal Education
Beykent University	3 Departments	Adobe Connect	http://connect.beyu.edu.tr/system/login	Distance Education
İstanbul Arel University	2 Departments	Perculus	http://www.etoplanti.sakarya.edu.tr/Perculus.aspx	Distance Education
İstanbul Aydın University	2 Departments	Adobe Presenter, Adobe Connect Pro Meeting	http://ue.aydin.edu.tr	Distance Education
Karabük Üniversitesi	9 Departments	Moodle	http://kbuzem.karabuk.edu.tr	Distance Education

2.2. Moodle

Moodle is an online course management system that is widely known and freely available. The word Moodle stands for “Modular Object Oriented Dynamic Learning Environment” and was created by Martin Dougiamas, a computer scientist and educator, who spent time supporting a Course Management System at a University in Perth, Australia (Cole et al., 2008).

Moodle is a software package that it is used to create Internet based courses and their Web sites. Moodle is being used in 212 countries, providing support for 77 languages, and has 44,276,245 registered users according to Moodle Statistics on Moodle web site in June 2011 (http://moodle.org/sites/). There are 53952 currently active sites that have registered from those countries. 4,619,589 registered courses and 1,111,602 teachers are on Moodle. Moodle is used by a variety of institutions and individuals, including universities, high schools, primary schools, government departments, healthcare organizations, military organizations, airlines, oil companies, homeschoolers, independent educators, and special educators (http://docs.moodle.org/20/en/Usage). As shown as in Figure 2.1. Moodle registrations for the last two months by version 1.9.x are higher than the other versions.

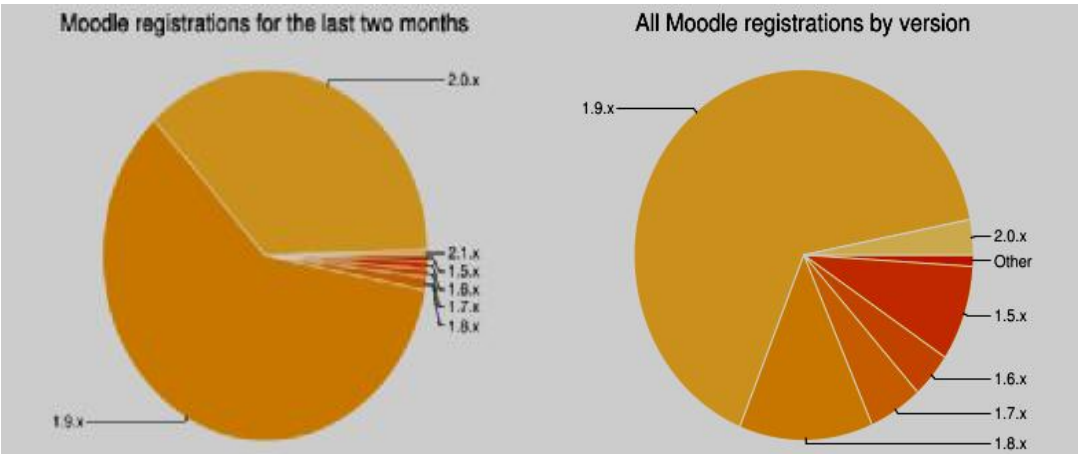


Figure 2.1. Moodle Statistics (http://moodle.org/)

Moodle has been used since 2002 as a distance education tool. It has various versions. Moodle is a PHP-based open-source online learning system. Courses set up in modules. Linux, Unix, Windows and Mac OS X operating systems are supported. Moodle, which is a course management system, is designed as a software package to help educators create online courses. This type of software packages are known as Learning Management Systems (LMS) or Virtual Learning Environments (VLE) (http://moodle.org/).

The Moodle environment includes a platform for several features including: class schedules, assignments, participant profiles, chats, wikis, an interactive glossary, e-mail, lessons that the teacher can create, quizzes, surveys, and workshops. Chats, forums, wikis, and workshops enable students to work collaboratively.

A number of programs, namely PHP, MySQL and Apache, are needed before the Moodle program is set up. PHP (www.php.net/) is a script language embedded into html codes, and it works in a server-side. An open source code that embedded into an html page is run on every visit and can be downloaded from the Internet. MySQL (www.mysql.com) is a database management system that can run in the background all the time and can respond to requests like a high performance Web server. The query language which is used to find the required data on MySQL Database is the SQL language. Apache (http://www.apache.org) is a Web server that is open source software, it is completely free and it has high performance.

The last version of Moodle is Moodle 2.1 and it is released on 1 July 2011. Support for 2.1.x will end on July 2012.

2.2.1. The Advantages of Moodle

Moodle is used as an open-source virtual learning environment mostly, because of its important features (Ajlan, Zedan, 2008; Barnard et al., 2007): Ease of use for system admins, application developers, course designers, faculty, and students; a rich set of resource and activity creation tools; availability of third party modules to extend the student experience; cohesiveness of the design and implementation; growing international contributions to the platform; easy to create

electronic portfolio for each student; allows for online/offline assignments, individual and group work, online assessment that can be graded or ungraded. Besides, Moodle costs nothing to download, and anyone can install it on as many servers as they want, because it is open-source and freeware unlike the other distance education tools. Moodle has a very large, active community of users who develop new features and enhancements. Moodle's design philosophy makes this a uniquely teacher-friendly package that represents the first generation of educational tools that are truly useful. These three advantages - open source, social constructivism, and community mentioned features make Moodle unique in the Course Management System space (Cole, 2008).

Moodle is designed as a modular structure so it has the flexibility to add or remove functionality at many levels. It is capable of easy updating to another version. It performs this process with its own version detection mechanism.

Moodle allows repairing the related database files. It is focused on system security. This feature applies to all versions. It provides security as controlling all forms, ensuring data consistency and managing with encrypted cookies.

Moodle is suitable for 100% online course and it can be also used for supporting face-to-face education. It needs as a simple web browser such as Internet Explorer, Firefox, etc. Courses can be divided into categories, so searches can be done on them.

The other advantages of Moodle are (http://docs.moodle.org/20/en/ Moodle_manuals):

- Immediate access to important course information and material.
- Additional learning tools, such as forums and quizzes (self tests).
- The freedom to access course materials at any time and away from campus.
- Applications such as wikis, the journal, the database and the glossary are perfect for collaborative activities.
- Teachers from within a school can network resources.
- Marking is made easy.
- Moodle tracks to the minute when a student has viewed a document, how long they have spent in a forum, when they uploaded an assignment.

2.3. Plagiarism

Plagiarism is the act of imitating or copying or using somebody else's creation or idea without permission and presenting it as one's own (Kustanto and Liem, 2009).

In terms of software plagiarism, Halstead (1977) metrics were used on Fortran programs in 1976 and only programs are considered to be plagiarisms where all four values coincide. These four values are number of unique operators *1*, number of unique operands eta_2, total number of operators N_1, total number of operands N_2. After 1977, Donaldson et al. (1981), Grier (1981), Berghel and Sallah (1984), Faidhi and Robinson (1987) introduced a much larger number of similarity metrics to improve performance.

Plagiarism has occurred for thousands of years in various forms. Online plagiarism has become difficult to detect (Redshield, 2011). As physical property is protected by laws, ideas also are guarded by legal system. Unfortunately, the theft of an idea is more difficult to track then stolen physical property. Our ability to detect and prevent plagiarism is extremely important for various industries, including publishing, music, research, education, media and finance.

One way that plagiarism is easily avoidable is through the proper use of quotations (www.mtlsd.org/highschool/stuff/10%20plagiarism%20tutorial.ppt*)*. Crediting other authors with their own ideas goes a long way in using their ideas to further his/her work. The internet makes plagiarism extremely easy since it is easy to find and use other people's work. It is as simple as copy-and-paste. This causes many people to plagiarize without thinking about the legality of the act. This reality is changing because new software solutions are developed on detecting and preventing plagiarism. Some plagiarism software allows individuals in various industries to efficiently search the web for copy content. These solutions provide the tools to detect plagiarized content on the internet. As this technology continues to advance, the act of plagiarism will be decreased.

Wilhoit (1994) argues that few students come to university understanding the rules of academic writing. Howard (1995) argues that some forms of inadvertent

plagiarism are learning strategies adopted by students who have not been taught how to make proper intellectual use of resources.

Studies of student plagiarism have found clear differences between students' and teachers' thinks of plagiarism. Generally students do not think that minor forms of cheating are wrong (Franklyn-Strokes and Newstead, 1995). Students plagiarize to get a good grade from the courses (Ashworthet al., 1997). Zobel and Hamilton (2002) found that financial problems were among the most common reasons given for committing plagiarism. There are also cultural issues such as respect for authority that can lead to reluctantly plagiarism (Zobel and Hamilton, 2002). The most common factors of plagiarism are the stress to obtain good grades, inefficient institutional deterrents and forgiving lecturers (Davis et al., 1992). When the assignment is evaluated to be of little consequence and others in the class are cheating the institution does not make plagiarism a high priority then students are more likely to plagiarize (Ashworth et al., 1997). Zobel and Hamilton (2002) found that when students find themselves with difficulties they preferred toconsult to their friends for help.

Academic dishonesty is lower where there is a high perception of being caught (McCabe et al., 2001). There is clearly an institutional responsibility to detect and deal with plagiarism. Yet, Zobel and Hamilton (2002) find that departments are often inadequately prepared to deal with plagiarism. They argue that the first step in avoiding plagiarism is arriving at staff agreement on what generates plagiarism.

2.3.1. Types of Plagiarism

Plagiarism in programming assignments is an unavoidable issue for most academic teaching programming. In the Internet, the rising number of articles and text books are common sources that are used by students to obtain material, and these facilities make it easier for students to plagiarize (Gomes, 2006). It is revealed that some students use the internet to hire expert coders to implement their programming assignments (2006). Bull et al. (2001) and Culwin et al. (2001) have carried out surveys on academics to determine the prevalence of plagiarism and have evaluated

the performance of free-text and source-code plagiarism detection software respectively.

When a piece of work is copied partially or entirely from one or more sources it is called a text plagiarism (Carbajosa and Bausela, 2011). Even if the source is acknowledged and referenced, it is still considered to be plagiarism (Carbajosa and Bausela, 2011). Plagiarized text often demonstrates a sudden change of style, and tone from an editor's usual style and may appear more advanced in grammar and vocabulary. Plagiarized material may contain unexplained acronyms or technical jargon that has been described in an earlier part of the plagiarized document (http://en.wikipedia.org/wiki/Wikipedia:Plagiarism). Because plagiarized material was written for other purposes, it is often slightly off topic, or un-encyclopedic in tone. An editor who plagiarizes multiple sources will appear to change writing style abruptly (http://en.wikipedia.org/wiki/Wikipedia:Plagiarism#Text_plagiarism).

Source Code Plagiarism is one form of academic dishonesty, which is often done by students in programming classes. In a large class, detecting plagiarism manually is both difficult. Around the world, many millions of programming exercises are submitted every year by Computer Science students. In most cases, instructors think that a few of these programs are in fact copies of programs supplied by somebody else. This kind of plagiarism is called Source Code Plagiarism. Source code plagiarism in programming assignments can occur when a student re-uses source code written by someone else by obtaining the source code either with or without the permission of the original author and knowingly or without knowing not properly acknowledging the source code and submits it as his/her own work (Cosma and Joy, 2006).

In source code assignments, students are required to acknowledge the source and authorship of the source code that was not written by them, within the program source code and in the appropriate documentation (Cosma and Joy, 2006). In programming assignment, plagiarism does not necessary only involve copying the source code but if someone include comments, program input data and variable names that can also be considered as plagiarizing (Iaqat and Aijaz, 2011). Generally, plagiarism in coding is hard to be detected because of the similar source codes used

for the same programs. Students copy all or part of a source code from different sources and submit the copy as their own work. This includes students who collaborate and submit similar assignment.

When a lecturer in a programming course gives same assignment to all students, then all students have to work on same assignment. So it is the possibility that some students write source code by their own, and the other students just take the code from them and alter it like changing of variable names, changing the order of statements, functions and submit it. These types of modifications in source code are very difficult to catch. There are two types of source code modification: i) Lexical change, and ii) Structural change. It is very easy to do lexical change. One can do lexical change without any knowledge of programming language by using just a simple editor. Structural changes cannot be done without knowingany programming language (Goel et. al., 2008; Hage et. al., 2010).

2.3.2. Plagiarism Detection Tools

Plagiarism detection programs have been around for a number of years. Some of them are used to detect text plagiarism as shown in Table 2.3; some of them are used to detect source code plagiarism. These tools are shown in Table 2.4.

Plagiarism has always been a major problem in the educational process. Therefore, Source Code Plagiarism Tools are important for programming assignments. Plague, the YAP programs (YAP, YAP2, and YAP3), SID, Moss, CodeMatch and JPlag are widely used source code plagiarism tools.

The Plague (Zeidman, 2010) program was developed by Geoff Whale at the University of New South Wales. Plague uses an algorithm that creates a structure-metric, based on matching code structures rather than matching the code itself. The idea is that two pieces of source code that have the same structures are likely to have been copied.

Table 2.3. Text Plagiarism Detection Tools

TEXT PLAGIARISM TOOLS	URL
CopyCatch, CopyCatch Gold	http://cflsoftware.com/?page_id=42
Glatt Plagiarism Services Inc	http://www.plagiarism.com
WordCheck Keyword Software	http://www.wordchecksystems.com
COPS	http://ir.shef.ac.uk/cloughie/papers/plagiarism2000.pdf
SCAM	http://infolab.stanford.edu/~shiva/SCAM/plag.html
YAP3	http://pam1.bcs.uwa.edu.au/~michaelw/YAP.html
CHECK	http://www.ics.heacademy.ac.uk/resources/assessment/plagiarism/detectiontools_comparison_source_free_text_comparison.html
VAST/PRAISE	http://www.ics.heacademy.ac.uk/resources/assessment/plagiarism/detectiontools_comparison_source_free_text_comparison.html
Sherlock	http://sydney.edu.au/engineering/it/~scilect/sherlock/
Turnitin	http://turnitin.com/static/index.php
MyDropBox	http://www.mydropbox.com/
Essay Verification Engine (EVE)	http://www.canexus.com/

Table 2.4. Source Code Plagiarism Detection Tools

SOURCE CODE PLAGIARISM TOOLS	URL
Moss (Measure Of Software Similarity)	http://theory.stanford.edu/~aiken/moss/
JPlag	https://www.ipd.uni-karlsruhe.de/jplag/
CodeMatch	http://www.safe-corp.biz/products_codematch.htm
YAP, YAP2, YAP3	http://pam1.bcs.uwa.edu.au/~michaelw/YAP.html
SIM	http://www.few.vu.nl/~dick/sim.html
SID	http://genome.math.uwaterloo.ca/SID/
CPD (Copy/Paste Detector), PMD	http://pmd.sourceforge.net/cpd.html
AC	http://tangow.ii.uam.es/ac/
Plague	http://www.freepatentsonline.com/y2010/0325614.html
Sherlock	http://sydney.edu.au/engineering/it/~scilect/sherlock/
Marble	http://www.cs.uu.nl/research/techreps/repo/CS-2010/2010-015.pdf
Plaggie	http://www.cs.uu.nl/research/techreps/repo/CS-2010/2010-015.pdf
Siff	http://glimpse.arizona.edu/javadup.html
Bandit	http://socrates.cs.man.ac.uk/~ajw/pd.html

The Plague algorithm ignores comments, variable names, function names, and other elements that can easily be globally or locally modified in an attempt to fool a plagiarism detection tool. Plague uses UNIX shell tools for processing. It runs slowly, because of using UNIX shell tools. It has three phases to its detection. These phases are Source Code Files, Structure Profiles and Similar Pairs. Each program is come down to sequences of tokens and structure metrics that represent control structures and data structures in the program in Source Code Files phase. The structure profiles are compared to find similar code structures in Structure Profiles phase. Pairs of files with similar code structures are compared using a variant of the Longest Common Subsequence Algorithm to find similarity in Similar Pairs phase. Plague is hard to adapt to new programming languages because the tokens depend on specific language statements and the structure metrics depend on specific programming language structures. Plague returns results in a list and then use an interpreter to process this list to show results in a way, so that a user can understand it easily.

YAP, which stands for Yet Another Plague, is a series of systems which follow a common pattern. The YAP Programs (YAP1, YAP2, and YAP3) were developed by Michael Wise at the University of Sydney, Australia (Zeidman, 2010). They are non-commercial tool and open source. YAP is an extension of Plague. YAP is designed to run locally. It supports only single file submissions. All three versions of YAP have two phases in their processes. The first phase is the generation phase, where a token file is created for each source code. In the first stage, which is common to all three systems, source texts are tokenized. Comments and string constants are removed. Upper-case letters are translated to lower-case. Synonymos are map to a common form. If possible, the functions are reordered into their calling order. All tokens that are not specific keywords for the language are removed. The second phase is comparison of every token file pairs. It is identical for YAP1 and YAP2. The result of each comparison is a percent match value. This value must be between 0 and 100. YAP1 relied on the sdiff function in UNIX to compare lists of tokens for the longest common sequence of tokens. It uses a mixture of UNIX utilities, tied together with a Bourne-shell script. It is a structure-metric system.

YAP2 improved performance in the second phase by benefiting from a more complicated algorithm known as Heckel's algorithm. It is implemented and written in Perl. For YAP3, the Running-Karp-Rabin Greedy-String-Tiling (RKR-GST) algorithm that is more immune to tokens being transposed is used in the second phase.

SID (Chen et al., 2004) stands for Shared Information Distance or Software Integrity Detection. It detects similarity between source codes of two programs by computing the shared information. It was originally an algorithm developed for comparing how similar or dissimilar genomes are. It is a modification of a genome comparison algorithm. It was then realized that this algorithm could be extended to many other applications including finding and detecting plagiarism in source code. It could find chain letter history. SID was built based on the hope that some crude approximation can still yield a useful system. In SID, a compression algorithm is used heuristically to approximate Kolmogorov complexity. It supports Java and C++ languages. It works in two phases. In the first phase, source codes are parsed to generate token sequences by a standard lexical analyzer. In the second phase, TokenCompress algorithm is used to compute the shared information metric between each program pair. Finally, all the source code pairs are ranked by their similarity distances. Similar parts are highlighted on SID graphical user interface. So instructors can compare the program source codes side by side using the SID graphical user interface easily. The SID system has been implemented in Java. Instructors can submit all the programs they wish to compare by creating a carefully formatted zip file that contains each program in its own directory, then submitting it to the SID server through a web browser. It is free but to use it, user first needs to create an account. It is not open source.

Moss stands for “Measure of Software Similarity”. Moss was developed at the University of California at Berkeley by Alex Aiken (Zeidman, 2010). It was developed in 1994. Since that date, it has been very effective in this role. Moss is an automatic system for determining the similarity of programs. It is used generally to detect plagiarism in programming classes. The algorithm behind Moss is a significant advancement over other plagiarism detection algorithms. It is being

provided as an Internet service. It is user friendly, because the service has been designed to be very easy to use. The current Moss submission script is for Linux. It does not accept jar files. Moss server produces HTML pages listing pairs of programs with similar code. Moss also highlights individual passages in source codes that appear the same. So it makes easy to quickly compare the files. Moss is fast and free. It is for non-commercial use. Moss supports C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2 languages. Anyone may obtain a Moss account. It uses k-grams for dividing non-whitespace characters of each file. In the last phase it compares source code fingerprints. Moss uses Winnowing algorithm to detect plagiarism.

CodeMatch (Zeidman, 2004) is a commercial plagiarism detectiontool. It takes an entirely different approach to plagiarism detection than the other programs. First, CodeMatch compares features of each pair of source-code files completely, rather than using a sampling method for comparing a small number of hashed samples of code. Therefore finding plagiarism among large sets of large files may be required to run for long hours. Second, CodeMatch makes use of knowledge of programming languages to improve the matching results. There is a small amount of information needed in the form of a list of common programming language statements that CodeMatch must recognize. In addition, it needs information on characters that are used to identify comments and characters. It currently supports ABAP, ASM-M68k, BASIC, C, C++, C#, Delphi, Flash ActionScript, Fortran, FoxPro, Java, JavaScript, LISP, MASM, MATLAB, Pascal, Perl, PHP, PowerBuilder, Python, RealBasic, Ruby, SQL, Verilog, VHDL, Visual Basic programming languages. CodeMatch compares thousands of source code files in multiple directories and subdirectories to determine which files are the most highly correlated. This can be used to significantly speed up the work of finding source code plagiarism, because it can direct the lecturer to look closely at a small amount of source code files rather than thousands of combinations. It is also useful for finding open source code within proprietary code, determining common authorship of two

different programs, and discovering common, standard algorithms within different programs. CodeMatch uses a combination of five algorithms to find plagiarism: Source Line Matching, Comment Line Matching, Word Matching, Partial Word Matching, and Semantic Sequence Matching. By using all five algorithms, chances of missing plagiarized code is significantly decreased.

JPlag (Prechelt, 2000) is a system that finds pairs of similar programs among a given set of programs. It has successfully been used in practice to detect plagiarisms among student programming assignment submissions. It supports C, C++ and Scheme languages. It is written in Java by Lutz Prechelt and Guido Malpohl of the University Karlsruhe and Michael Philippsen of the University of Erlangen-Nuremberg, to detect source code plagiarism (Zeidman, 2010). JPlag is publicly available as a web service. It takes as input a set of programs, compares these programs in groups of two and provides as output a set of HTML pages that allow for exploring and understanding the similarities found in detail. Similar segments of the code will be marked with different font colors. It works by parsing and converting each program into tokens. It has two interesting features which are its availability as a web service and its unique graphical user interface for understanding the results. Users who want to use JPlag can apply for a JPlag account by email. If they send their request, a username and password are sent to them with an e-mail and then they can log in and use JPlag at any time. For using the service, the user supplies a directory of files on user's local machine. The files are uploaded to the remote server, and the server applies JPlag to the files. It stores the resulting HTML pages to be viewed by a standard web browser. Only this particular JPlag user can view the web pages containing the results. The web pages remain on the server until they expire after several days or are overwritten by a new submission of the same user. JPlag uses Greedy String Tiling algorithm to detect plagiarism.

2.3.2.1. Comparison of Source Code Plagiarism Tools

The main difference of the three versions of YAP is the algorithm used during the comparison phase. YAP1 uses Longest Common Subsequence algorithm (Wise,

1996). YAP2 uses Heckel algorithm (Wise, 1996). YAP3 uses Running Karp-Rabin Greedy String Tiling algorithm (Wise, 1996). According to Heckel (1978) YAP2 is much faster than YAP1. YAP3 is an improvement over Plague in that it does not attempt a full parse of the programming language as Plague does. YAP3 is still open to changing the order of code lines in the source code less than Plague. YAP3 is able to deal effectively with transposed subsequences. YAP2 also has this ability, too.

Moss and JPlag plagiarism softwares are popular today (Zeidman, 2010). It is observed that Moss uses the Winnowing Algorithm, and JPlag uses the Greedy String Tiling Algorithm for similarity computations. We tested Moss and JPlag on some datasets and we observed that although Moss works faster than JPlag, JPlag produces more accurate results than Moss. JPlag uses roughly the same basic comparison algorithm as developed by Wise, but adds different optimizations for improving its run time efficiency. Moss uses a slightly different approach for much higher speed at the cost of some detection performance. The Moss and JPlag systems are stand-alone softwares. Both are running on their own remote server. Sending and loading the assignments to their server and comparison processes take much time because of running on their external server. Moreover, it is a huge disadvantage of Moss and JPlag that they do not produce results in the case of server crash and network connectivity issues and the Internet problems.

JPlag is available as a web service. The email service of Moss is roughly comparable, but nothing similar is available for YAP3. JPlag has a stronger user interface for interpreting the results. It has a similar interface for Moss, but no such interface is available for YAP3. JPlag is resource-efficient and scales to large submissions. Moss is even better in this respect, but YAP3 is inferior. JPlag's performance with repect to similarity score computation is as good as or slightly better than that of Moss and YAP3. One other difference between JPlag and YAP3 is that JPlag produces a very nice detailed HTML output. It also allows the user to click on a file combination to bring up windows showing both files with areas of similarity highlighted.

There is another source code plagiarism detection software namely CodeMatch. When CodeMatch is compared with JPlag, to use JPlag user must create

an account, while to use CodeMatch an account does not be created. The other difference of JPlag from CodeMatch is that JPlag is a free tool, but CodeMatch is a commercial tool.

SID is another source code plagiarism detection tool and it has a web interface. SID can use Approximate Matching to compute similarity between source codes, but Moss uses Winnowing Algorithm. SID maintains account information online, but Moss doesn't maintain account information online. SID is not a commercial tool like Moss. Moss is faster than SID. Moss is more successful software than SID. Moss gives more accurate resultThe features offered by various tools that aid with the detection of source-code plagiarism are compared in Table 2.5 (http://www.ics.heacademy.ac.uk/resources/assessment/plagiarism/detectiontools_comparison_source.html).

Table 2.5. Comparison of Source Code Plagiarism Tools

	JPlag	Moss	CodeMatch	SIM	SID	YAP1/YAP2/YAP3 (Yet Aanother Plague)	Sherlock	Copy/Paste Detector (CPD)
System Started from	1996	1994	2005	1989	2004	1992 - 1996	1994	2003
Language (s) supported	Java, C#, C, C++, Scheme and natural language text	C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, HCL2	BASIC, C, C++, C#, Delphi, Flash ActionScript, Java, JavaScript, MASM, Pascal, Perl, PHP, PowerBuilder, Ruby, SQL, Verilog, VHDL	C, Java, Pascal, Modula-2, Lisp, Miranda, and for natural language	C, C++, Java	Pascal, C, LISP	Java, C, C++, natural language text	Java, JSP, C, C++, Fortan and PHP code
Account	User must create an account	User must create an account	No	No	User must create an account	No	No	No
Service	Web service	Internet service	Standalone application	Standalone application	Web-based	Standalone application	Standalone application	Standalone application
Interface	GUI	Web interface	GUI	GUI	Web-based	GUI	GUI	GUI
Security	User id and e-mail needed	User id and e-mail needed	Runs locally	Runs locally	Runs locally	Runs locally	Runs locally	Runs locally

Table 2.5. Con't.

	JPlag	Moss	CodeMatch	SIM	SID	YAP1/YAP2/YAP3 (Yet Aanother Plague)	Sherlock	Copy/Paste Detector (CPD)
Cost	Free	Free	Commercial tool, 40 license tokens which is a $400 value	Do not advertise licenses, but make the source code and instructions for running the program on user's home machine available	Free and Not Open Sourced	The code is released only non-commercial use is allowed as per the README in the download and thus it is not Open Source either.	Free and open sourced	Free and open sourced
Requirements	Web browser, Java Runtime Environment (JRE), Java 1.5 or higher	A submission script for either UNIX or Windows		N/A	Web browser	UNIX OS	JDK 1.4 or later	JDK 1.4 or later
Speed	Fast	Fast	Large sets of files will require long amounts of time to analyze	N/A	Large sets of files will require long amounts of time to analyze	N/A	Large sets of files will require long amounts of time to analyze	Fast

Table 2.5. Con't.

	JPlag	Moss	CodeMatch	SIM	SID	YAP1/YAP2/YAP3 (Yet Aanother Plague)	Sherlock	Copy/Paste Detector (CPD)
Option for excluding files	Yes	Yes	Yes	N/A	N/A	N/A	Yes	No
Submission Methods	Standalone Java software application that can be deployed over the network	Command line	Standalone application	Command line	Web form	Command line	Standalone Java application	Standalone application
Source Data	Files, or a directory with or without subdirectories	Files, or a directory with or without subdirectories	Files, or a directory with or without subdirectories	Only single file submission	Zip file	Only single file, By specifying folder as parameter to Perl executable	A directory containing subdirectories, each containing one or more zip archives or submissions	Files, or a directory with or without subdirectories
Result Output	HTML output for exploring the similar code fragments found	Both text and HTML reports	HTML report, and spreadsheet showing statistical information	N/A	SID site	Written to file	Interactive dialogues for comparing detected similarities and printing reports	Text report, xml file, cvs file.
Result storage	Local	Remote	Local	N/A	SID Site	N/A	Local	Local
Overview results display method?	Histogram, statistics	Ordered list	Ordered list	N/A	Ordered list	N/A	Matching pair tree	Ordered list

Table 2.5. Con't.

	JPlag	Moss	CodeMatch	SIM	SID	YAP1/YAP2/YAP3 (Yet Aanother Plague)	Sherlock	Copy/Paste Detector (CPD)
Visual display of matched file pairs?	Yes	Yes	No	N/A	Yes	N/A	Yes	No
Miscellaneous	Several detection parameter settings, including sensitivity	Self adjusting	Some detection parameter settings	Some detection parameter	Some detection parameter	Some detection parameter	Several detection parameter settings, including sensitivity	Some detection parameter settings
Metrics produced	Percentage similarity, token matches	Percentage similarity, token matches, lines matched	Percentage similarity, lines matched	Percentage similarity	Percentage similarity	Percentage similarity	Percentage similarity, token matches	Token matches, lines matched
Algorithms	Greedy String Tiling	Winnowing algorithm	String matching	A dynamic programming string alignment technique	Kolmogorov's Complexity, TokenCompress algorithm, Lempel-Ziv type algorithms	Longest Common Subsequence (LCS) / Heckel's algorithm / Running Karp-Rabin Greedy String Tiling	Token based matching algorithm for source-code, string matching for natural language texts	Greedy String Tiling
Usability	easy to use Java Web Start client	after registration, you obtain a submission script	Easy to use	command line interface, fairly usable,	Easy to use	Easy to use	Easy to use	Easy to use

2.3.3. Plagiarism Detection Algorithms

Plagiarism detection in source code requires different algorithms than in textual documents. Paraphrasing, Sentence Matching, Keyword Matching, Semantic Similarity, Support Vector Machines (SVM), Character N-Gram- Based (CNG), Vector-Based (VEC), Syntax-Based (SYN), Semantic- Based (SEM), Fuzzy-Based (FUZZY), Structural-Based (STRUC), Stylometric-Based (STYLE), Cross-Lingual Techniques (CROSS), Fingerprinting, N-Gram, Semantic Networks and Inverted Index-Based Algorithm are kinds of textual document plagiarism algorithms (Alzahrani et al., 2011). The source code similarity detection algorithms are Running Karp-Rabin Greedy String Tiling (Wise, 1993), The Longest Common Subsequence (Zeidman, 2010), Heckel's Algorithm (Heckel, 1978), Winnowing Algorithm (Schleimer et al., 2003), Greedy String Tiling (Wise, 1993), Word Matching (Zeidman, 2004), Partial Word Matching (Zeidman, 2004), Source Line Matching (Zeidman, 2004), Comment Line Matching (Zeidman, 2004) and Semantic Sequence Matching Algorithms (Zeidman, 2004).

2.3.3.1. Running Karp-Rabin Greedy String Tiling Algorithm

Running Karp-Rabin Greedy String Tiling (RKR-GST) involves tiling one string with matching substrings of a second string. RKR-GST (Wise, 1993) is able to detect transposed substrings. Karp and Rabin's algorithm for fast substring matching is apparently the earliest version of fingerprinting based on k-grams. The RKR-GST algorithm employs a search for substrings of some fixed length. Instead of having a single hash-value for the entire pattern string, a hash value is created for each unmarked substring of the pattern string length for the substring. Each of the hash values for the pattern string is then compared with the hash-values for text string. After the each iteration the length of the matches is decreased until the minimum match length is reached. The worst-case complexity of Running Karp-Rabin Greedy String Tiling is $O(n^3)$. YAP2, which is one of the YAP Programs, uses Running Karp-Rabin Greedy String Tiling Algorithm to detect plagiarism.

2.3.3.2. Longest Common Subsequence Algorithm

The Longest Common Subsequence (LCS) Algorithm (Zeidman, 2010) is to find the longest subsequence common to all sequences in a set of just two sequences of items. Subsequence is different from a substring. A common subsequence of two strings is a subsequence that appears in both strings. It is based on the concept of creating fingerprints from hashes of k-grams guaranteed to detect copies. A k-gram is a continuous substring of characters (numbers or letters) of length k created by dividing the document into k-grams where k is a parameter chosen by the user. A longest common subsequence is a common subsequence of maximal length. It finds the longest common subsequence with gaps. It is a simple formal statement of the problem that yields good results. But LCS has two disadvantages. The first disadvantage is that it is not necessarily the correct formalization of the problem. The second disadvantage is that it concerns the computational problems. In the worst case, it can take O(*mn*) time. The only implementation of LCS for file comparison takes in worst-case O(*mnlogn*) time and O(*mn*) space where *n* is the sum of the lengths of the input strings, and *m* is the number of tokens. YAP2, which is one of the YAP programs, uses the Longest Common Subsequence Algorithm to detect plagiarism.

2.3.3.3. Heckel's Algorithm

Heckel's Algorithm (Heckel, 1978) is an algorithm to detect source code similarity. In order to compare two files, it is usually suitable to take a line as the basic unit, although word, sentence, paragraph, or character units are possible. Similarities are found until only differences remain. Two observations are made. The first observation is that a line that occurs only once in each file must be the same line. The second observation is that if in each file adjacent to a found line pair there are lines identical to each other, these lines must be the same line. YAP1, which is one of the YAP Programs, uses Heckel's Algorithm to detect plagiarism. It requires several passes, its overall complexity is linear (Wise, 1993).

2.3.3.4. Winnowing Algorithm

The Winnowing Algorithm (Schleimer et al., 2003) is used for detecting plagiarism. It is an efficient local fingerprinting algorithm. This algorithm uses hashing technique for document fingerprinting. Fingerprinting using the Winnowing method appears to be an effective way to detect potentially suspicious similarities between source code files. The set of fingerprints is a small subset of the set of all k-gram hashes. A fingerprint also contains positional information describing the document and the location within that document that the fingerprint came from. If the hash function is chosen in order that the probability of collisions is very small, then at any time when two documents share one or more fingerprints, it is extremely likely that they share a k-gram as well. The Winnowing algorithm is very simple. Firstly whitespaces and variable names are removed, and then "n-grams" are hashed. The smallest hash in each window is stored. If the first hash in each window were stored, adding an extra character at the start of a file would result in no matching fingerprints being found for otherwise identical documents. As well as comparing fingerprints filenames, size of the jar file, the number of entities in the jar file, and the name of the jar file itself are also compared. The minimum hash value should be selected in each window. If there is more than one hash with the minimum value, the rightmost occurrence must be selected. Then all selected hashes as the fingerprints of the source code are saved. After applying the winnowing algorithm, each document becomes a multi-set of hash values. The Winnowing algorithm selects the fingerprints from a sequence of hashes. The key property of winnowing is that the choice of hash depends only on the contents of the window. It does not depend on any external information about the position of the window in the file. It also does not depend on its relationship to other windows. One of the advantages of the winnowing algorithm for hashing selection is the trade-off between the fingerprint length and the shortest matching string to be detected. A disadvantage of this method is that it gives no guarantee that matches between source codes are detected. Moss uses the Winnowing Algorithm to detect plagiarism. Its time complexity is $O(n^3)$.

2.3.3.5. Greedy String Tiling Algorithm

Greedy String Tiling Algorithm is introduced by Michael Wise (Wise, 1993). It is used to compare two token strings. When comparing two token strings, the aim is to find a set of substrings that are the same. Any token of the first source code file may only be matched with exactly one token from the second source code file. Any token of the first source code lines may only be matched with exactly one token from the second source code lines. Substrings are to be found independent of their position in the string, but the relative positions do not change. Long substring matches are preferred over short ones, because they are more reliable. Short matches are more likely to be inaccurate.

Greedy String Tiling Algorithm has two phases. Before the phases, in the preprocessing step, whitespace, comments and identifier names are removed. Remaining language statements are replaced by tokens. In the first phase, the two source code lines are searched for the longest sequential matches. In order to achieve this, all the tokens in the first source code file are compared with every token in the second source code file. If they are identical, the match is extended as far as possible, and the set of all longest common substrings are collected. In the second phase, all matches of maximal length found in the first phase are marked. So every token is only used in one match. If some of the matches overlap, the first found match is chosen and the others will be ignored. These processes are repeated until no further matches are found. The length of the maximal matches decreases by one in each step, so the algorithm is terminated. The Minimum Match Length is used for avoiding inaccurate matches. JPlag uses Greedy String Tiling Algorithm to detect plagiarism. Its worst case complexity is $O(n^3)$.

2.3.3.6. Word Matching Algorithm

For each file pair, the Word Matching Algorithm (Zeidman, 2004) is used to count the number of matching identifiers. Identifiers being words are not programming language keywords. In order to determine whether a word is a

programming language keyword, comparison is done with a list of known programming language keywords. In some programming languages, keywords are case sensitive while in other programming languages keywords are not case sensitive. So case sensitivity is depend on the programming language is examined. In either case, when comparing non-keyword words in the file pairs, case is ignored. This case-insensitive comparison is done to prevent being fooled by simple case changes in plagiarized code in an attempt to avoid detection. CodeMatch uses Word Matching Algorithm to detect plagiarism.

2.3.3.7. Source Line Matching Algorithm

The Source Line Matching Algorithm (Zeidman, 2004) compares each line of source code from both files, by ignoring case. Comment lines are excluded. Sequences of whitespace are converted to single spaces so that the syntax structure of the line is preserved. If a line of source code has a comment at the end, the comment is removed for the comparison. Source lines that contain only programming language keywords are not examined. CodeMatch uses Source Line Matching Algorithm to detect plagiarism.

2.3.3.8. Comment Line Matching Algorithm

The Comment Line Matching Algorithm (Zeidman, 2004) compares each line of comments from both files, again by ignoring case. If a line of source code has a comment at the end, the source code will removed for the comparison. Only the comment lines are processed. The entire comments are compared, regardless of whether there are keywords in the comment or not. CodeMatch uses Comment Line Matching Algorithm to detect plagiarism.

2.3.3.9. Semantic Sequence Algorithm

The Semantic Sequence Algorithm (Zeidman, 2004) compares the first word of every source line in the pair of files. It ignores blank lines and comment lines. Whitespace and punctuation are removed. This algorithm finds sequences of code that appear to perform the same functions despite changed comments and identifier names. The algorithm finds the longest common semantic sequence within both files. The sequence of source lines in the two files is considered as matching because the first word is identical. If a longer sequence of source lines is found in the file, this algorithm returns the number of source lines in the longest sequence. CodeMatch uses Semantic Line Matching Algorithm to detect plagiarism.

3. MATERIAL AND METHOD

3. 1. Material

Moodle, GCC, Moss, JPlag and CodeMatch systems are used in this thesis, because Moodle is a popular free and open-source distance education system, GCC is well known and free compiler, and Moss, JPlag and CodeMatch are the most widely used plagiarism detection tools. Moodle, which is open-source software, is used as a distance education tool; Moss, JPlag and CodeMatch are used during the design and analysis of our plagiarism detection algorithm. In this section, detailed information about these tools are given.

3.1.1. GCC

GCC (http://gcc.gnu.org/) is originally named the GNU C Compiler. However, today GCC is used as the compiler system of the GNU environment. It distributed under the GNU Public License (GPL). The GCC was first designed and developed as compiler for C language. Today GCC supports various programming languages. It is a very high quality, very portable compiler for C, C++, and Objective C, Fortran, Java, Ada, Modula-3, Pascal, and Go languages.

GCC has been adopted as the standard compiler by most Unix-like operating systems, including Linux, the BSD family, and Mac OS X.

The GNU project has been headed by Richard Stallman from the MIT and started in late 1983 with the basic goal of developing free software to work in collaboration with other software or platforms that are not free (http://www.qwhatis.com/what-is-gcc/). The GNU system was developed to be 100% free software. This particular project started out with their own operating system called GNU in 1992. GCC has played a significant role in promoting the development of free software. GCC is also available for most embedded platforms, for example Symbian, AMCC, and Freescale Power Architecture-based chips. Because of being freely available for wide range of platforms, in this study GCC is used to compile C programming assignments.

3.1.2. Moodle

This study is targeted on making a plugin to Moodle software that provides compiling and checking plagiarism of programming assignments written in C language. As programming courses are important especially for Computer Engineering Education, in this study, our aim is to increase efficiency of grading process of C language assignments. Moodle is selected as a distance education tool for this study because of being one of the most widely used open-source Web Based Distance Education Model. The other reason is that; no compilation and plagiarism detection features of programming assignments for distance education tools are available. In this study Moodle is chosen among related softwares because, it is user friendly, it is free, it is open source, it includes all features of an open source learning management system that should have in a Learning Management System and it has features to improve educational quality. The detailed information about Moodle was given in Section 2.2.

.

3.1.3. Moss

Moss (Zeidman, 2010) is one of the widely used and successful plagiarism detection software. It is analyzed and compared with the other source code plagiarism detection tools such as JPlag and it is tested with a small group of datasets which are the assignments of Computer Engineering students in Çukurova University, the results obtained by Moss are not as accurate as the results of JPlag. For this reason the similarity computation algorithm of Moss, which is Winnowing Algorithm, is not selected for this study. However, we used Moss to determine the efficiency of our system. The detailed information about Moss was given in section 2.3.2. Moss should run on UNIX systems which have perl, uuencode, mail and either zip or tar. Moss allows for the programs that are compared to be composed of sets of files in different directories. The installed script will be referred to as the command moss. The moss command results in the programs being sent to the Moss server at Berkeley. When the results are ready, an email is sent back to the login name that invoked the moss command. The

email gives a web page address for the results. The return email from the similarity checking currently states that the results will be kept available for 14 days on the Moss server.

Moss Results

Wed Jul 27 12:59:56 PDT 2011

Options -l c -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
c:\xampp\scenario3\odev4_2008638001.c (84%)	c:\xampp\scenario3\odev4_2009638019.c (83%)	156
c:\xampp\scenario3\odev4_2008638012.c (3%)	c:\xampp\scenario3\odev4_2009638005.cpp (8%)	28
c:\xampp\scenario3\odev4_2008638012.c (2%)	c:\xampp\scenario3\odev4_2009638019.c (4%)	6
c:\xampp\scenario3\odev4_2008638001.c (4%)	c:\xampp\scenario3\odev4_2008638012.c (2%)	6

Any errors encountered during this query are listed below.

Figure 3.1. Moss Screenshot

3.1.4. JPlag

JPlag (Prechelt, 2000) is another successful plagiarism detection software which is also free. It is also tested with the same dataset and it gives more accurate result with respect to Moss. According to the similarity algorithm used by JPlag is more successful than a great deal of plagiarism algorithms (Prechelt, 2000). That's why Greedy String Tiling is chosen as the basis algorithm for detecting the similarities of source codes submitted by the students.

Users can use JPlag without installing it on a local machine. The login dialog of JPlag Web Start Client is on Figure 3.2. For obtaining a JPlag account lecturers must fill out the registration form on Jplag web site (https://www.ipd.uni-karlsruhe.de/jplag/). The admins of that web site do not give accounts to users of anonymous email addresses like Hotmail, Yahoo, Gmail, etc. With this account lecturers can then access the Java Web Start client to use JPlag.

Figure 3.2. JPlag Web Start Client - Login Dialog

When a registered user logins the program, menu dialog is opened as in shown Figure 3.3.

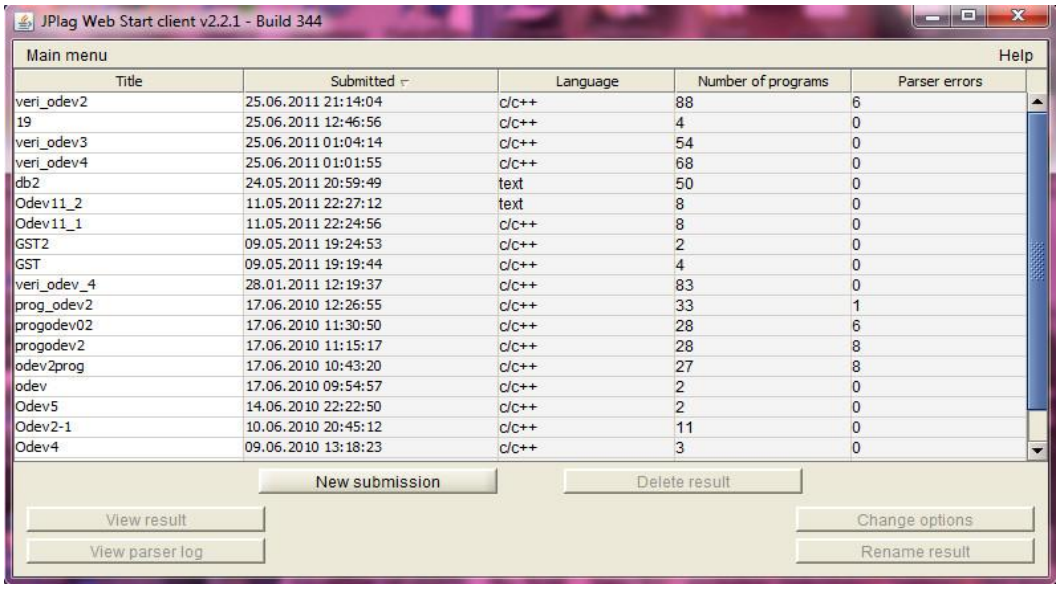


Figure 3.3. JPlag Web Start Client - Menu Dialog

When users want to load a new submission on a system, they must use “New Submission” button. If they click this button, a form as shown in Figure 3.4 is opened. Users can change some options of assignments such as suffixes, minimum match length and cluster type with Advanced Options button as shown in Figure 3.5.

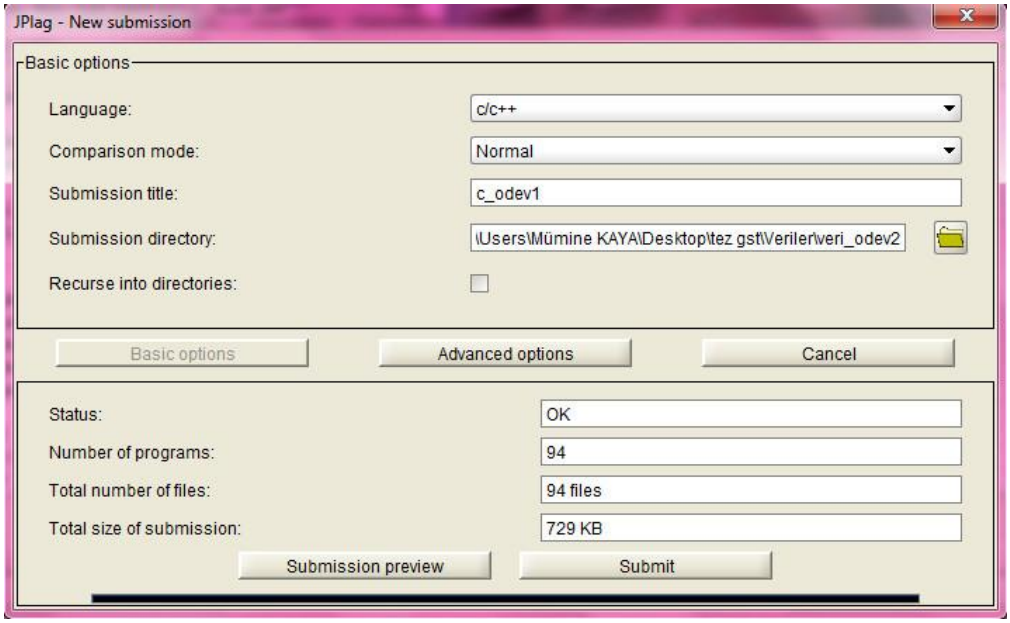


Figure 3.4. JPlag Web Start Client - New Submission

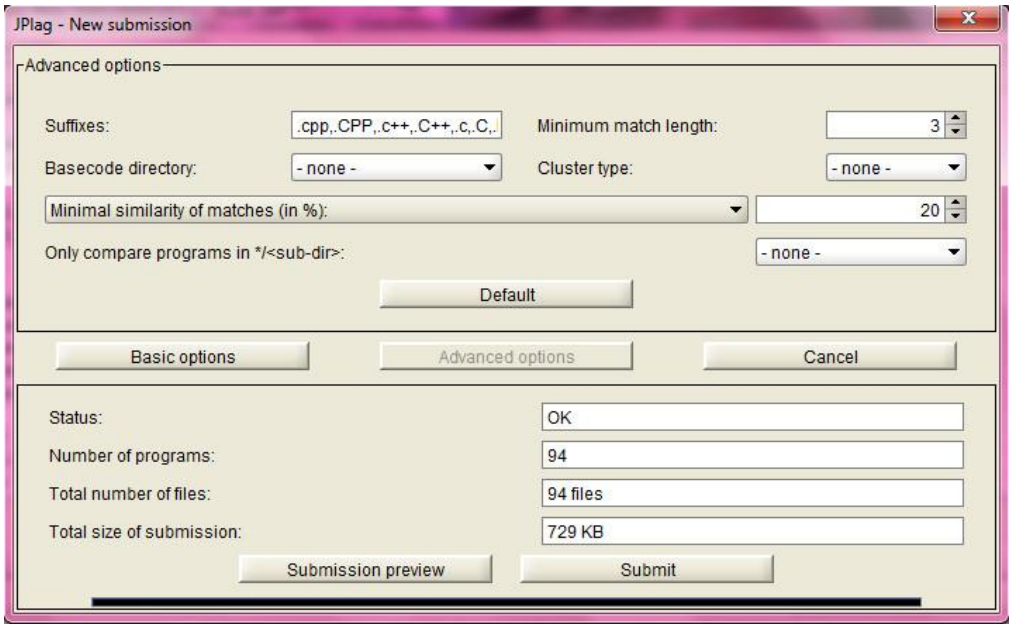


Figure 3.5. JPlag Web Start Client - Advanced Options of New Submission

Users can view the assignments before they submit them with “Submission Preview” button. As shown in Figure 3.6 all accepted assignments and invalid assignments are viewed.

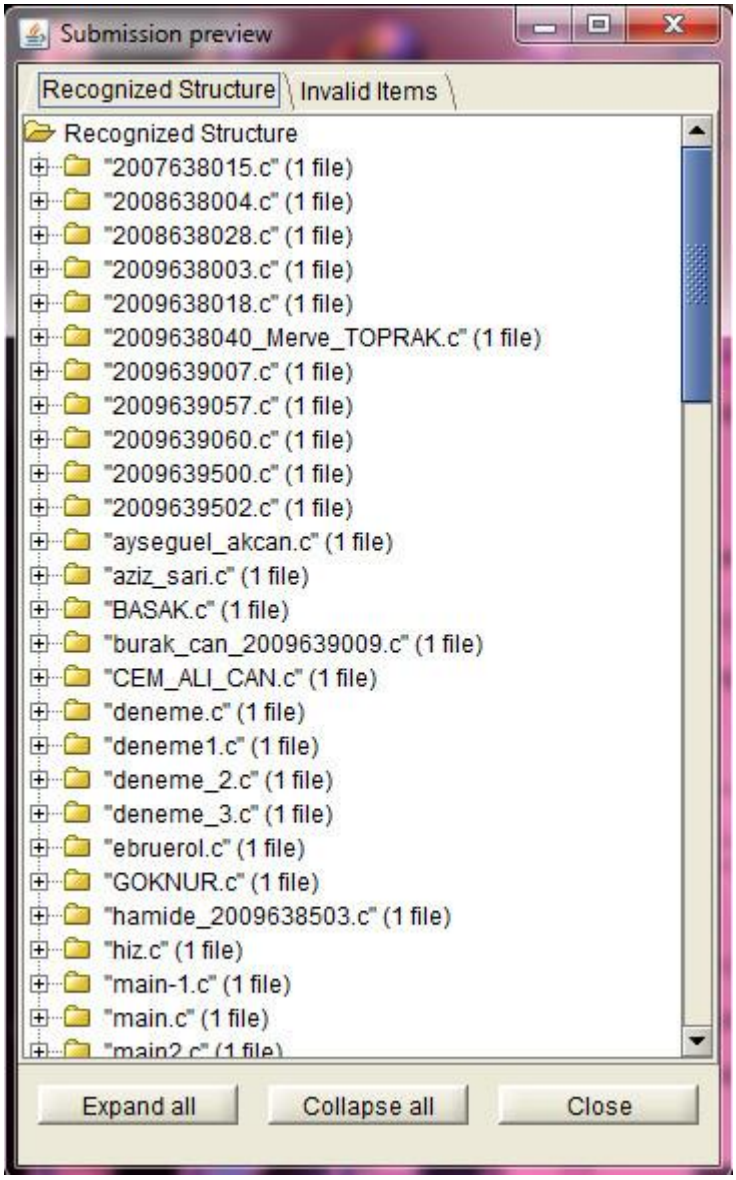


Figure 3.6. JPlag Web Start Client - Submission Preview

To see parser log, users can click “View Parser Log” button as shown in Figure 3.3. If they use this button, a form as in Figure 3.7 is shown.

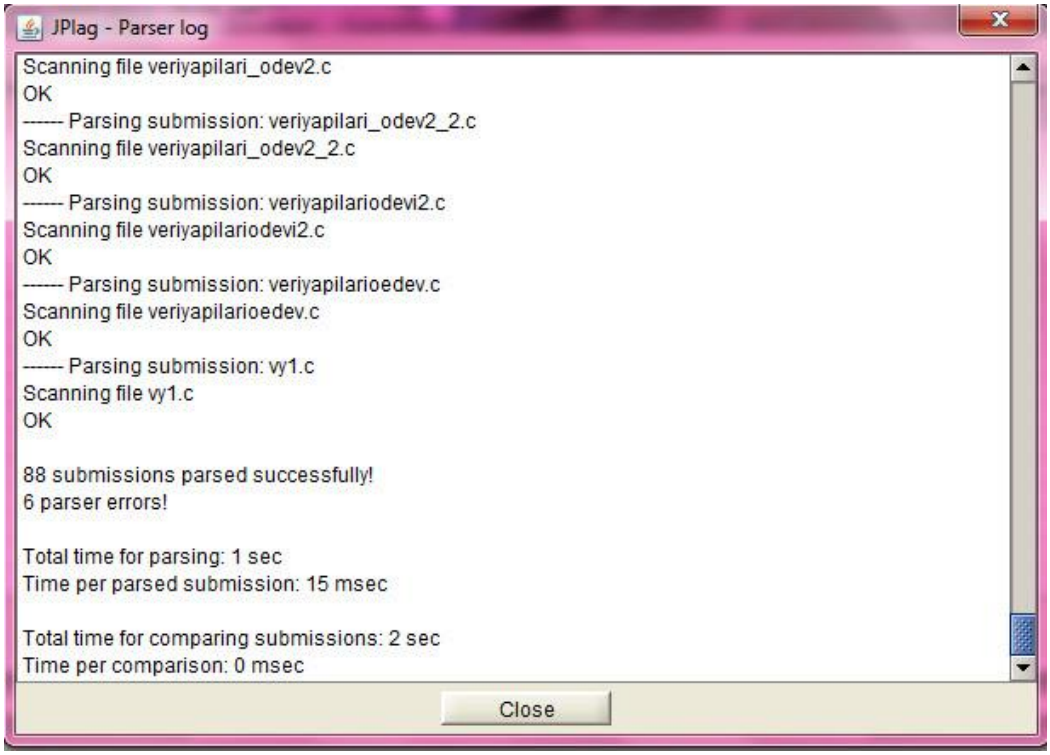


Figure 3.7. JPlag Web Start Client - Parser Log

If users click “Submit” button, then all assignments are sent to server and the progress is started. The progress consists of packing files, sending files, waiting in queue, parsing files, comparing files (Figure 3.8) and downloading the results from the server (Figure 3.9).

Figure 3.8. JPlag Web Start Client - JPlag Comparing Files Process

Figure 3.9. JPlag Web Start Client - JPlag Downloading Results Process

When all processes are finished, a link is created. The result locates in this link is shown in Figure 3.10.

Figure 3.10. JPlag Web Start Client - JPlag Progress

To view the result page, users must click the result link. For a set of submitted assignments, JPlag generates a set of HTML pages that presents the results. At the top of the web page as shown in Figure 3.11, there are a title, which is the name of the assignment, the names of the submitted programs, the language, the number of submissions, the matches, the date, the minimum match length, and suffixes of the submitted files.

Figure 3.11. JPlag Web Start Client - JPlag Search Results

The distribution of the assignments is on the middle of the web page as shown in Figure 3.12.

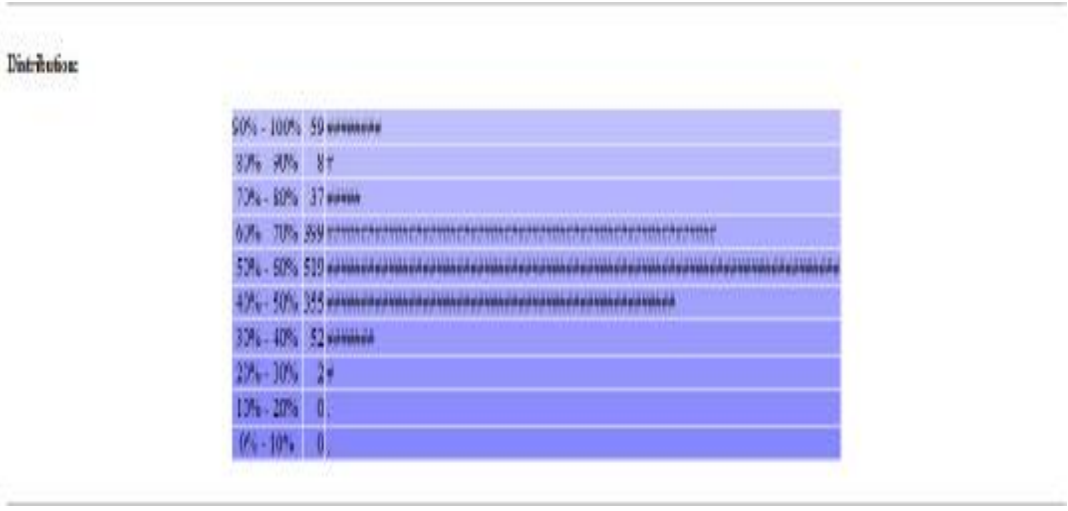


Figure 3.12. JPlag Web Start Client - Distribution

At the bottom of the web page there is a histogram of the similarity values found for all program pairs. A list of the highest-similarity pairs shown on the web page below the histogram allows for selecting these pairs. These matches sorted by two features. These features are average similarity, which is shown in Figure 3.13, and maximum similarity, which is shown in Figure 3.14.

Matches sorted by average similarity (What is this?):

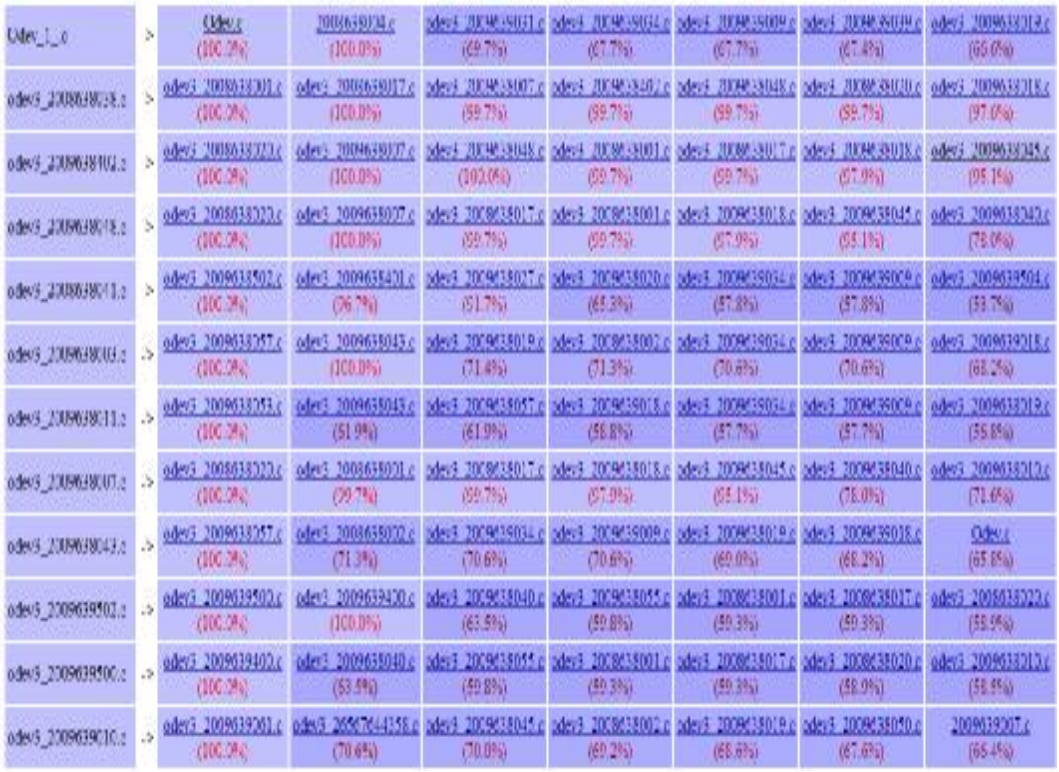


Figure 3.13. Matches Sorted By Average Similarity

Matches sorted by maximum similarity (What is this?):

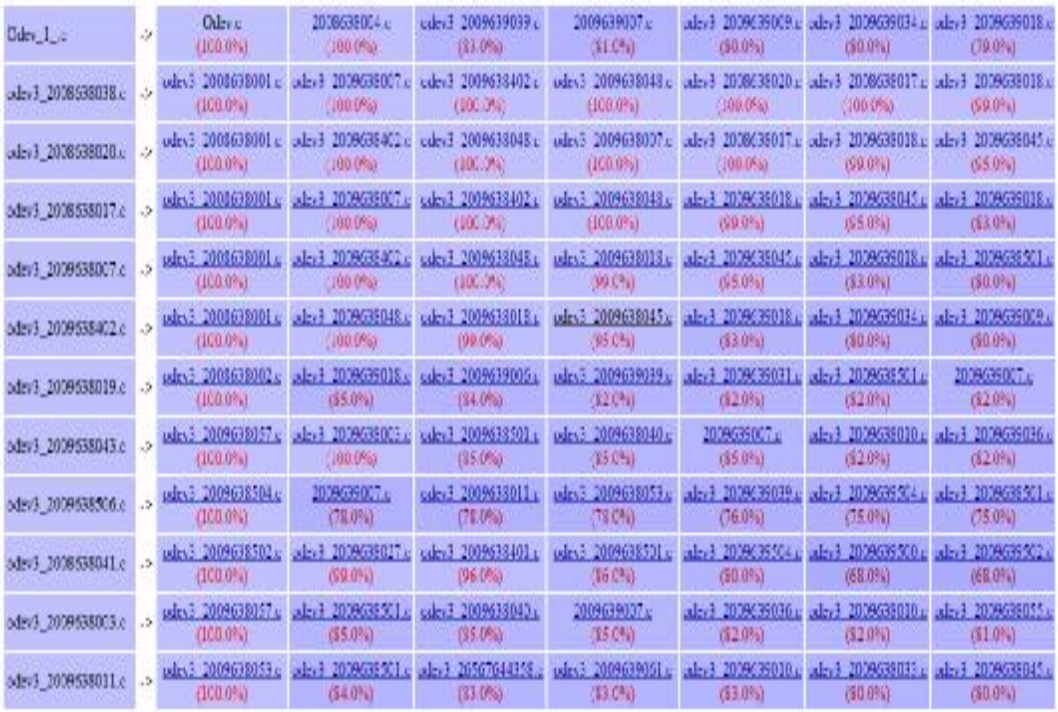


Figure 3.14. Matches Sorted By Maximum Similarity

For each selected pair clicked by the user, a side-by-side comparison of the programs is shown in Figure 3.15. Ranges of lines that have been found to be similar in both files are marked with the same color. A hyperlink arrow at each region, when clicked, aligns the opposite half of the display such that the similar region is shown. Likewise, one can jump to each pair of same regions from a correspondence table shown at the top which lists color code, position in each of the files, and length in tokens for each region. This unique presentation makes it very easy to judge whether or not the pair of source codes should in fact be considered as plagiarized.

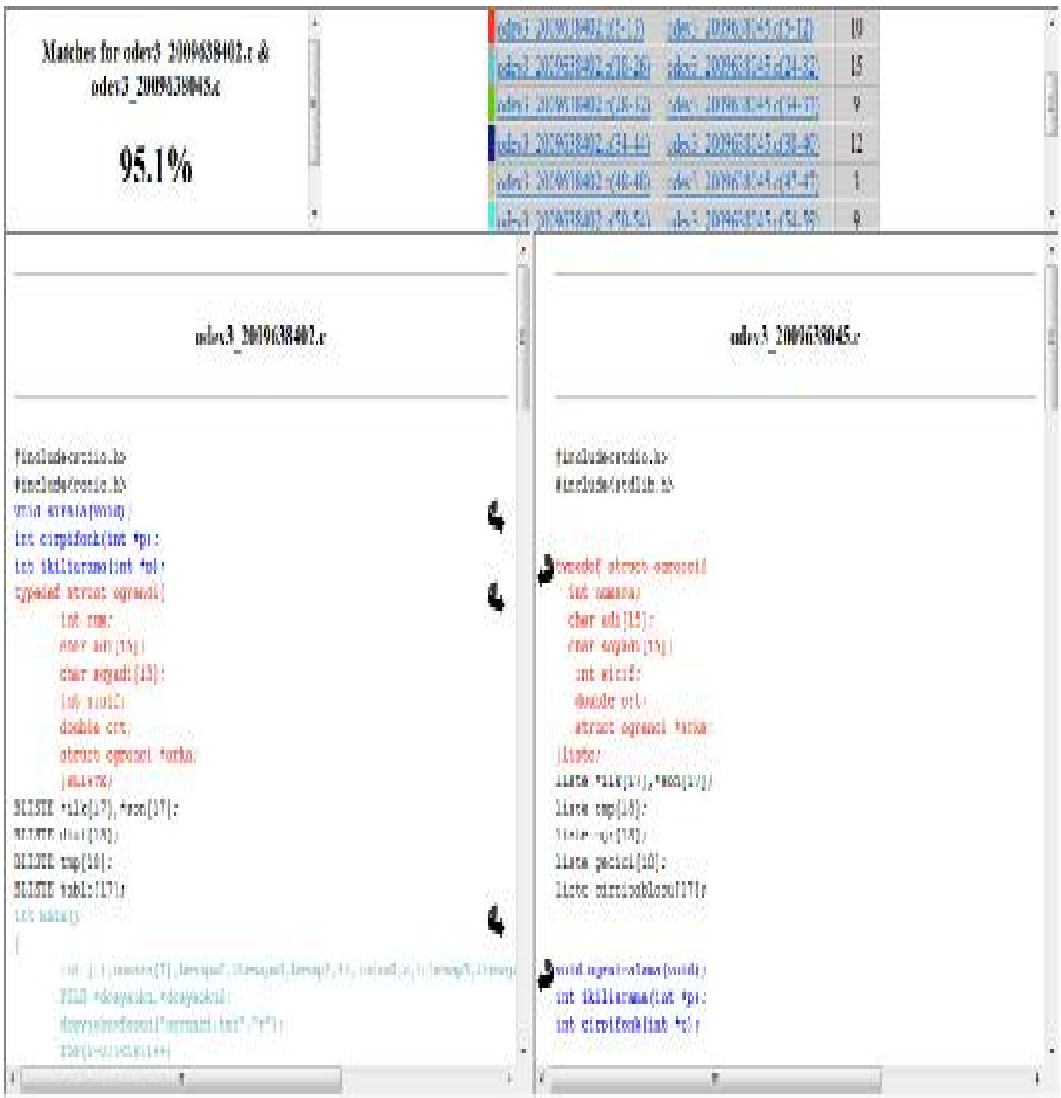


Figure 3.15. JPlag Results Display Page

3.1.5. Greedy String Tiling Algorithm

Greedy String Tiling Algorithm (Prechelt, 2000) is a successful string matching algorithm. The algorithm is used in JPlag Software. It gives more accurate results and it detects more successful similarities when it is compared with the other algorithms (Wise, 1993). The algorithm works according to the following rules (Prechelt, 2000): The first rule of this algorithm is that it is impossible to completely match source code parts that have been copied in a program which is plagiarized. The second rule of this algorithm is that long matches are chosen instead of short matches. The third rule is that reordering source code parts is no effective attack.

Tokenization is an important phase of Greedy String Tiling Algorithm. An example of Java Source Text and its tokens are as shown in Table 3.1.

Table 3.1. An Example of Java Source Text and Its Tokens (Prechelt, 2000)

Java Source Code	**Generated Tokens**
public class Count {	BEGINCLASS
public static void main(String[] args)	VARDEF, BEGINMETHOD
throws java.io.IOException {	
int count = 0;	VARDEF, ASSIGN
while (System.in.read() !=	APPLY, BEGINWHILE
count++;	ASSIGN, ENDWHILE
System.out.println(count+" chars.");	APPLY
}	ENDMETHOD
}	ENDCLASS

To search for the biggest contiguous matches of two source code files, three nested loops are used. The first loop iterates over all the tokens in the first source code file, the second loop compares the token with every token in the second source code file. If they are the same tokens, the second loop tries to extend the match as far as possible. These two loops collect the set of all longest common substrings. Then all founded matches of maximal length are marked, so

they cannot be used for further matches again and every token is going to be used only in one match. In the terminology of Wise (1993), by marking all the tokens a match becomes a tile and a lower bound for the length of a match is called the "Minimum Match Length".

The pseudocode of Greedy String Tiling Algorithm is as shown as in Figure 3.16:

```
Greedy-String-Tiling(String A, String B) {
        tiles = {};

        do {
                maxmatch = MinimumMatchLength;
                matches = {};

                Forall unmarked tokens A_a in A {
                        Forall unmarked tokens B_b in B {
                                j=0;
                                while ( A_{a+j} == B_{b+j} && unmarked(A_{a+j}) &&
                                        unmarked(B_{b+j}))
                                    j++;

                                if (j == maxmatch)
                                        matches = matches ⊕ match(a, b, j);

                                else if (j >maxmatch) {
                                        matches = {match(a, b, j)};
                                        maxmatch = j;
                                }
                        }
                }

                Forall match(a, b, maxmatch) ∈ matches {
                        For j = 0. . .(maxmatch- 1) {
                                mark(A_{a+j});
                                mark(B_{b+j} );
                        }
                        tiles = tiles ∪ match(a, b, maxmatch);
                }
        } while (maxmatch > MinumumMatchLength);
        return tiles;
}
```

Figure 3.16. The Pseudocode of Greedy String Tiling Algorithm (Prechelt, 2000)

3.1.6. CodeMatch

CodeMatch, developed Bob Zeidman (2004), is analyzed and used for the algorithm, which is going to be generated; because CodeMatch is one of the successful plagiarism detection software. It cannot be tested because it is a commercial program. For a typical evaluation CodeMatch provides 40 license tokens which is a $400 value and will enable the users to analyze up to 1MByte of code using CodeMatch. That's why CodeMatch cannot be examined in this study.

However in the articles about CodeMatch it is tested with some databases and it is compared with other plagiarism tools. The results are more accurate than the results of the other plagiarism tools. But JPlag gives as much good results as CodeMatch. CodeMatch uses unique algorithms to find various different ways that source code files are correlated. That's why the algorithms of CodeMatch which are Source Line Matching, Comment Line Matching, Word Matching, Partial Word Matching and Semantic Sequence Matching, is used together in this study.

By using these five algorithms, users can find more plagiarized code. It also can produce detailed reports and statistics spreadsheets about the comparison results. There is also a feature to allow filtering of the results to eliminate files, folders, and code elements that are not interesting to the analysis. CodeMatch produces a total correlation score based on the combination of five algorithms that the user chooses when running CodeMatch. Each match is given a weight and all weights are combined into a single matching score called the CodeMatch Score. The minimum score is 0 while the maximum score is 100.

The screenshot of CodeMatch program is shown in Figure 3.17 (http://www.safe-corp.biz/products_codematch.htm).

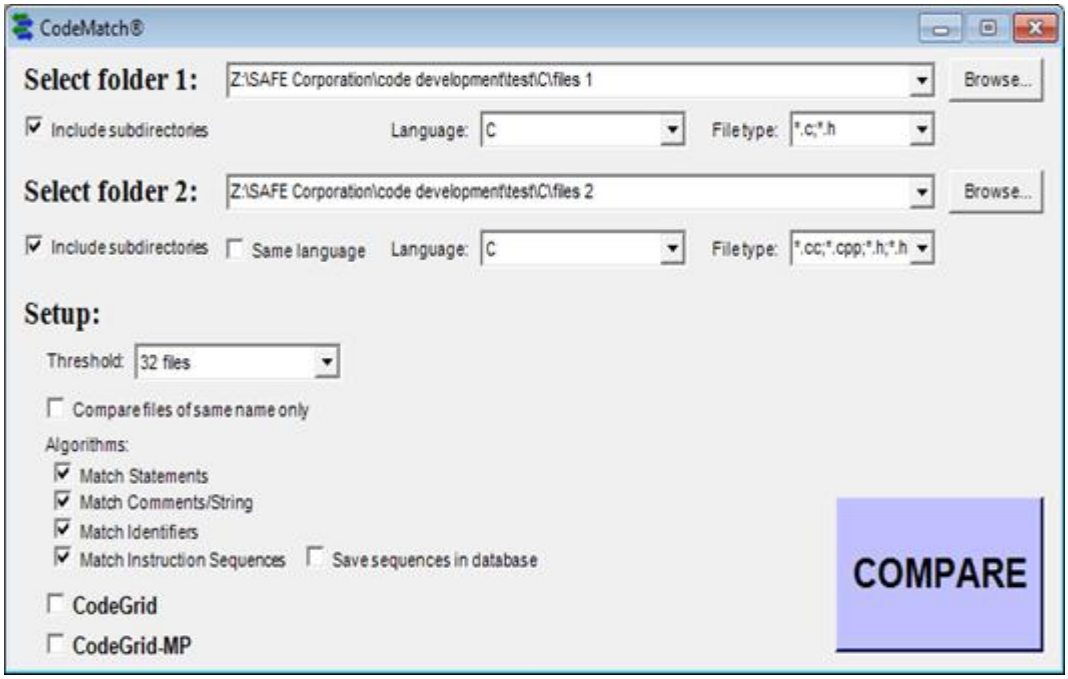


Figure 3.17. Screenshot of CodeMatch (http://www.safe-corp.biz/products_codematch.htm)

When the user clicks the Compare buton, the comparison between the submitted files begins. Results are then saved as a CodeMatch database file, and to view the results the user must convert the database into a html file. CodeMatch returns the top most similar files for each of the files submitted based on the threshold selected as shown as in Figure 3.18.

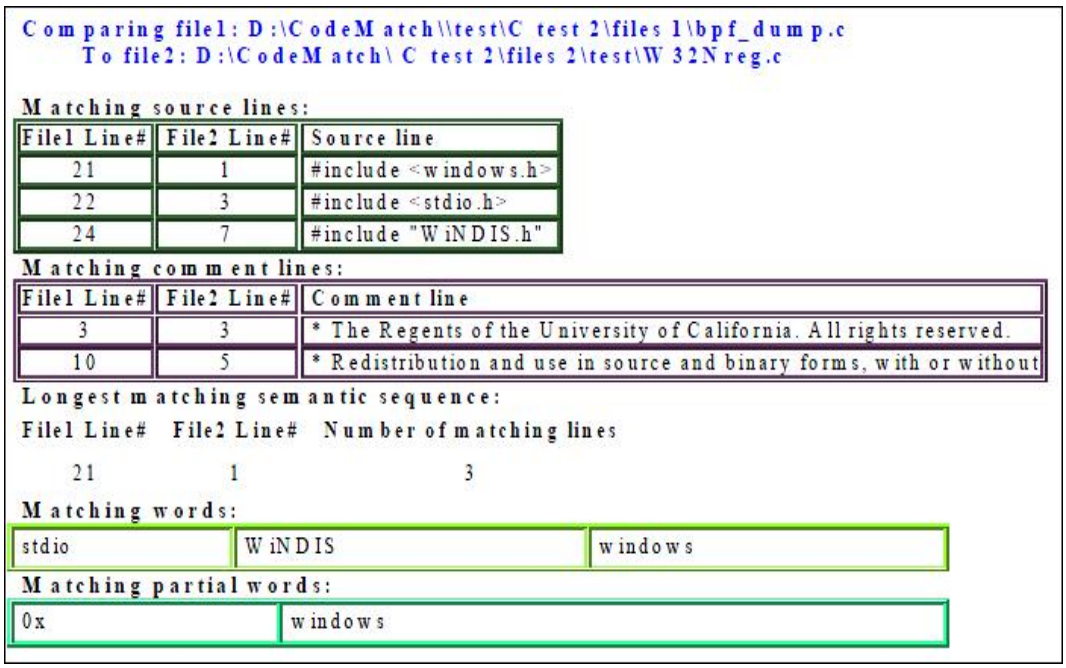
Comparing files in folder D:\CodeMatch\Code Development\test\C test 2\files 1
To files in folder D:\CodeMatch\Code Development\test\C test 2\files 2

D:\CodeMatch\Code Development\test\C test 2\files 1\bpf_dump.c
Compared To File
D:\CodeMatch\Code Development\test\C test 2\files 2\bpf_dump.c
D:\CodeMatch\Code Development\test\C test 2\files 2\W32NReg.c
D:\CodeMatch\Code Development\test\C test 2\files 2\test\W32NReg (variable names changed).c
D:\CodeMatch\Code Development\test\C test 2\files 2\test\W32NReg (no comments).c

D:\CodeMatch\Code Development\test\C test 2\files 1\bpf_filter.c
Compared To File
D:\CodeMatch\Code Development\test\C test 2\files 2\W32NReg.c
D:\CodeMatch\Code Development\test\C test 2\files 2\test\W32NReg (no comments).c
D:\CodeMatch\Code Development\test\C test 2\files 2\test\W32NReg (variable names changed).c
D:\CodeMatch\Code Development\test\C test 2\files 2\bpf_dump.c

Figure 3.18. CodeMatch Basic Result Report (Zeidman, 2004)

If user clicks the Match Score link, then a detailed result report will be opened as shown in Figure 3.19. In a detailed report, there are matching source lines, matching comment lines, longest matching semantic sequence, matching words and matching partial words sections.



Comparing file1: D:\CodeMatch\test\C test 2\files 1\bpf_dump.c
To file2: D:\CodeMatch\C test 2\files 2\test\W32Nreg.c

Matching source lines:

File1 Line#	File2 Line#	Source line
21	1	#include <windows.h>
22	3	#include <stdio.h>
24	7	#include "WiNDIS.h"

Matching comment lines:

File1 Line#	File2 Line#	Comment line
3	3	* The Regents of the University of California. All rights reserved.
10	5	* Redistribution and use in source and binary forms, with or without

Longest matching semantic sequence:

File1 Line#	File2 Line#	Number of matching lines
21	1	3

Matching words:

stdio	WiNDIS	windows

Matching partial words:

0x	windows

Figure 3.19. CodeMatch Detailed Result Report (Zeidman, 2004)

The algorithms used in CodeMatch are summarized briefly in the following subsections.

3.1.6.1. Source Line (Statement) Matching Algorithm

The Source Line Matching Algorithm (Zeidman, 2004) compares each line of source code from both files by ignoring case. The comment lines of source codes are removed for the comparison. It is valid not only the comments at the indent of source code line but also the comments at the end of source code line. For source lines to be considered as matches, they must contain at least one nonkeyword. This algorithm produces a similarity score and then it is weighted.

3.1.6.2. Comment Line Matching Algorithm

The Comment Line Matching Algorithm (Zeidman, 2004) compares each line of comments from both files, again ignoring case. The source code lines are removed for the comparison. The entire comment at the indent of source code line or at the end of source code line, is compared, regardless of whether there are keywords in the comment or not. CodeMatch looks for identical comments and strings, ignoring whitespace characters. Comment lines and strings that contain only programming language keywords are still considered as matches. This algorithm produces a similarity score and then it is weighted.

3.1.6.3. Word (Identifier) Matching Algorithm

CodeMatch finds every instance in each file where identifiers match exactly. It eliminates programming language keywords and only reports matches for non-keyword identifiers such as variable names and function names. So, for each file pair, the Word Matching Algorithm (Zeidman, 2004) counts the number of matching words that are not programming-language keywords. This algorithm is case sensitive. That's why for the case sensitive programming languages the words would not be considered as a language keyword would not be ignored. This algorithm produces a similarity score and then it is weighted.

3.1.6.4. Partial Word (Identifier) Matching Algorithm

The Partial Word Matching Algorithm (Zeidman, 2004) examines each nonkeyword word in the source code of one file and finds all words that match a sequence within one or more nonkeyword words in the other file. Like the word-matching algorithm, this one is also case insensitive. This algorithm also produces a similarity score which is weighted.

3.1.6.5. Semantic (Instruction) Sequence Matching Algorithm

CodeMatch looks for sequences of instructions that match. CodeMatch notes the longest such sequence in each pair of files. A sequence matches if the initial programming language statement on each line is identical, regardless of what follows it. Even if variable names are altered in one file, CodeMatch will report similarities in the files. This algorithm also produces a similarity score which is weighted.

3.2. Method

The purpose of this study is to design and develop a system that educators can load lecture notes and homework as online over the Internet, they can do online exams, and they can also compile and control plagiarism of the submitted C programming language homeworks. In addition to these, students can follow the course content, load their homeworks and discuss any questions regarding the course with the instructor. This new algorithm is included into the Moodle Distance Education Application as a plug-in. Therefore, the plugin which is prepared in this study is targeted to run at our local server. Unlike other plagiarism softwares, this new algorithm is going to be used as a module on the Distance Education Application with Moodle. In this study, that's why it is studied a new hybrid algorithm based on Greedy String Tiling Algorithm which is also used in JPlag and Source Line Matching, Comment Line Matching, Word Matching and Semantic Sequence Matching Algorithms of CodeMatch.

3.2.1. GCC Based Online Compiler

In this study, a module for compiling C programming assignments is created for programming based courses which are the core subjects in the computer engineering education. The compiler module is aimed to operate with the aid of the GCC which is a package that is designed as a part of the GNU Project and includes C and other compilers.

It was built in Moodle. It compiles only C and C++ source code assignments. As GCC supports other programming languages, in the future, online compiler for other languages can be included into Moodle. When students submit their assignments to Moodle as shown in Figure 3.20, teachers can read and compile them one by one or in the lump. In order to control all assignments in the lump, first "… ödev gönderisine bak" link should be clicked in Figure 3.20, and then "Karşılaştır & Derle" button in Figure 3.21 should be used. If it is clicked, then a web page will be opened. In that page, a table is shown like Figure 3.22.

Figure 3.20. Moodle Web Page of Submitted Assignment

Figure 3.21. Moodle Detailed Web Page of Submitted Assignment

Figure 3.22. Moodle Web Page of Viewing Assignment Content

When teachers click the “Ödev İçerik Görüntüle” button, then Moodle leads them to a new web page as shown as Figure 3.23 and Figure 3.24. On this page, the content of the selected assignment is showed. After the “Derle” button is clicked, the submitted assignment is sent to GCC program to compile.

Figure 3.23. The Content of the Selected Assignment

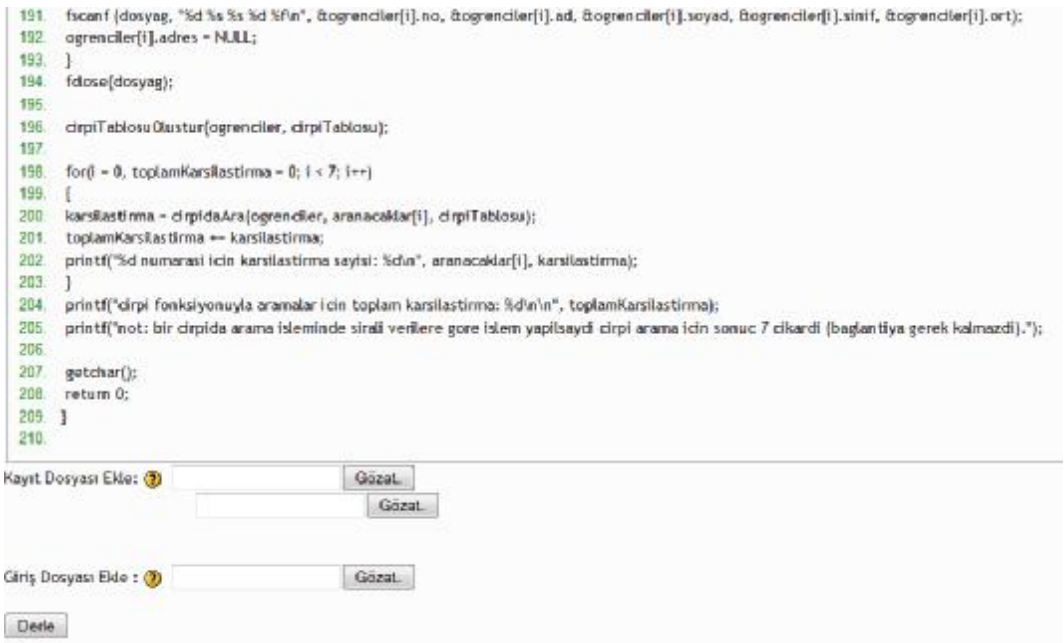


Figure 3.24. The Content of the Selected Assignment - 2

The result of the compilation process of the selected assignment is shown as Figure 3.25. If an error occurred during the compilation, than an error message is written to the bottom of the form. If this compilation is successful, a message is written to the form that compilation process is successful.

Figure 3.25. The Compilation of the Selected Assignment

In Figure 3.25, the "Programı Çalıştır" button is used to run the source code which is compiled successfully. Teachers can see the results (Figure 3.26 and Figure 3.27) and compare the results produced by the assignment. They can grade the assignments easily.

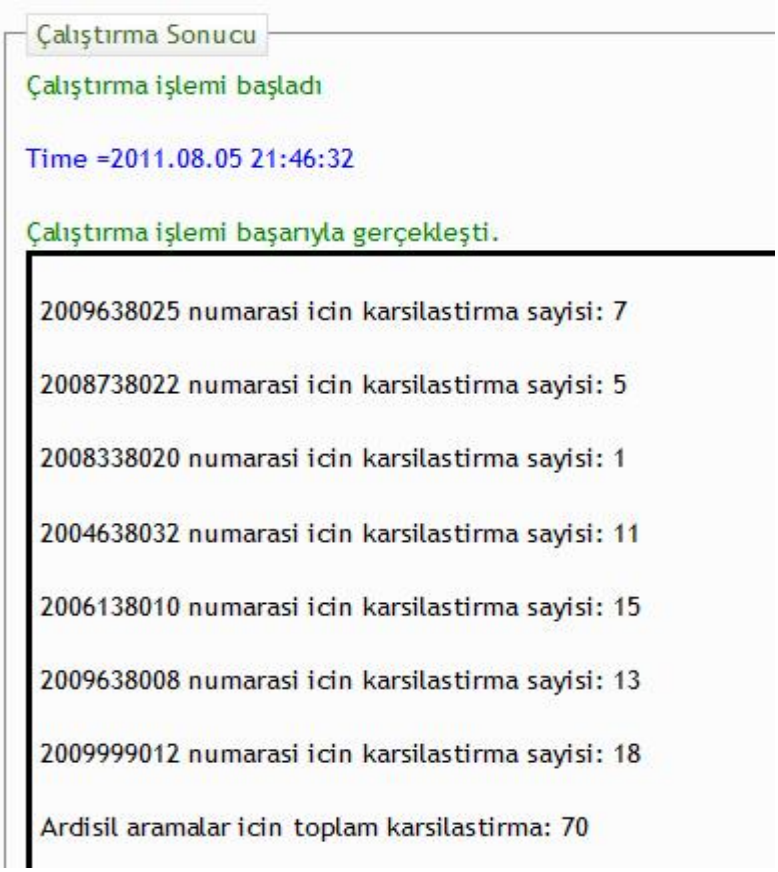


Figure 3.26. The Execution of the Selected Assignment

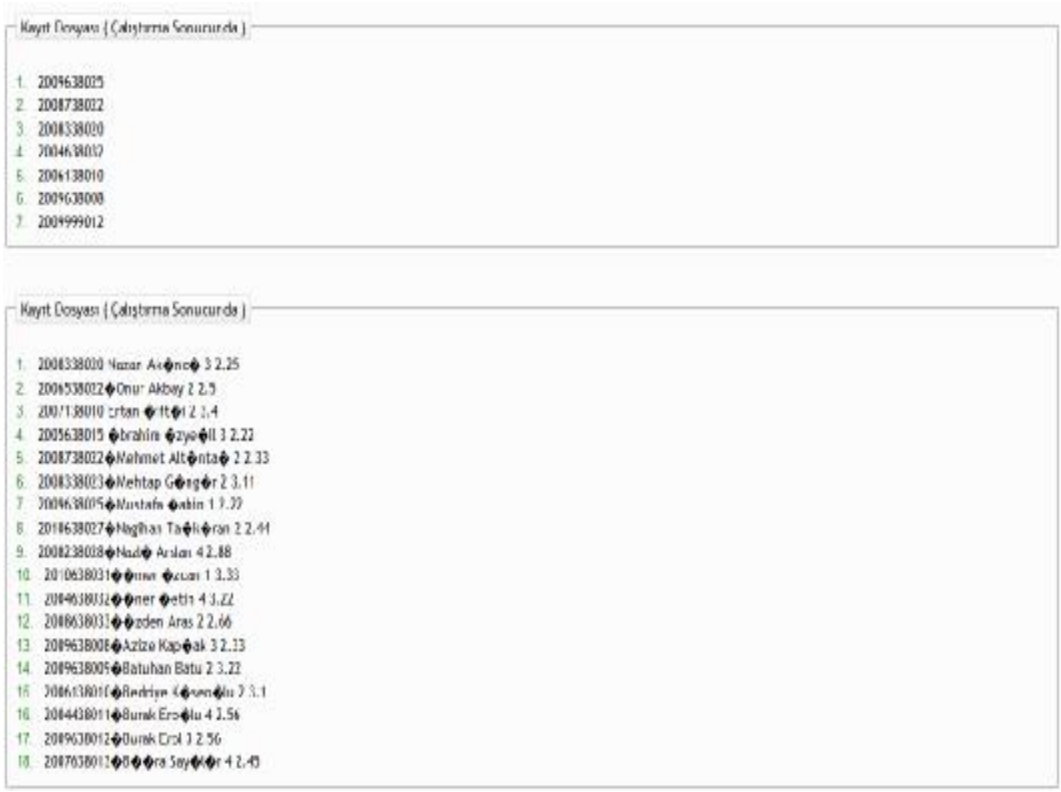


Figure 3.27. The Execution of the Selected Assignment - 2

Figure 3.28. Moodle Web Page of Viewing Assignment Content - 2

"Moss ile Karşılaştır" button, which is shown in Figure 3.28, is used for comparing submitted assignments with the help of Moss. The submitted assignments are sent to Moss server. They are compared on the remote server and the result is sent to back. Then it is shown on a table with their similarity rate as shown in Figure 3.29. Moss also highlights individual passages in programs that appear the same shown as in Figure 3.30, so it makes easy to quickly compare the files.

File 1	File 2	Lines Matched
[illegible]	[illegible]	149
[illegible]	[illegible]	149
[illegible]	[illegible]	149
[illegible]	[illegible]	276
[illegible]	[illegible]	146
[illegible]	[illegible]	146
[illegible]	[illegible]	146

Figure 3.29. Comparing Assignments with Moss

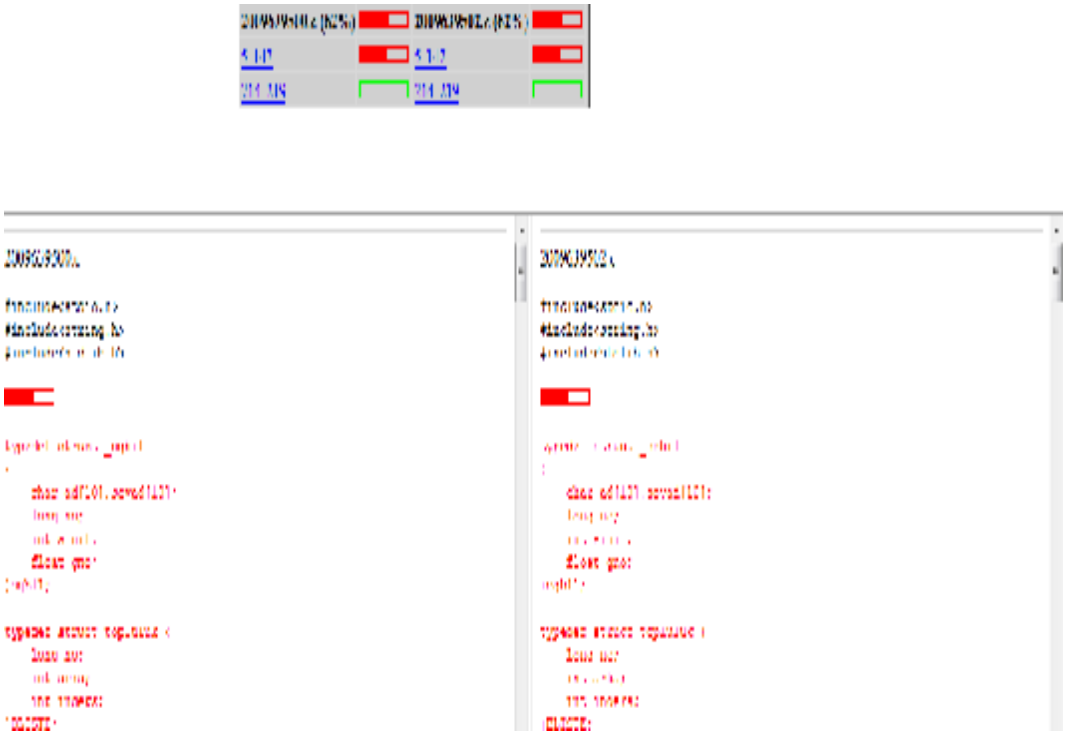


Figure 3.30. Result of the Compared Assignments with Moss

3.2.2. Our Hybrid Plagiarism Detection Algorithm

In this study a hybrid algorithm which is a combination of GST Based Source Line Matching, GST Based Comment Line Matching, LCS Based Semantic Sequence Matching, and Hash Based Word Matching Algorithm is proposed. This hybrid algorithm can be easily used to detect plagiarism on the programming assignments that are written in C programming languages, and this helps to increase the quality of the computer engineering education by forcing the students to make their own homeworks.

As shown in Figure 3.31, some preprocessing is done on the input source codes. Then all matching algorithms produce a similarity score. A single score is obtained by taking the weighted average of these scores for each file pairs. If a file pair have a high score, it is implied that these files are similar and may be plagiarized from each other or from a common third file.

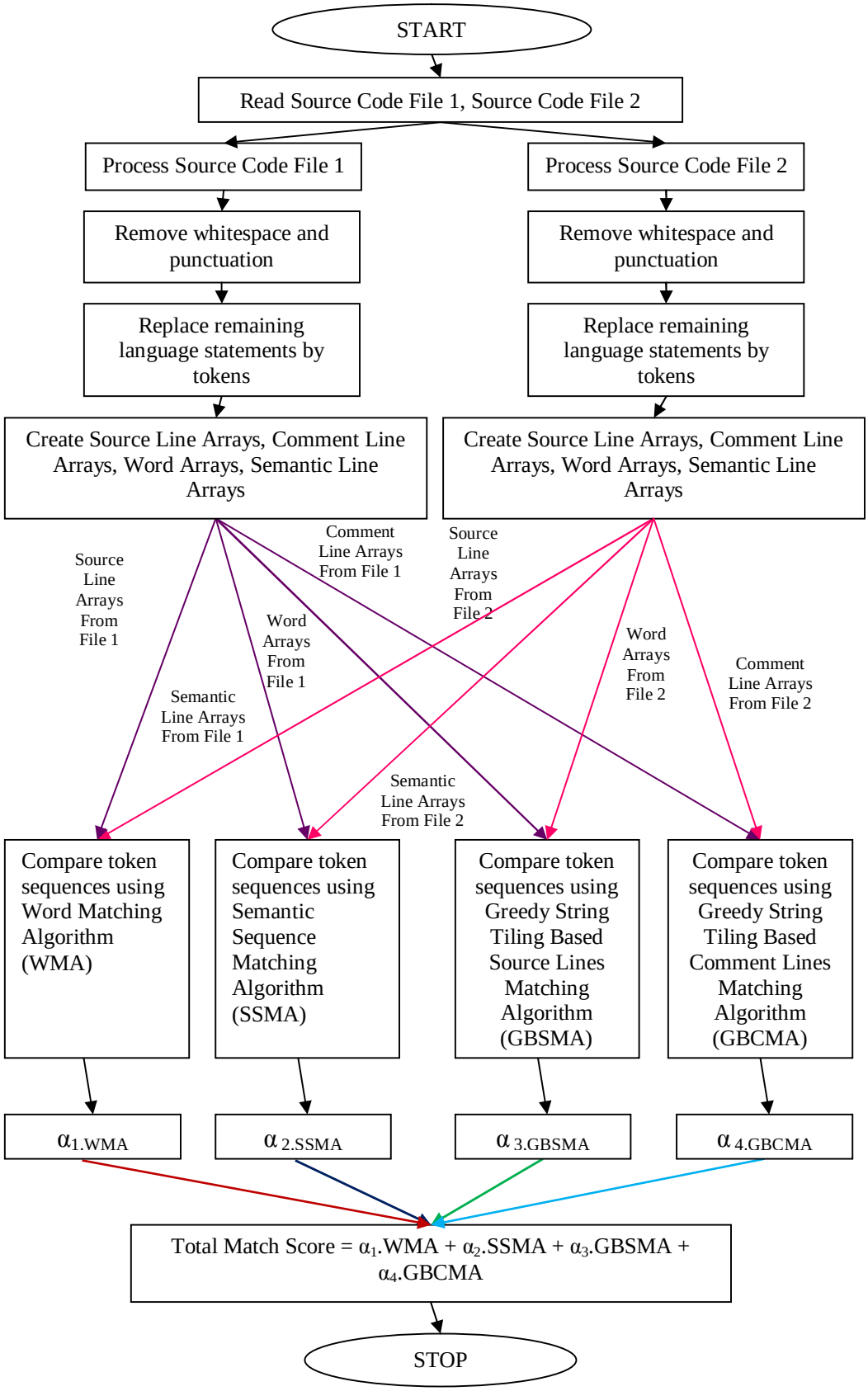


Figure 3.31. Source Code Matching Algorithm Flowchart

"Karşılaştır" button of the Moodle web page as shown in Figure 3.28 for controlling the assignments of students is used for comparing submitted assignments with the hybrid algorithm, which is consist of Greedy String Tiling, Source Line Matching, Comment Line Matching, Word Matching and Semantic Sequence Matching Algorithms. On the top of the page, there is a table on which are showed the name of assignment, the number of submissions, the name of submitted assignments, the date and the minimum match length. Then a table is created dynamically. It takes the result from a database. On the table, there are names of assignments and their similarity rate between each other. For example, four assignments are assumed to load to Moodle. Firstly, the first assignment is compared with the second assignment. The match score is calculated and it is written to the database. Then the first assignment is compared with the third assignment. The match score is written to the database as a different record. Then the first assignment is compared with the fourth assignment ultimately. The calculated match score is written to database again. So the first assignment is compared with the all submitted assignments. The order is now on the second assignment. The same processes are made for the second and third assignments. When the order is on the third assignment, the same processes are made for the third assignment. Therefore all assignments are compared with the all submitted assignments, too. The comparison is not a symmetrical comparison so all comparisons are made only once.

3.2.2.1. Preprocessing of Source Codes

All source code programs to be compared are parsed firstly. Then whitespace and punctuations are removed from the files. For source lines matching the comment lines and blank lines are removed from the files. For comment lines matching the source lines and blank lines are removed from the files. For word matching blank lines and all words except identifiers are removed from the files. For Semantic Sequence matching the comment lines and blank lines are removed from the files. After this preprocessing, remaining language statements are replaced by tokens.

3.2.2.2. Tokenization

The tokens are created according to the C programming language. For instance, the C language tokens include keywords like ‘void’, ‘int’ etc. Separate tokens are required for these keywords individually. The punctuations like ‘#’, ‘;’, ‘,’, ’:’ are not transformed into tokens, but the punctuations like ‘(’, ‘)’, ‘[’, ‘]’, ‘{’, ‘}’, which are used mostly in C language with functions, are transformed into tokens. A few examples of the formed tokens are given in Table 3.2.

3.2.2.3. Greedy String Tiling Algorithm

Greedy String Tiling Algorithm is the basis algorithm in this study. The other algorithms base on Greedy String Tiling Algorithm. It is selected as the basis algorithm in this study, because it gives more accurate results, and it detects more successful similarities when Greedy String Tiling Algorithm is compared with the other algorithms. It’s important phase is tokenization.

Table 3.2. Tokenization

KEYWORDS	GENERATED TOKENS
int/char/long/short/ double/ float/ unsigned	@WORD_INT/ @WORD_CHAR/ @WORD_LONG/ @WORD_SHORT/ @WORD_DOUBLE/ @WORD_FLOAT/ @WORD_UNSIGNED
#include	@HEADER
"+" "-" "*" "/"	@OPERATOR_PLUS/ @OPERATOR_MINUS/ @OPERATOR_MUL/ @OPERATOR_DIV
void	@VOID
main	@FUNCTION
#define	@DEFINE
static	@STATIC
FILE	@FILE
/* */ //	@COMMENT
<= >= != == <>	@COMPARE
&&	@AND
||	@OR
&	@BITWISE_AND
|	@BITWISE_OR
= += -= *= /= %=	@ASSIGN
" "	@STRING
' '	@CHARACTER
printf	@APPLY_PRINTF
scanf	@APPLY_SCANF
fprintf	@APPLY_FPRINTF
fscanf	@APPLY_FSCANF
fopen	@APPLY_FOPEN
fclose	@APPLY_FCLOSE
fwrite	@APPLY_FWRITE
fread	@APPLY_FREAD
fseek	@APPLY_FSEEK
feof	@APPLY_FEOF

Table 3.2. Con't.

KEYWORDS	GENERATED TOKENS
puts	@APPLY_PUTS
gets	@APPLY_GETS
fputs	@APPLY_FPUTS
fgets	@APPLY_FGETS
getchar	@APPLY_GETCHAR
putchar	@APPLY_PUTCHAR
getch	@APPLY_GETCH
putch	@APPLY_PUTCH
getc	@APPLY_GETC
sizeof	@APPLY_SIZEOF
strcmp	@APPLY_STRCMP
strncmp	@APPLY_STRNCMP
strcpy	@APPLY_STRCPY
strncpy	@APPLY_STRNCPY
strcat	@APPLY_STRCAT
strncat	@APPLY_STRNCAT
strlen	@APPLY_STRLEN
strstr	@APPLY_STRSTR
strrev	@APPLY_STRREV
strlwr	@APPLY_STRLWR
strupr	@APPLY_STRUPR
strchr	@APPLY_STRCHR
strnchr	@APPLY_STRNCHR
atoi	@ATOI
atof	@ATOF
return	@RETURN
if	@IF
else	@ELSE
switch	@SWITCH_CONTROL
case	@CASE_CONTROL

Table 3.2. Con't.

KEYWORDS	GENERATED TOKENS
for	@FOR_CONTROL
while	@WHILE_CONTROL
do	@DOWHILE_CONTROL
goto	@GOTO
struct	@STRUCT
default	@DEFAULT
break	@BREAK
continue	@CONTINUE
typedef	@TYPEDEF
enum	@ENUM
union	@UNION
const	@CONST
NULL	@NULL
exit	@EXIT
1-2-3-4-5-6-7-8-9-0	@NUMBER
a-zA-Z_	@USER_DEFINED
{	@BEGIN
}	@END
(	@OPEN_PARENTHESIS
)	@CLOSE_PARENTHESIS
[	@OPEN_BRACKET
]	@CLOSE_BRACKET

3.2.2.4. GST Based Source Line Matching Algorithm

Greedy String Tiling Algorithm is used for making comparison in the Source Line Matching Algorithm. The reason of integrating Greedy String Tiling Algorithm to Source Line Matching Algorithm is that JPlag and CodeMatch are the most successful ones among the plagiarism detection tools. Our GST based source line matching algorithm works on only source lines of the source code. The whitespace characters, punctuation and the comments are removed from the source code. The words of the remaining lines are converted to tokens in the preprocessing step. For every unmarked token in the first source code file an identical token in the second source code file is searched. In this algorithm minimum match length is taken as a value of 3. The tokens are compared in order to extend the match as far as possible. After maximum match is found, a similarity rate is produced and it is written to the database. It is then weighted when all algorithms are finished to run. The process time is calculated and written to the database also.

3.2.2.5. GST Based Comment Line Matching Algorithm

Greedy String Tiling Algorithm is used for the comparison in the Comment Line Matching Algorithm. It compares only comment lines. The source code lines, punctuation and whitespace characters are removed from the file in the preprocessing step. The same processes in GST Based Source Code Line Matching Algorithm are applied in this algorithm, too. Only tokenization process is not performed for this algorithm. In this algorithm minimum match length is taken as a value of 1. This algorithm begins by looking for an identical comment word in the second file. The words are compared in order to extend the match as far as possible. After a maximum match is found, this algorithm produces a score and it is written to the database. It is then weighted when all algorithms are finished to run. The process time is calculated and written to the database too.

3.2.2.6. LCS Based Semantic Sequence Matching Algorithm

The Longest Common Subsequence Algorithm is used to find the longest sequences that are subsequences of two or more sequences. In this study Semantic Sequence Matching Algorithm is used with Longest Common Subsequence Algorithm. It tries to find a maximum match of the source code. When it finds a match, then it checks whether this match is at least as long as the longest ones found before. It produces a score and it is then weighted when all algorithms are finished to run. The process time is calculated and written to the database.

3.2.2.7. Hash Based Word Matching Algorithm

A hash table is a data structure that increases the search efficiency. A hash function provides a way for assigning numbers to the input data such that the data can then be stored at the hash table (array) with the index corresponding to the assigned number. In this study hash data structure of Perl is used for Word Matching Algorithm. Because in this way accessing the searched data is more easy and faster. It reads all words, which are identifiers, from the first source code file to the hash. Then it reads all words, which consists of identifiers, in the second source code file and each identifier in the second file is searched from the hash. If the identifier is found in the hash, then a match occurs. The number of identifiers in each file is calculated. The similarity rate is calculated according to the source code having less number of identifiers. The similarity score is computed according to Equation 3.1.

$$\text{WordMatchScore} = \left[. \frac{\text{Number of matched identifiers}}{\min_{i \in \{1,2\}} (\text{number of identifiers in file i})} \right] * 100 \qquad (3.1)$$

The similarity rate is then written to the database. It is then weighted when all other similarity algorithms are finished to run. The process time is calculated and written to the database.

3.2.2.8. Computation of the Total Match Score

Each score of each individual algorithm is weighted and the Total Match Score is calculated according to Equation 3.2. These weights are adjusted to give the optimal results. α is the weight coefficient, WMA is the score of the Word Matching Algorithm, SSMA denotes the score of the Semantic Sequence Matching Algorithm, GBSMA is the score of the Greedy String Tiling Based Source Line Matching Algorithm, GBCMA represents the score computed by the Greedy String Tiling Based Comment Line Matching Algorithm on Equation 3.2. Then each algorithm is produced a match score. After that each score is weighted to form the Total Match Score.

$$\text{TotalMatchScore} = \alpha_1.\text{WMA} + \alpha_2.\text{SSMA} + \alpha_3.\text{GBSMA} + \alpha_4.\text{GBCMA} \quad (3.2)$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1 \quad (3.3)$$

Over time, these weights compare results of copied files and copied but changed files. α_1 is taken as 0.097, α_2 is taken as 0.001, α_3 is taken as 0.9 and α_4 is taken as 0.001 in Equation 3.3 and these α values are determined experimentally. In the end of this study an HTML output report with a list of file pairs ordered by their Total Match Scores is produced. Users are able to click on a file-pair hyperlink to see a detailed HTML report. In this way, users are directed to suspicious similarities and they are allowed to make their own judgments.

4. RESEARCH AND DISCUSSION

In this section experiments performed and their results are presented. Perl programming language (http://www.perl.org/) was used for implementing our hybrid algorithm.

The proposed method was tested under Microsoft Windows 7 Ultimate operating system. The hardware used in the experiments had 2 GB of RAM and Intel Core2Duo T7500 2.20 GHz processor. Our proposed method was tested on the Moodle Web Page of Data Structures Course Assignments in Computer Engineering Department of Çukurova University (http://mmf.cu.edu.tr/mkaya/moodle/course/view.php?id=51). In this study to run Apache (http://www.apache.org/), MySql (http://www.mysql.com/), Perl source codes and PHP (http://www.php.net/) source codes, XAMPP (http://www.apachefriends.org/en/xampp.html) program was installed on a computer. To run Perl on a Windows computer, Strawberry Perl (http://strawberryperl.com/) was installed.

The proposed method was compared with for Moss and JPlag in ten scenarios.

4.1. The First Scenario

The first scenario contains 10 short assignments which are completely the same with each other. The source code used in this scenario is shown Figure 4.1. It has 22 lines and it has no comment lines.

```
#include <stdio.h>
int main ( void )
{
  static const char filename[] = "file.txt";
  FILE *file = fopen ( filename, "r" );
  if ( file != NULL )
  {
    char line [ 128 ]; /* or other suitable maximum line size */
    while ( fgets ( line, sizeof line, file ) != NULL ) /* read a line */
    {
      fputs ( line, stdout ); /* write the line */
    }
    fclose ( file );
  }
  else
  {
    perror ( filename ); /* why didn't the file open? */
  }
  return 0;
}
```

Figure 4.1. The Example of Compared Assignment For Scenario 1

The assignments are compared with Moss firstly. To run Moss script which is written in Perl, the commands given in Figure 4.2 should be written in DOS command prompt.

```
C:\xampp>perl Moss.pl -l c
c:\xampp\senaryo1\2011638001_odev1.c c:\xampp\senaryo1\2011638002_odev1.c
c:\xampp\senaryo1\2011638003_odev1.c c:\xampp\senaryo1\2011638004_odev1.c
c:\xampp\senaryo1\2011638005_odev1.c c:\xampp\senaryo1\2011638006_odev1.c
c:\xampp\senaryo1\2011638007_odev1.c c:\xampp\senaryo1\2011638008_odev1.c
c:\xampp\senaryo1\2011638009_odev1.c c:\xampp\senaryo1\2011638010_odev1.c
```

Figure 4.2. Moss Script For Scenario 1

The result page of Moss is shown in Figure 4.3. The similarity scores obtained by Moss for this scenario is presented in Table 4.1. As shown as in Table 4.1, File 1 represents the assignment of the first user. File 10 represents the assignment of the last user.

The expected result is that the similarity rates of all assignments with each other are 100%. But the observed similarity rates are 95%. 19 lines of 22 lines are matched in every file with each other. Moss takes 3 seconds to compare these 10 files with each other.

Table 4.1. Moss Results as a Table For Scenario 1

MOSS	**File1**	**File2**	**File3**	**File4**	**File5**	**File6**	**File7**	**File8**	**File9**	**File10**
File1	-	95 %	95 %	95 %	95 %	95 %	95 %	95 %	95 %	95 %
File2	-	-	95 %	95 %	95 %	95 %	95 %	95 %	95 %	95 %
File3	-	-	-	95 %	95 %	95 %	95 %	95 %	95 %	95 %
File4	-	-	-	-	95 %	95 %	95 %	95 %	95 %	95 %
File5	-	-	-	-	-	95 %	95 %	95 %	95 %	95 %
File6	-	-	-	-	-	-	95 %	95 %	95 %	95 %
File7	-	-	-	-	-	-	-	95 %	95 %	95 %
File8	-	-	-	-	-	-	-	-	95 %	95 %
File9	-	-	-	-	-	-	-	-	-	95 %
File10	-	-	-	-	-	-	-	-	-	-

Moss Results

Fri Jul 22 03:20:36 PDT 2011

Options -l c -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
c:\xampp\senaryo1\2011638009_odev1.c (95%)	c:\xampp\senaryo1\2011638010_odev1.c (95%)	19
c:\xampp\senaryo1\2011638008_odev1.c (95%)	c:\xampp\senaryo1\2011638010_odev1.c (95%)	19
c:\xampp\senaryo1\2011638008_odev1.c (95%)	c:\xampp\senaryo1\2011638009_odev1.c (95%)	19
c:\xampp\senaryo1\2011638007_odev1.c (95%)	c:\xampp\senaryo1\2011638010_odev1.c (95%)	19
c:\xampp\senaryo1\2011638007_odev1.c (95%)	c:\xampp\senaryo1\2011638009_odev1.c (95%)	19
c:\xampp\senaryo1\2011638007_odev1.c (95%)	c:\xampp\senaryo1\2011638008_odev1.c (95%)	19
c:\xampp\senaryo1\2011638006_odev1.c (95%)	c:\xampp\senaryo1\2011638010_odev1.c (95%)	19
c:\xampp\senaryo1\2011638006_odev1.c (95%)	c:\xampp\senaryo1\2011638009_odev1.c (95%)	19
c:\xampp\senaryo1\2011638006_odev1.c (95%)	c:\xampp\senaryo1\2011638008_odev1.c (95%)	19
c:\xampp\senaryo1\2011638006_odev1.c (95%)	c:\xampp\senaryo1\2011638007_odev1.c (95%)	19
c:\xampp\senaryo1\2011638005_odev1.c (95%)	c:\xampp\senaryo1\2011638010_odev1.c (95%)	19
c:\xampp\senaryo1\2011638005_odev1.c (95%)	c:\xampp\senaryo1\2011638009_odev1.c (95%)	19
c:\xampp\senaryo1\2011638005_odev1.c (95%)	c:\xampp\senaryo1\2011638008_odev1.c (95%)	19
c:\xampp\senaryo1\2011638005_odev1.c (95%)	c:\xampp\senaryo1\2011638007_odev1.c (95%)	19
c:\xampp\senaryo1\2011638005_odev1.c (95%)	c:\xampp\senaryo1\2011638006_odev1.c (95%)	19
c:\xampp\senaryo1\2011638004_odev1.c (95%)	c:\xampp\senaryo1\2011638010_odev1.c (95%)	19
c:\xampp\senaryo1\2011638004_odev1.c (95%)	c:\xampp\senaryo1\2011638009_odev1.c (95%)	19
c:\xampp\senaryo1\2011638004_odev1.c (95%)	c:\xampp\senaryo1\2011638008_odev1.c (95%)	19
c:\xampp\senaryo1\2011638004_odev1.c (95%)	c:\xampp\senaryo1\2011638007_odev1.c (95%)	19
c:\xampp\senaryo1\2011638004_odev1.c (95%)	c:\xampp\senaryo1\2011638006_odev1.c (95%)	19
c:\xampp\senaryo1\2011638004_odev1.c (95%)	c:\xampp\senaryo1\2011638005_odev1.c (95%)	19
c:\xampp\senaryo1\2011638003_odev1.c (95%)	c:\xampp\senaryo1\2011638010_odev1.c (95%)	19
c:\xampp\senaryo1\2011638003_odev1.c (95%)	c:\xampp\senaryo1\2011638009_odev1.c (95%)	19
c:\xampp\senaryo1\2011638003_odev1.c (95%)	c:\xampp\senaryo1\2011638008_odev1.c (95%)	19
c:\xampp\senaryo1\2011638003_odev1.c (95%)	c:\xampp\senaryo1\2011638007_odev1.c (95%)	19
c:\xampp\senaryo1\2011638003_odev1.c (95%)	c:\xampp\senaryo1\2011638006_odev1.c (95%)	19
c:\xampp\senaryo1\2011638003_odev1.c (95%)	c:\xampp\senaryo1\2011638005_odev1.c (95%)	19
c:\xampp\senaryo1\2011638003_odev1.c (95%)	c:\xampp\senaryo1\2011638004_odev1.c (95%)	19
c:\xampp\senaryo1\2011638002_odev1.c (95%)	c:\xampp\senaryo1\2011638010_odev1.c (95%)	19
c:\xampp\senaryo1\2011638002_odev1.c (95%)	c:\xampp\senaryo1\2011638009_odev1.c (95%)	19
c:\xampp\senaryo1\2011638002_odev1.c (95%)	c:\xampp\senaryo1\2011638008_odev1.c (95%)	19
c:\xampp\senaryo1\2011638002_odev1.c (95%)	c:\xampp\senaryo1\2011638007_odev1.c (95%)	19
c:\xampp\senaryo1\2011638002_odev1.c (95%)	c:\xampp\senaryo1\2011638006_odev1.c (95%)	19
c:\xampp\senaryo1\2011638002_odev1.c (95%)	c:\xampp\senaryo1\2011638005_odev1.c (95%)	19
c:\xampp\senaryo1\2011638002_odev1.c (95%)	c:\xampp\senaryo1\2011638004_odev1.c (95%)	19
c:\xampp\senaryo1\2011638002_odev1.c (95%)	c:\xampp\senaryo1\2011638003_odev1.c (95%)	19
c:\xampp\senaryo1\2011638001_odev1.c (95%)	c:\xampp\senaryo1\2011638010_odev1.c (95%)	19
c:\xampp\senaryo1\2011638001_odev1.c (95%)	c:\xampp\senaryo1\2011638009_odev1.c (95%)	19
c:\xampp\senaryo1\2011638001_odev1.c (95%)	c:\xampp\senaryo1\2011638008_odev1.c (95%)	19
c:\xampp\senaryo1\2011638001_odev1.c (95%)	c:\xampp\senaryo1\2011638007_odev1.c (95%)	19
c:\xampp\senaryo1\2011638001_odev1.c (95%)	c:\xampp\senaryo1\2011638006_odev1.c (95%)	19
c:\xampp\senaryo1\2011638001_odev1.c (95%)	c:\xampp\senaryo1\2011638005_odev1.c (95%)	19
c:\xampp\senaryo1\2011638001_odev1.c (95%)	c:\xampp\senaryo1\2011638004_odev1.c (95%)	19
c:\xampp\senaryo1\2011638001_odev1.c (95%)	c:\xampp\senaryo1\2011638003_odev1.c (95%)	19
c:\xampp\senaryo1\2011638001_odev1.c (95%)	c:\xampp\senaryo1\2011638002_odev1.c (95%)	19

Any errors encountered during this query are listed below.

Figure 4.3. Moss Results For Scenario 1

To view the detailed comparison, user can click the name of the assignment, which the user want to compare it with the others. The detailed result page is shown in Figure 4.4.

Figure 4.4. Moss Detailed Results For Scenario 1

The assignments are then compared with JPlag. Java Web Start Application must be installed on a computer to run JPlag.

The result page of JPlag is shown in Figure 4.5 and Figure 4.6. Figure 4.5 shows the name of the assignment, the file names, the number of submissions, the minimum match length, the programming language, the suffixes, the date, the threshold and the directory. Figure 4.6 shows the average similarity. JPlag has two results as a similarity rate: Average Similarity Rate and Maximum Similarity Rate. The maximum similarity is defined as the maximum of both program coverages.

This ranking is especially useful if the programs are very different in size. This can happen when dead code was inserted to disguise the origin of the plagiarized program. The average similarity is defined as the average of both program coverages. This is the default similarity which works in most cases. Matches with a high average similarity indicate that the programs work in a very similar way. That's why the average similarity is based on the main JPlag Similarity in this thesis.

Figure 4.5. JPlag Results For Scenario 1

Figure 4.6. JPlag Average Similarity Results For Scenario 1

To view the detailed result of similarities, user can click the name of the assignment, which the user wants to compare with the others. The detailed result page is shown in Figure 4.7.

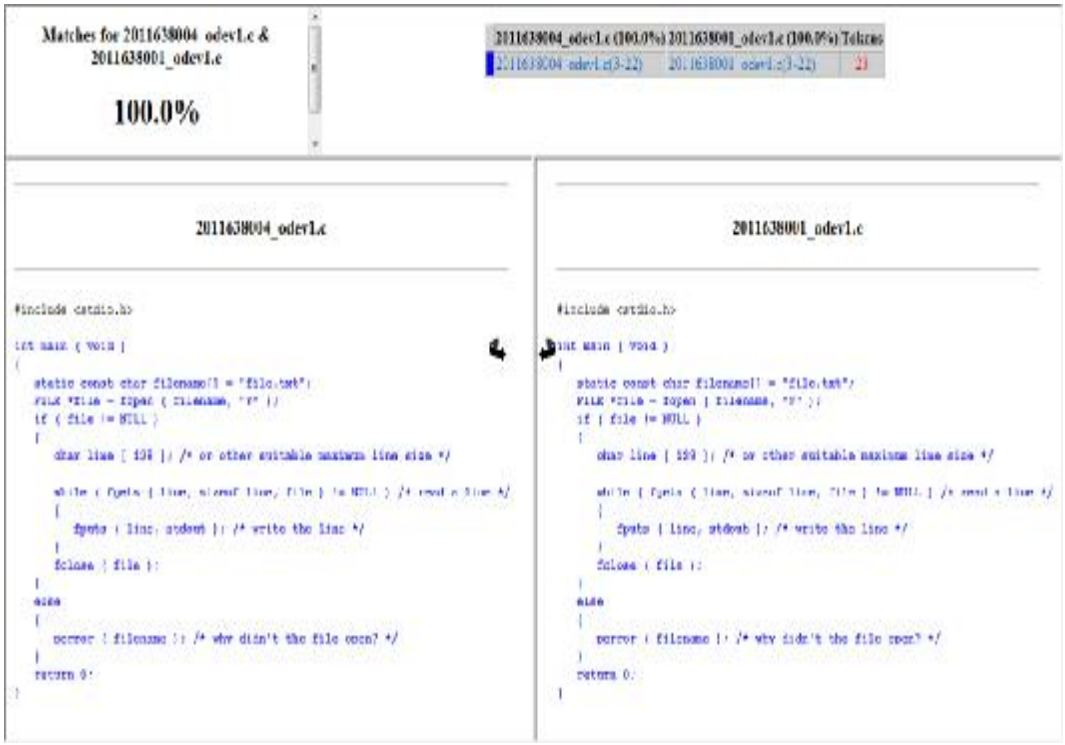


Figure 4.7. JPlag Detailed Result For Scenario 1

As shown in Table 4.2, JPlag can find the similarity rates between the files as 100%.

Table 4.2. JPlag Results as a Table For Scenario 1

JPlag	**File1**	**File2**	**File3**	**File4**	**File5**	**File6**	**File7**	**File8**	**File9**	**File10**
File1	-	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
File2	-	-	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
File3	-	-	-	100 %	100 %	100 %	100 %	100 %	100 %	100 %
File4	-	-	-	-	100 %	100 %	100 %	100 %	100 %	100 %
File5	-	-	-	-	-	100 %	100 %	100 %	100 %	100 %
File6	-	-	-	-	-	-	100 %	100 %	100 %	100 %
File7	-	-	-	-	-	-	-	100 %	100 %	100 %
File8	-	-	-	-	-	-	-	-	100 %	100 %
File9	-	-	-	-	-	-	-	-	-	100 %
File10	-	-	-	-	-	-	-	-	-	-

Finally the assignments are compared with our hybrid algorithm which runs under Moodle.

The result page is shown in Figure 4.8 and Figure 4.9. Figure 4.8 shows the name of the assignment, the file names, the number of submissions, the minimum match length, the programming language of the assignment, the suffixes and the date of the comparison. Figure 4.9. shows the match score of the submitted assignments.

Table 4.3 summarizes the results of our hybrid algorithm. As shown in Table 4.3. File 1 represents the assignment of the first user. File 10 represents the assignment of the last user. This table is a comprehensible presentation of the Figure 4.9. The expected result is that the similarity rates of all assignments with each other are 100% and the observed similarity rates are 100%. So, 22 lines of 22 lines are matched in every file with each other.

Table 4.3. The Hybrid Algorithm Results As a Table For Scenario 1

HYBRID	File1	File2	File3	File4	File5	File6	File7	File8	File9	File10
File1	-	100%	100%	100%	100%	100%	100%	100%	100%	100 %
File2	-	-	100%	100%	100%	100%	100%	100%	100%	100 %
File3	-	-	-	100%	100%	100%	100%	100%	100%	100 %
File4	-	-	-	-	100%	100%	100%	100%	100%	100 %
File5	-	-	-	-	-	100%	100%	100%	100%	100 %
File6	-	-	-	-	-	-	100%	100%	100%	100 %
File7	-	-	-	-	-	-	-	100%	100%	100 %
File8	-	-	-	-	-	-	-	-	100%	100 %
File9	-	-	-	-	-	-	-	-	-	100 %
File10	-	-	-	-	-	-	-	-	-	-

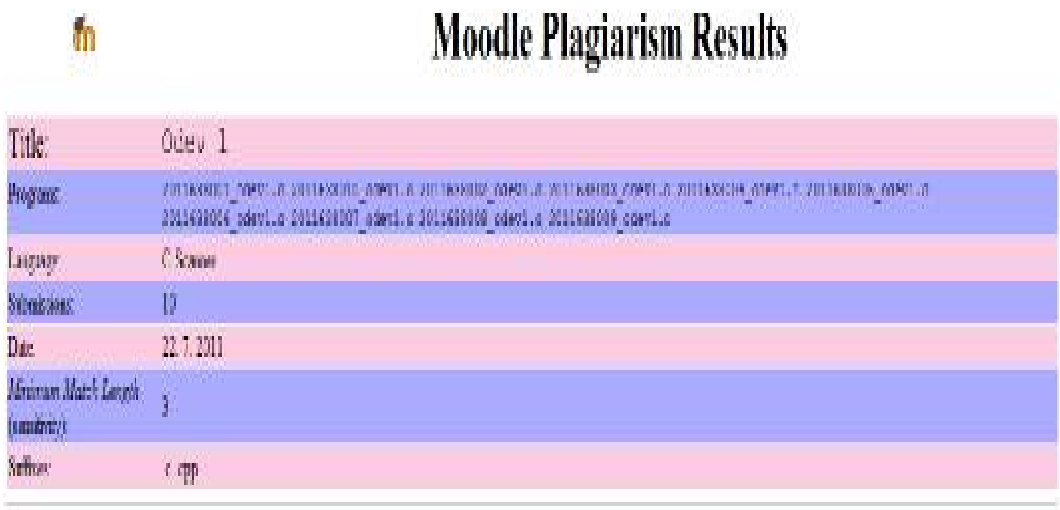


Figure 4.8. The Result Page of This Hybrid Algorithm For Scenario 1

SIMILARITY RATE :

File1	File2	MatchScore
2011638010_odev1.c	2011638006_odev1.c	% 100.00
2011638002_odev1.c	2011638004_odev1.c	% 100.00
2011638002_odev1.c	2011638009_odev1.c	% 100.00
2011638003_odev1.c	2011638008_odev1.c	% 100.00
2011638001_odev1.c	2011638002_odev1.c	% 100.00
2011638004_odev1.c	2011638008_odev1.c	% 100.00
2011638001_odev1.c	2011638007_odev1.c	% 100.00
2011638005_odev1.c	2011638009_odev1.c	% 100.00
2011638010_odev1.c	2011638004_odev1.c	% 100.00
2011638007_odev1.c	2011638009_odev1.c	% 100.00
2011638010_odev1.c	2011638009_odev1.c	% 100.00
2011638002_odev1.c	2011638007_odev1.c	% 100.00
2011638003_odev1.c	2011638006_odev1.c	% 100.00
2011638004_odev1.c	2011638006_odev1.c	% 100.00
2011638001_odev1.c	2011638005_odev1.c	% 100.00
2011638005_odev1.c	2011638007_odev1.c	% 100.00
2011638010_odev1.c	2011638002_odev1.c	% 100.00
2011638006_odev1.c	2011638009_odev1.c	% 100.00
2011638010_odev1.c	2011638007_odev1.c	% 100.00
2011638002_odev1.c	2011638005_odev1.c	% 100.00
2011638003_odev1.c	2011638004_odev1.c	% 100.00
2011638003_odev1.c	2011638009_odev1.c	% 100.00
2011638001_odev1.c	2011638003_odev1.c	% 100.00
2011638004_odev1.c	2011638009_odev1.c	% 100.00
2011638001_odev1.c	2011638008_odev1.c	% 100.00
2011638006_odev1.c	2011638007_odev1.c	% 100.00
2011638010_odev1.c	2011638005_odev1.c	% 100.00
2011638008_odev1.c	2011638009_odev1.c	% 100.00
2011638002_odev1.c	2011638003_odev1.c	% 100.00
2011638002_odev1.c	2011638008_odev1.c	% 100.00
2011638003_odev1.c	2011638007_odev1.c	% 100.00
2011638001_odev1.c	2011638010_odev1.c	% 100.00
2011638004_odev1.c	2011638007_odev1.c	% 100.00
2011638001_odev1.c	2011638006_odev1.c	% 100.00
2011638005_odev1.c	2011638008_odev1.c	% 100.00
2011638010_odev1.c	2011638003_odev1.c	% 100.00
2011638007_odev1.c	2011638008_odev1.c	% 100.00
2011638010_odev1.c	2011638008_odev1.c	% 100.00
2011638002_odev1.c	2011638006_odev1.c	% 100.00
2011638003_odev1.c	2011638005_odev1.c	% 100.00
2011638004_odev1.c	2011638005_odev1.c	% 100.00
2011638001_odev1.c	2011638004_odev1.c	% 100.00
2011638005_odev1.c	2011638006_odev1.c	% 100.00
2011638001_odev1.c	2011638009_odev1.c	% 100.00
2011638006_odev1.c	2011638008_odev1.c	% 100.00

Detayli Göster

Figure 4.9. The Result of This Hybrid Algorithm For Scenario 1

The detailed result page that our hybrid algorithm produces is shown in Figure 4.10. It shows that the assignments are grouped and compared with each other. Source code similarity rate, comment line similarity rate, word matching similarity rate, semantic sequence similarity rate and match score which are all 100% are showed in Figure 4.10.

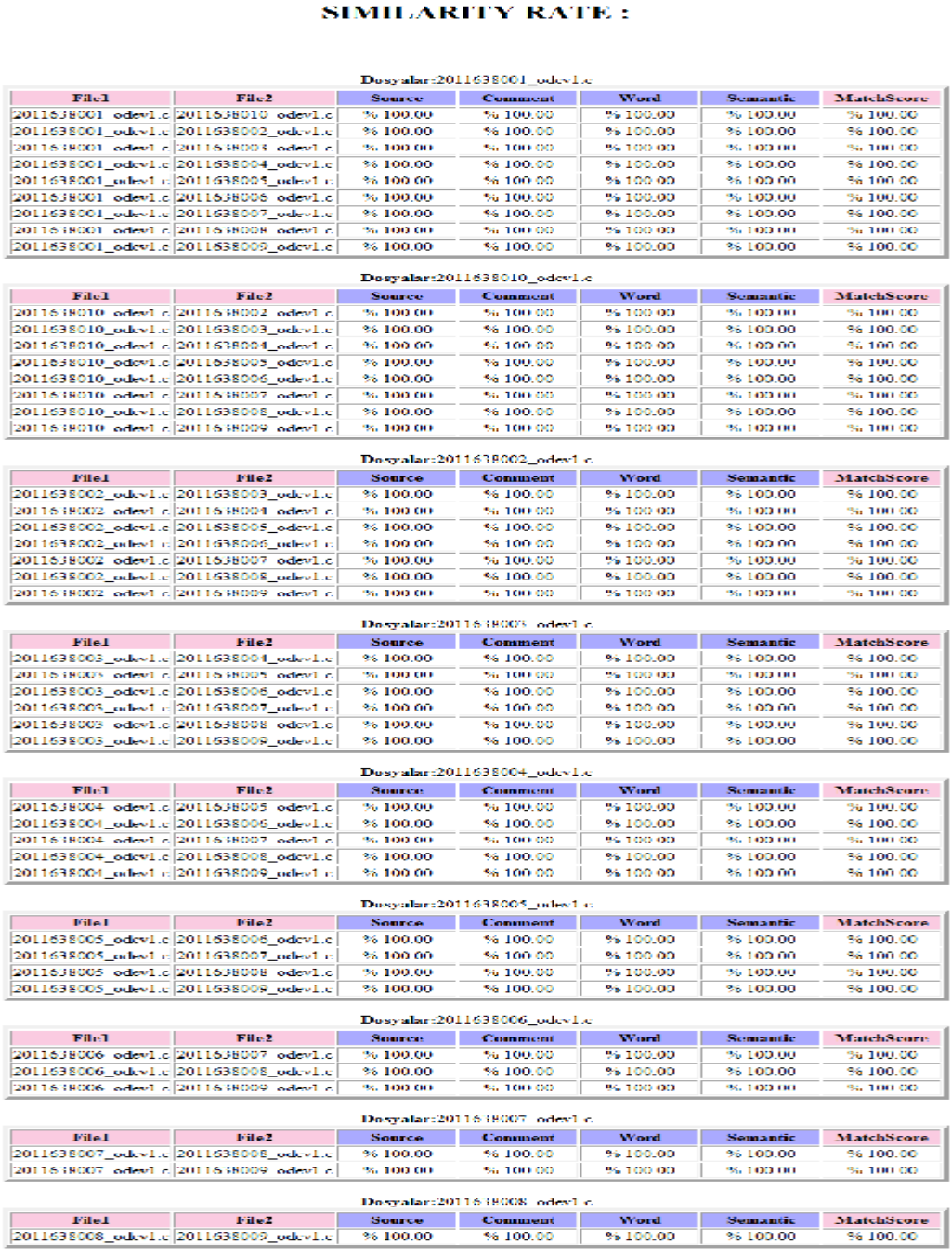


Figure 4.10. The Detailed Result of This Hybrid Algorithm For Scenario 1

The numbers of matched tokens computed by our algorithm for every file are calculated and shown in Figure 4.11. and written to the database.

NUMBERS of MATCHED TOKENS:

File1	File2	SourceToken	CommentToken	WordToken	MaxSemanticToken
2011638010_odev1.c	2011638003_odev1.c	70	25	6	20
2011638010_odev1.c	2011638009_odev1.c	70	25	6	20
2011638010_odev1.c	2011638008_odev1.c	70	25	6	20
2011638010_odev1.c	2011638007_odev1.c	70	25	6	20
2011638010_odev1.c	2011638006_odev1.c	70	25	6	20
2011638010_odev1.c	2011638005_odev1.c	70	25	6	20
2011638010_odev1.c	2011638004_odev1.c	70	25	6	20
2011638010_odev1.c	2011638002_odev1.c	70	25	6	20
2011638008_odev1.c	2011638009_odev1.c	70	25	6	20
2011638007_odev1.c	2011638008_odev1.c	70	25	6	20
2011638007_odev1.c	2011638009_odev1.c	70	25	6	20
2011638006_odev1.c	2011638008_odev1.c	70	25	6	20
2011638006_odev1.c	2011638007_odev1.c	70	25	6	20
2011638006_odev1.c	2011638009_odev1.c	70	25	6	20
2011638005_odev1.c	2011638008_odev1.c	70	25	6	20
2011638005_odev1.c	2011638006_odev1.c	70	25	6	20
2011638005_odev1.c	2011638007_odev1.c	70	25	6	20
2011638005_odev1.c	2011638009_odev1.c	70	25	6	20
2011638004_odev1.c	2011638009_odev1.c	70	25	6	20
2011638004_odev1.c	2011638005_odev1.c	70	25	6	20
2011638004_odev1.c	2011638006_odev1.c	70	25	6	20
2011638004_odev1.c	2011638007_odev1.c	70	25	6	20
2011638004_odev1.c	2011638008_odev1.c	70	25	6	20
2011638003_odev1.c	2011638007_odev1.c	70	25	6	20
2011638003_odev1.c	2011638004_odev1.c	70	25	6	20
2011638003_odev1.c	2011638005_odev1.c	70	25	6	20
2011638003_odev1.c	2011638006_odev1.c	70	25	6	20
2011638003_odev1.c	2011638009_odev1.c	70	25	6	20
2011638003_odev1.c	2011638008_odev1.c	70	25	6	20
2011638002_odev1.c	2011638003_odev1.c	70	25	6	20
2011638002_odev1.c	2011638004_odev1.c	70	25	6	20
2011638002_odev1.c	2011638005_odev1.c	70	25	6	20
2011638002_odev1.c	2011638006_odev1.c	70	25	6	20
2011638002_odev1.c	2011638007_odev1.c	70	25	6	20
2011638002_odev1.c	2011638008_odev1.c	70	25	6	20
2011638002_odev1.c	2011638009_odev1.c	70	25	6	20
2011638001_odev1.c	2011638003_odev1.c	70	25	6	20
2011638001_odev1.c	2011638004_odev1.c	70	25	6	20
2011638001_odev1.c	2011638005_odev1.c	70	25	6	20
2011638001_odev1.c	2011638007_odev1.c	70	25	6	20
2011638001_odev1.c	2011638006_odev1.c	70	25	6	20
2011638001_odev1.c	2011638008_odev1.c	70	25	6	20
2011638001_odev1.c	2011638009_odev1.c	70	25	6	20
2011638001_odev1.c	2011638010_odev1.c	70	25	6	20
2011638001_odev1.c	2011638002_odev1.c	70	25	6	20

Figure 4.11. Numbers of Matched Tokens Computed By Hybrid Algorithm For Scenario 1

The run time of our hybrid algorithm takes 1 second for this scenario shown as in Figure 4.12. Source Code Matching, Comment Line Matching, Semantic Sequence Matching, Word Matching Algorithms and Tokenization processes take too little time to be measured. The Match Score process takes 1 second. So total process time is 1 second.

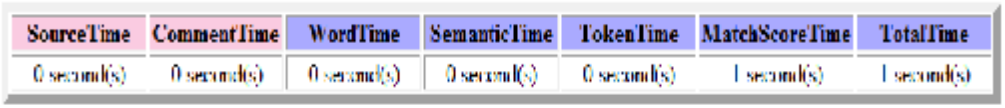


Figure 4.12. Process Time of This Hybrid Algorithm For Scenario 1

JPlag, Moss and our Hybrid Algorithm are compared with respect to similarity rate in Table 4.4. The random three matches are selected from the Scenario 1 assignments. As shown in the table, our hybrid algorithm and JPlag are more successful than Moss.

Table 4.4. Comparison of the Similarity Rate Computed By Moss JPlag and Hybrid For Scenario 1

FILE 1	**FILE 2**	**MOSS**	**JPLAG**	**HYBRID**
2011638001_odev1.c	2011638002_odev1.c	95 %	100 %	100 %
2011638003_odev1.c	2011638007_odev1.c	95 %	100 %	100 %
2011638004_odev1.c	2011638008_odev1.c	95 %	100 %	100 %

All the algorithms (i.e., Moss, JPlag, and Hybrid) are compared with respect to processing time for the first scenario and results are presented in Table 4.5. As shown in the table, for this scenario our hybrid algorithm is the fastest.

Table 4.5. Comparison of the Processing Time For Scenario 1

MOSS	**JPLAG**	**HYBRID**
3 seconds	4 seconds	1 second

4.2. The Second Scenario

The second scenario contains 10 long assignments which are completely the same with each other. The assignments are compared with Moss, JPlag and the Hybrid Algorithm. The source code used in this scenario has 285 lines in total and it has also comment lines.

The assignments are compared with Moss firstly. The required command line to run Moss for yhis scenario is given in Figure 4.13.

```
C:\xampp>perl Moss.pl -l c
c:\xampp\ scenario2\2011638001_odev2.c
c:\xampp\scenario2\2011638002_odev2.c
c:\xampp\ scenario2\2011638003_odev2.c
c:\xampp\ scenario2\2011638004_odev2.c
c:\xampp\ scenario2\2011638005_odev2.c
c:\xampp\ scenario2\2011638006_odev2.c
c:\xampp\ scenario2\2011638007_odev2.c
c:\xampp\ scenario2\2011638008_odev2.c
c:\xampp\ scenario2\2011638009_odev2.c
c:\xampp\ scenario2\2011638010_odev2.c
```

Figure 4.13. Moss Script For Scenario 2

The result page is shown in Figure 4.14.

Moss Results

Fri Jul 22 12:28:58 PDT 2011

Options -l c -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
c:\xampp\scenario2\2011638009_odev2.c (99%)	c:\xampp\scenario2\2011638010_odev2.c (99%)	270
c:\xampp\scenario2\2011638008_odev2.c (99%)	c:\xampp\scenario2\2011638010_odev2.c (99%)	270
c:\xampp\scenario2\2011638008_odev2.c (99%)	c:\xampp\scenario2\2011638009_odev2.c (99%)	270
c:\xampp\scenario2\2011638007_odev2.c (99%)	c:\xampp\scenario2\2011638010_odev2.c (99%)	270
c:\xampp\scenario2\2011638007_odev2.c (99%)	c:\xampp\scenario2\2011638009_odev2.c (99%)	270
c:\xampp\scenario2\2011638007_odev2.c (99%)	c:\xampp\scenario2\2011638008_odev2.c (99%)	270
c:\xampp\scenario2\2011638006_odev2.c (99%)	c:\xampp\scenario2\2011638010_odev2.c (99%)	270
c:\xampp\scenario2\2011638006_odev2.c (99%)	c:\xampp\scenario2\2011638009_odev2.c (99%)	270
c:\xampp\scenario2\2011638006_odev2.c (99%)	c:\xampp\scenario2\2011638008_odev2.c (99%)	270
c:\xampp\scenario2\2011638006_odev2.c (99%)	c:\xampp\scenario2\2011638007_odev2.c (99%)	270
c:\xampp\scenario2\2011638005_odev2.c (99%)	c:\xampp\scenario2\2011638010_odev2.c (99%)	270
c:\xampp\scenario2\2011638005_odev2.c (99%)	c:\xampp\scenario2\2011638009_odev2.c (99%)	270
c:\xampp\scenario2\2011638005_odev2.c (99%)	c:\xampp\scenario2\2011638008_odev2.c (99%)	270
c:\xampp\scenario2\2011638005_odev2.c (99%)	c:\xampp\scenario2\2011638007_odev2.c (99%)	270
c:\xampp\scenario2\2011638005_odev2.c (99%)	c:\xampp\scenario2\2011638006_odev2.c (99%)	270
c:\xampp\scenario2\2011638004_odev2.c (99%)	c:\xampp\scenario2\2011638010_odev2.c (99%)	270
c:\xampp\scenario2\2011638004_odev2.c (99%)	c:\xampp\scenario2\2011638009_odev2.c (99%)	270
c:\xampp\scenario2\2011638004_odev2.c (99%)	c:\xampp\scenario2\2011638008_odev2.c (99%)	270
c:\xampp\scenario2\2011638004_odev2.c (99%)	c:\xampp\scenario2\2011638007_odev2.c (99%)	270
c:\xampp\scenario2\2011638004_odev2.c (99%)	c:\xampp\scenario2\2011638006_odev2.c (99%)	270
c:\xampp\scenario2\2011638004_odev2.c (99%)	c:\xampp\scenario2\2011638005_odev2.c (99%)	270
c:\xampp\scenario2\2011638003_odev2.c (99%)	c:\xampp\scenario2\2011638010_odev2.c (99%)	270
c:\xampp\scenario2\2011638003_odev2.c (99%)	c:\xampp\scenario2\2011638009_odev2.c (99%)	270
c:\xampp\scenario2\2011638003_odev2.c (99%)	c:\xampp\scenario2\2011638008_odev2.c (99%)	270
c:\xampp\scenario2\2011638003_odev2.c (99%)	c:\xampp\scenario2\2011638007_odev2.c (99%)	270
c:\xampp\scenario2\2011638003_odev2.c (99%)	c:\xampp\scenario2\2011638006_odev2.c (99%)	270
c:\xampp\scenario2\2011638003_odev2.c (99%)	c:\xampp\scenario2\2011638005_odev2.c (99%)	270
c:\xampp\scenario2\2011638003_odev2.c (99%)	c:\xampp\scenario2\2011638004_odev2.c (99%)	270
c:\xampp\scenario2\2011638002_odev2.c (99%)	c:\xampp\scenario2\2011638010_odev2.c (99%)	270
c:\xampp\scenario2\2011638002_odev2.c (99%)	c:\xampp\scenario2\2011638009_odev2.c (99%)	270
c:\xampp\scenario2\2011638002_odev2.c (99%)	c:\xampp\scenario2\2011638008_odev2.c (99%)	270
c:\xampp\scenario2\2011638002_odev2.c (99%)	c:\xampp\scenario2\2011638007_odev2.c (99%)	270
c:\xampp\scenario2\2011638002_odev2.c (99%)	c:\xampp\scenario2\2011638006_odev2.c (99%)	270
c:\xampp\scenario2\2011638002_odev2.c (99%)	c:\xampp\scenario2\2011638005_odev2.c (99%)	270
c:\xampp\scenario2\2011638002_odev2.c (99%)	c:\xampp\scenario2\2011638004_odev2.c (99%)	270
c:\xampp\scenario2\2011638002_odev2.c (99%)	c:\xampp\scenario2\2011638003_odev2.c (99%)	270
c:\xampp\scenario2\2011638001_odev2.c (99%)	c:\xampp\scenario2\2011638010_odev2.c (99%)	270
c:\xampp\scenario2\2011638001_odev2.c (99%)	c:\xampp\scenario2\2011638009_odev2.c (99%)	270
c:\xampp\scenario2\2011638001_odev2.c (99%)	c:\xampp\scenario2\2011638008_odev2.c (99%)	270
c:\xampp\scenario2\2011638001_odev2.c (99%)	c:\xampp\scenario2\2011638007_odev2.c (99%)	270
c:\xampp\scenario2\2011638001_odev2.c (99%)	c:\xampp\scenario2\2011638006_odev2.c (99%)	270
c:\xampp\scenario2\2011638001_odev2.c (99%)	c:\xampp\scenario2\2011638005_odev2.c (99%)	270
c:\xampp\scenario2\2011638001_odev2.c (99%)	c:\xampp\scenario2\2011638004_odev2.c (99%)	270
c:\xampp\scenario2\2011638001_odev2.c (99%)	c:\xampp\scenario2\2011638003_odev2.c (99%)	270
c:\xampp\scenario2\2011638001_odev2.c (99%)	c:\xampp\scenario2\2011638002_odev2.c (99%)	270

Any errors encountered during this query are listed below.

Figure 4.14. Moss Results For Scenario 2

To view the detailed result of similarities, user can click the name of the assignment, which the user want to compare it with the others. The detailed result page is shown in Figure 4.15.

Figure 4.15. Moss Detailed Results For Scenario 2

Table 4.6. Moss Results as a Table For Scenario 2

MOSS	File1	File2	File3	File4	File5	File6	File7	File8	File9	File10
File1	-	99 %	99 %	99 %	99 %	99 %	99 %	99 %	99 %	99 %
File2	-	-	99 %	99 %	99 %	99 %	99 %	99 %	99 %	99 %
File3	-	-	-	99 %	99 %	99 %	99 %	99 %	99 %	99 %
File4	-	-	-	-	99 %	99 %	99 %	99 %	99 %	99 %
File5	-	-	-	-	-	99 %	99 %	99 %	99 %	99 %
File6	-	-	-	-	-	-	99 %	99 %	99 %	99 %
File7	-	-	-	-	-	-	-	99 %	99 %	99 %
File8	-	-	-	-	-	-	-	-	99 %	99 %
File9	-	-	-	-	-	-	-	-	-	99 %
File10	-	-	-	-	-	-	-	-	-	-

Table 4.6. includes the summary of the results given in Figure 4.14. In this scenario, the expected result is that the similarity rates of all assignments with each other are 100%. But the observed similarity rates are 99%. 270 lines of 285 lines are matched in every file with each other. Moss takes 6 seconds to compare the assignments in this scenario.

The assignments are then compared with JPlag. The result page is shown in Figure 4.16 and Figure 4.17. Figure 4.16 shows that the name of the assignment, the file names, the number of submissions, the minimum match length, the programming language of the assignment, the suffixes, the date, the threshold and the directory. It also shows a histogram for similarity score distribution. Figure 4.17 shows the average similarity.

Figure 4.16. JPlag Results For Scenario 2

Figure 4.17. JPlag Average Similarity Results For Scenario 2

To view the detailed result of similarities, user can click the name of the assignment, which the user want to compare it with the others. The detailed result page is shown in Figure 4.18.

Figure 4.18. JPlag Detailed Result For Scenario 2

In Table 4.7, File 1 represents the assignment of the first user. File 10 represents the assignment of the last user. This table is a comprehensible presentation of the Figure 4.17.

The expected result is that the similarity rates of all assignments with each other are 100% and the observed similarity rates are also 100%. As shown in Table 4.10 273 lines of 285 lines are matched in every file with each other and the header line is removed from the comparison. That's why all lines are matched. The process of JPlag takes 20 seconds.

Table 4.7. JPlag Results as a Table For Scenario 2

JPlag	**File1**	**File2**	**File3**	**File4**	**File5**	**File6**	**File7**	**File8**	**File9**	**File10**
File1	-	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
File2	-	-	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
File3	-	-	-	100 %	100 %	100 %	100 %	100 %	100 %	100 %
File4	-	-	-	-	100 %	100 %	100 %	100 %	100 %	100 %
File5	-	-	-	-	-	100 %	100 %	100 %	100 %	100 %
File6	-	-	-	-	-	-	100 %	100 %	100 %	100 %
File7	-	-	-	-	-	-	-	100 %	100 %	100 %
File8	-	-	-	-	-	-	-	-	100 %	100 %
File9	-	-	-	-	-	-	-	-	-	100 %
File10	-	-	-	-	-	-	-	-	-	-

The result page is shown in Figure 4.19 and Figure 4.20. Figure 4.19 shows the name of the assignment, the file names, the number of submissions, the minimum match length, the programming language of the assignment, the suffixes and the date of the comparison. Figure 4.20. shows the match score of the submitted assignments.

Finally, the assignments are compared with our hybrid algorithm which runs under Moodle. Table 4.8 summarizes the results of our hybrid algorithm. As shown in Table 4.8, File 1 represents the assignment of the first user. File 10 represents the assignment of the last user. This table is a comprehensible presentation of the Figure 4.20. The expected result is that the similarity rates of

all assignments with each other are 100% and the observed similarity rates are 100%. So, 273 lines of 285 lines are matched in every file with each other.

Table 4.8. The Hybrid Algorithm Results As a Table For Scenario 2

Hybrid	File1	File2	File3	File4	File5	File6	File7	File8	File9	File10
File1	-	100 %	100%	100%	100%	100%	100%	100%	100%	100 %
File2	-	-	100%	100%	100%	100%	100%	100%	100%	100 %
File3	-	-	-	100%	100%	100%	100%	100%	100%	100 %
File4	-	-	-	-	100%	100%	100%	100%	100%	100 %
File5	-	-	-	-	-	100%	100%	100%	100%	100 %
File6	-	-	-	-	-	-	100%	100%	100%	100 %
File7	-	-	-	-	-	-	-	100%	100%	100 %
File8	-	-	-	-	-	-	-	-	100%	100 %
File9	-	-	-	-	-	-	-	-	-	100 %
File10	-	-	-	-	-	-	-	-	-	-

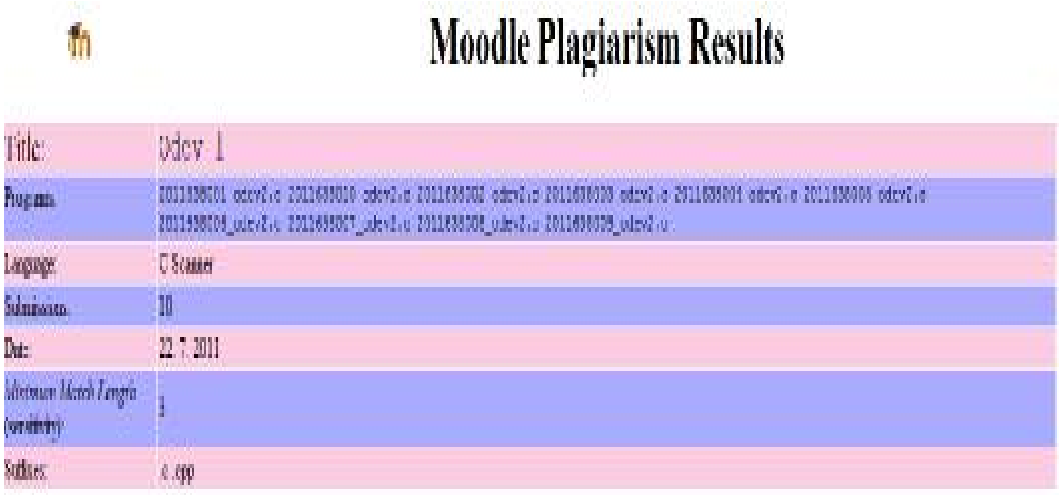


Figure 4.19. The Result Page of This Hybrid Algorithm For Scenario 2

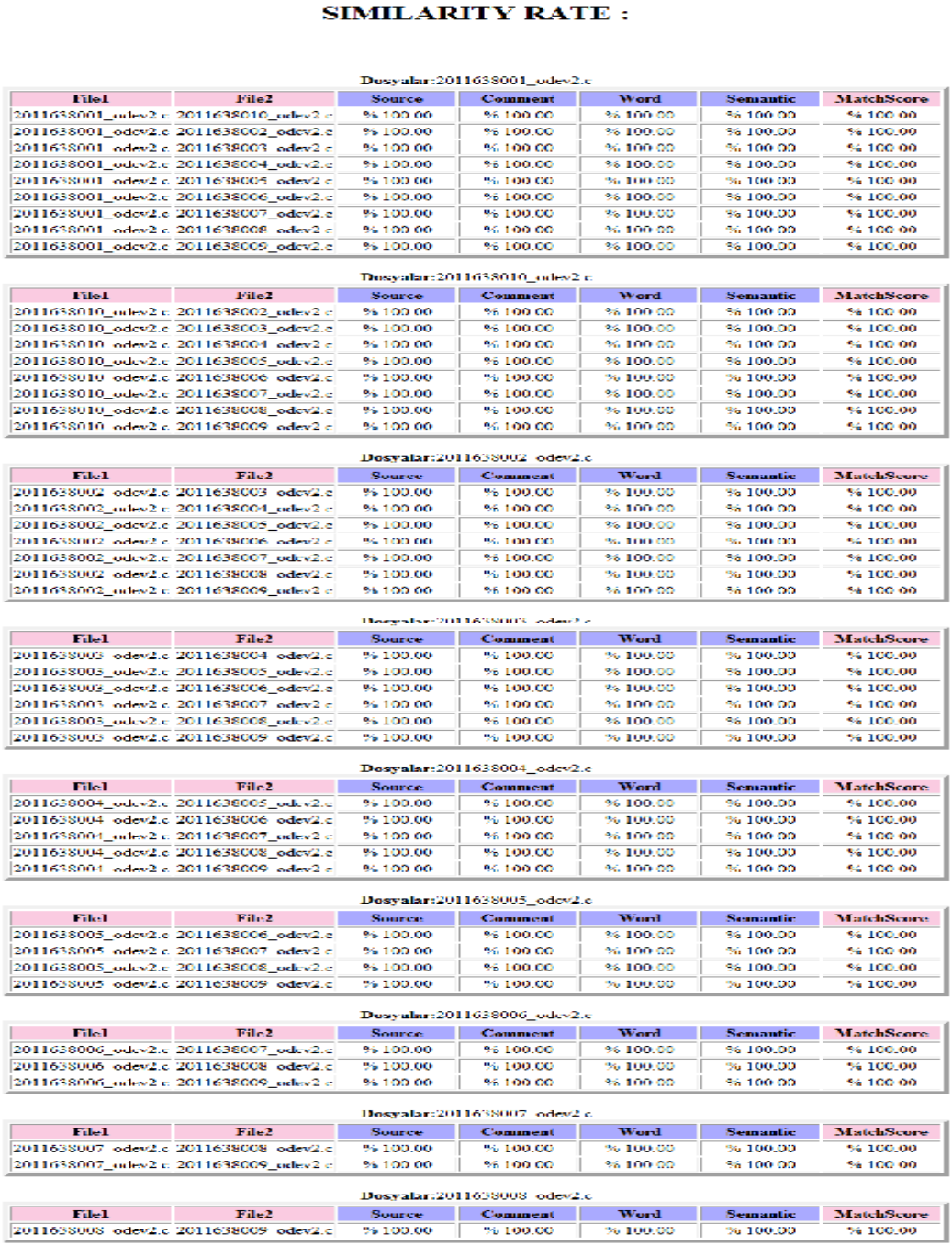
SIMILARITY RATE :

File1	File2	MatchScore
2011638010_odev2.c	2011638006_odev2.c	% 100.00
2011638002_odev2.c	2011638004_odev2.c	% 100.00
2011638002_odev2.c	2011638009_odev2.c	% 100.00
2011638003_odev2.c	2011638008_odev2.c	% 100.00
2011638001_odev2.c	2011638002_odev2.c	% 100.00
2011638004_odev2.c	2011638008_odev2.c	% 100.00
2011638001_odev2.c	2011638007_odev2.c	% 100.00
2011638005_odev2.c	2011638009_odev2.c	% 100.00
2011638010_odev2.c	2011638004_odev2.c	% 100.00
2011638007_odev2.c	2011638009_odev2.c	% 100.00
2011638010_odev2.c	2011638009_odev2.c	% 100.00
2011638002_odev2.c	2011638007_odev2.c	% 100.00
2011638003_odev2.c	2011638006_odev2.c	% 100.00
2011638004_odev2.c	2011638006_odev2.c	% 100.00
2011638001_odev2.c	2011638005_odev2.c	% 100.00
2011638005_odev2.c	2011638007_odev2.c	% 100.00
2011638010_odev2.c	2011638002_odev2.c	% 100.00
2011638006_odev2.c	2011638009_odev2.c	% 100.00
2011638010_odev2.c	2011638007_odev2.c	% 100.00
2011638002_odev2.c	2011638005_odev2.c	% 100.00
2011638003_odev2.c	2011638004_odev2.c	% 100.00
2011638003_odev2.c	2011638009_odev2.c	% 100.00
2011638001_odev2.c	2011638003_odev2.c	% 100.00
2011638004_odev2.c	2011638009_odev2.c	% 100.00
2011638001_odev2.c	2011638008_odev2.c	% 100.00
2011638006_odev2.c	2011638007_odev2.c	% 100.00
2011638010_odev2.c	2011638005_odev2.c	% 100.00
2011638008_odev2.c	2011638009_odev2.c	% 100.00
2011638002_odev2.c	2011638003_odev2.c	% 100.00
2011638002_odev2.c	2011638008_odev2.c	% 100.00
2011638003_odev2.c	2011638007_odev2.c	% 100.00
2011638001_odev2.c	2011638010_odev2.c	% 100.00
2011638004_odev2.c	2011638007_odev2.c	% 100.00
2011638001_odev2.c	2011638006_odev2.c	% 100.00
2011638005_odev2.c	2011638008_odev2.c	% 100.00
2011638010_odev2.c	2011638003_odev2.c	% 100.00
2011638007_odev2.c	2011638008_odev2.c	% 100.00
2011638010_odev2.c	2011638008_odev2.c	% 100.00
2011638002_odev2.c	2011638006_odev2.c	% 100.00
2011638003_odev2.c	2011638005_odev2.c	% 100.00
2011638004_odev2.c	2011638005_odev2.c	% 100.00
2011638001_odev2.c	2011638004_odev2.c	% 100.00
2011638005_odev2.c	2011638006_odev2.c	% 100.00
2011638001_odev2.c	2011638009_odev2.c	% 100.00
2011638006_odev2.c	2011638008_odev2.c	% 100.00

Detayli Göster

Figure 4.20. The Result of This Hybrid Algorithm For Scenario 2

The detailed result page that our hybrid algorithm produces is shown in Figure 4.21. It shows that the assignments are grouped and compared with each other. Source code similarity rate, comment line similarity rate, word matching similarity rate, semantic sequence similarity rate and match score which are all 100% are showed in Figure 4.21.



SIMILARITY RATE :

Dosyalar:2011638001_odev2.c

File1	File2	Source	Comment	Word	Semantic	MatchScore
2011638001_odev2.c	2011638010_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638001_odev2.c	2011638002_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638001_odev2.c	2011638003_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638001_odev2.c	2011638004_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638001_odev2.c	2011638005_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638001_odev2.c	2011638006_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638001_odev2.c	2011638007_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638001_odev2.c	2011638008_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638001_odev2.c	2011638009_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00

Dosyalar:2011638010_odev2.c

File1	File2	Source	Comment	Word	Semantic	MatchScore
2011638010_odev2.c	2011638002_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638010_odev2.c	2011638003_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638010_odev2.c	2011638004_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638010_odev2.c	2011638005_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638010_odev2.c	2011638006_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638010_odev2.c	2011638007_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638010_odev2.c	2011638008_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638010_odev2.c	2011638009_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00

Dosyalar:2011638002_odev2.c

File1	File2	Source	Comment	Word	Semantic	MatchScore
2011638002_odev2.c	2011638003_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638002_odev2.c	2011638004_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638002_odev2.c	2011638005_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638002_odev2.c	2011638006_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638002_odev2.c	2011638007_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638002_odev2.c	2011638008_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638002_odev2.c	2011638009_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00

Dosyalar:2011638003_odev2.c

File1	File2	Source	Comment	Word	Semantic	MatchScore
2011638003_odev2.c	2011638004_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638003_odev2.c	2011638005_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638003_odev2.c	2011638006_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638003_odev2.c	2011638007_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638003_odev2.c	2011638008_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638003_odev2.c	2011638009_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00

Dosyalar:2011638004_odev2.c

File1	File2	Source	Comment	Word	Semantic	MatchScore
2011638004_odev2.c	2011638005_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638004_odev2.c	2011638006_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638004_odev2.c	2011638007_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638004_odev2.c	2011638008_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638004_odev2.c	2011638009_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00

Dosyalar:2011638005_odev2.c

File1	File2	Source	Comment	Word	Semantic	MatchScore
2011638005_odev2.c	2011638006_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638005_odev2.c	2011638007_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638005_odev2.c	2011638008_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638005_odev2.c	2011638009_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00

Dosyalar:2011638006_odev2.c

File1	File2	Source	Comment	Word	Semantic	MatchScore
2011638006_odev2.c	2011638007_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638006_odev2.c	2011638008_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638006_odev2.c	2011638009_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00

Dosyalar:2011638007_odev2.c

File1	File2	Source	Comment	Word	Semantic	MatchScore
2011638007_odev2.c	2011638008_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00
2011638007_odev2.c	2011638009_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00

Dosyalar:2011638008_odev2.c

File1	File2	Source	Comment	Word	Semantic	MatchScore
2011638008_odev2.c	2011638009_odev2.c	% 100.00	% 100.00	% 100.00	% 100.00	% 100.00

Figure 4.21. The Detailed Result of This Hybrid Algorithm For Scenario 2

The numbers of matched tokens computed by our algorithm for every file are calculated and shown in Figure 4.22. and written to the database.

NUMBERS of MATCHED TOKENS:

File1	File2	SourceToken	CommentToken	WordToken	MaxSemanticToken
2011638010_odev2.c	2011638003_odev2.c	1227	15	47	235
2011638010_odev2.c	2011638009_odev2.c	1227	15	47	235
2011638010_odev2.c	2011638008_odev2.c	1227	15	47	235
2011638010_odev2.c	2011638007_odev2.c	1227	15	47	235
2011638010_odev2.c	2011638006_odev2.c	1227	15	47	235
2011638010_odev2.c	2011638005_odev2.c	1227	15	47	235
2011638010_odev2.c	2011638004_odev2.c	1227	15	47	235
2011638010_odev2.c	2011638002_odev2.c	1227	15	47	235
2011638008_odev2.c	2011638009_odev2.c	1227	15	47	235
2011638007_odev2.c	2011638008_odev2.c	1227	15	47	235
2011638007_odev2.c	2011638009_odev2.c	1227	15	47	235
2011638006_odev2.c	2011638008_odev2.c	1227	15	47	235
2011638006_odev2.c	2011638007_odev2.c	1227	15	47	235
2011638006_odev2.c	2011638009_odev2.c	1227	15	47	235
2011638005_odev2.c	2011638008_odev2.c	1227	15	47	235
2011638005_odev2.c	2011638006_odev2.c	1227	15	47	235
2011638005_odev2.c	2011638007_odev2.c	1227	15	47	235
2011638005_odev2.c	2011638009_odev2.c	1227	15	47	235
2011638004_odev2.c	2011638009_odev2.c	1227	15	47	235
2011638004_odev2.c	2011638005_odev2.c	1227	15	47	235
2011638004_odev2.c	2011638006_odev2.c	1227	15	47	235
2011638004_odev2.c	2011638007_odev2.c	1227	15	47	235
2011638004_odev2.c	2011638008_odev2.c	1227	15	47	235
2011638003_odev2.c	2011638007_odev2.c	1227	15	47	235
2011638003_odev2.c	2011638004_odev2.c	1227	15	47	235
2011638003_odev2.c	2011638005_odev2.c	1227	15	47	235
2011638003_odev2.c	2011638006_odev2.c	1227	15	47	235
2011638003_odev2.c	2011638009_odev2.c	1227	15	47	235
2011638003_odev2.c	2011638008_odev2.c	1227	15	47	235
2011638002_odev2.c	2011638003_odev2.c	1227	15	47	235
2011638002_odev2.c	2011638004_odev2.c	1227	15	47	235
2011638002_odev2.c	2011638005_odev2.c	1227	15	47	235
2011638002_odev2.c	2011638006_odev2.c	1227	15	47	235
2011638002_odev2.c	2011638007_odev2.c	1227	15	47	235
2011638002_odev2.c	2011638008_odev2.c	1227	15	47	235
2011638002_odev2.c	2011638009_odev2.c	1227	15	47	235
2011638001_odev2.c	2011638003_odev2.c	1227	15	47	235
2011638001_odev2.c	2011638004_odev2.c	1227	15	47	235
2011638001_odev2.c	2011638005_odev2.c	1227	15	47	235
2011638001_odev2.c	2011638007_odev2.c	1227	15	47	235
2011638001_odev2.c	2011638006_odev2.c	1227	15	47	235
2011638001_odev2.c	2011638008_odev2.c	1227	15	47	235
2011638001_odev2.c	2011638009_odev2.c	1227	15	47	235
2011638001_odev2.c	2011638010_odev2.c	1227	15	47	235
2011638001_odev2.c	2011638002_odev2.c	1227	15	47	235

Figure 4.22. Numbers of Matched Tokens For Scenario 2

The run time of our hybrid algorithm takes 111 seconds for this scenario shown as in Figure 4.23. Source Code Matching, Comment Line Matching, Semantic Sequence Matching, Word Matching Algorithms and Tokenization processes take 1 second. The Match Score process takes 1 second. So total process time is 113 seconds.

PROCESS TIME:

SourceTime	CommentTime	WordTime	SemanticTime	TokenTime	MatchScoreTime	TotalTime
111 second(s)	0 second(s)	0 second(s)	1 second(s)	0 second(s)	1 second(s)	113 second(s)

Figure 4.23. Process Time of This Hybrid Algorithm For Scenario 2

JPlag, Moss and our Hybrid Algorithm are compared with respect to similarity rate in Table 4.9. The random three matches are selected from the Scenario 2 assignments. As shown in the table, our hybrid algorithm and JPlag are more successful than Moss.

Table 4.9. Comparison of the Similarity Rate Computed By Moss JPlag and Hybrid For Scenario 2

FILE 1	**FILE 2**	**MOSS**	**JPLAG**	**HYBRID**
2011638001_odev2.c	2011638006_odev2.c	99 %	100 %	100 %
2011638003_odev2.c	2011638008_odev2.c	99 %	100 %	100 %
2011638005_odev2.c	2011638007_odev2.c	99 %	100 %	100 %

All the algorithms (i.e., Moss, JPlag, and Hybrid) are compared with respect to processing time for the first scenario and results are presented in Table 4.10. As shown in the table, for this scenario our hybrid algorithm is the slowest.

Table 4.10. Comparison of the Processing Time For Scenario 2

MOSS	**JPLAG**	**HYBRID**
6 seconds	20 seconds	113 seconds

4.3. The Third Scenario

The third scenario consist of 10 long source codes which are submitted by different students for the same assignment. These files are compared with each other using Moss, JPlag and The Hybrid Algorithm. Source files have between 84 and 601 lines and they do not have comment lines.

The assignments are compared with Moss firstly. The process of Moss takes 7 seconds. The command line for running Moss is shown in Figure 4.24. The result page is shown in Figure 4.25.

```
perl Moss.pl -l c c:\xampp\scenario3\odev4_2008638001.c
c:\xampp\scenario3\odev4_2008638012.c c:\xampp\scenario3\odev4_2009638019.c
c:\xampp\scenario3\odev4_2009638005.cpp c:\xampp\scenario3\odev4_2009638027.c
c:\xampp\scenario3\odev4_2009638048.c c:\xampp\scenario3\bagliliste.cpp
c:\xampp\scenario3\odev4_2009639039.c c:\xampp\scenario3\odev4_2009638401.c
c:\xampp\scenario3\odev4_2009638506.c
```

Figure 4.24. Moss Script For Scenario 3

Moss Results

Sun Jul 24 03:20:56 PDT 2011

Options -l c -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
c:\xampp\scenario3\odev4_2008638001.c (84%)	c:\xampp\scenario3\odev4_2009638019.c (83%)	156
c:\xampp\scenario3\odev4_2009638401.c (79%)	c:\xampp\scenario3\odev4_2009638506.c (40%)	59
c:\xampp\scenario3\odev4_2009638019.c (10%)	c:\xampp\scenario3\odev4_2009638506.c (18%)	26
c:\xampp\scenario3\odev4_2008638012.c (3%)	c:\xampp\scenario3\odev4_2009638005.cpp (8%)	28
c:\xampp\scenario3\odev4_2009638019.c (5%)	c:\xampp\scenario3\odev4_2009638401.c (18%)	13
c:\xampp\scenario3\odev4_2008638012.c (2%)	c:\xampp\scenario3\odev4_2009638019.c (4%)	6
c:\xampp\scenario3\odev4_2008638001.c (4%)	c:\xampp\scenario3\odev4_2008638012.c (2%)	6
c:\xampp\scenario3\odev4_2008638012.c (1%)	c:\xampp\scenario3\odev4_2009638506.c (6%)	10
c:\xampp\scenario3\odev4_2008638012.c (1%)	c:\xampp\scenario3\odev4_2009638401.c (13%)	10

Any errors encountered during this query are listed below.

Figure 4.25. Moss Results For Scenario 3

The assignments are then compared with JPlag. The process of JPlag takes 23 seconds. The result page is shown in Figure 4.26. It shows the average similarity.

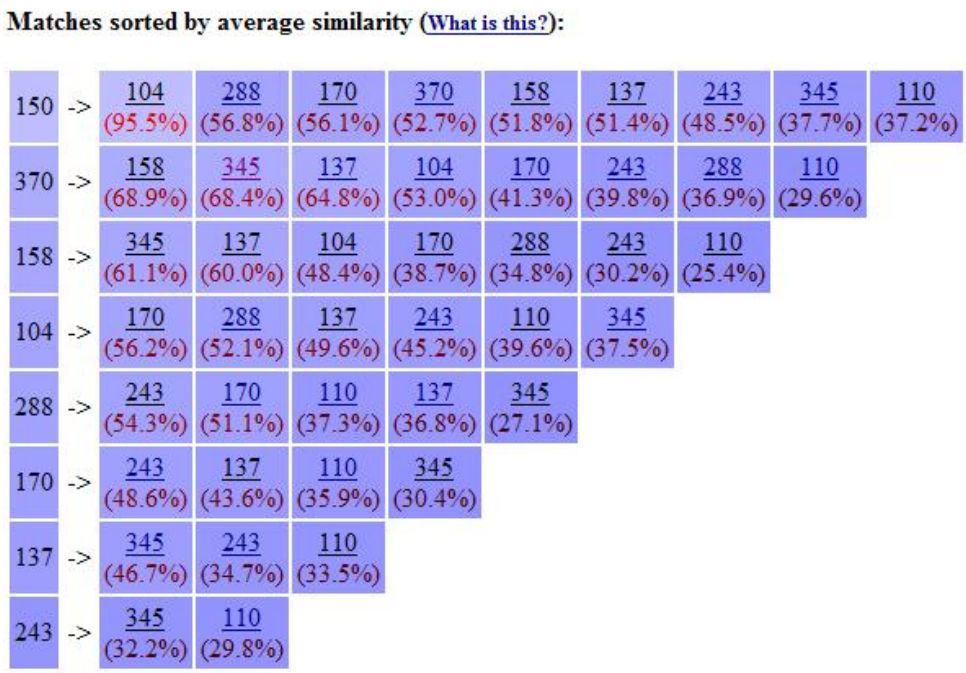
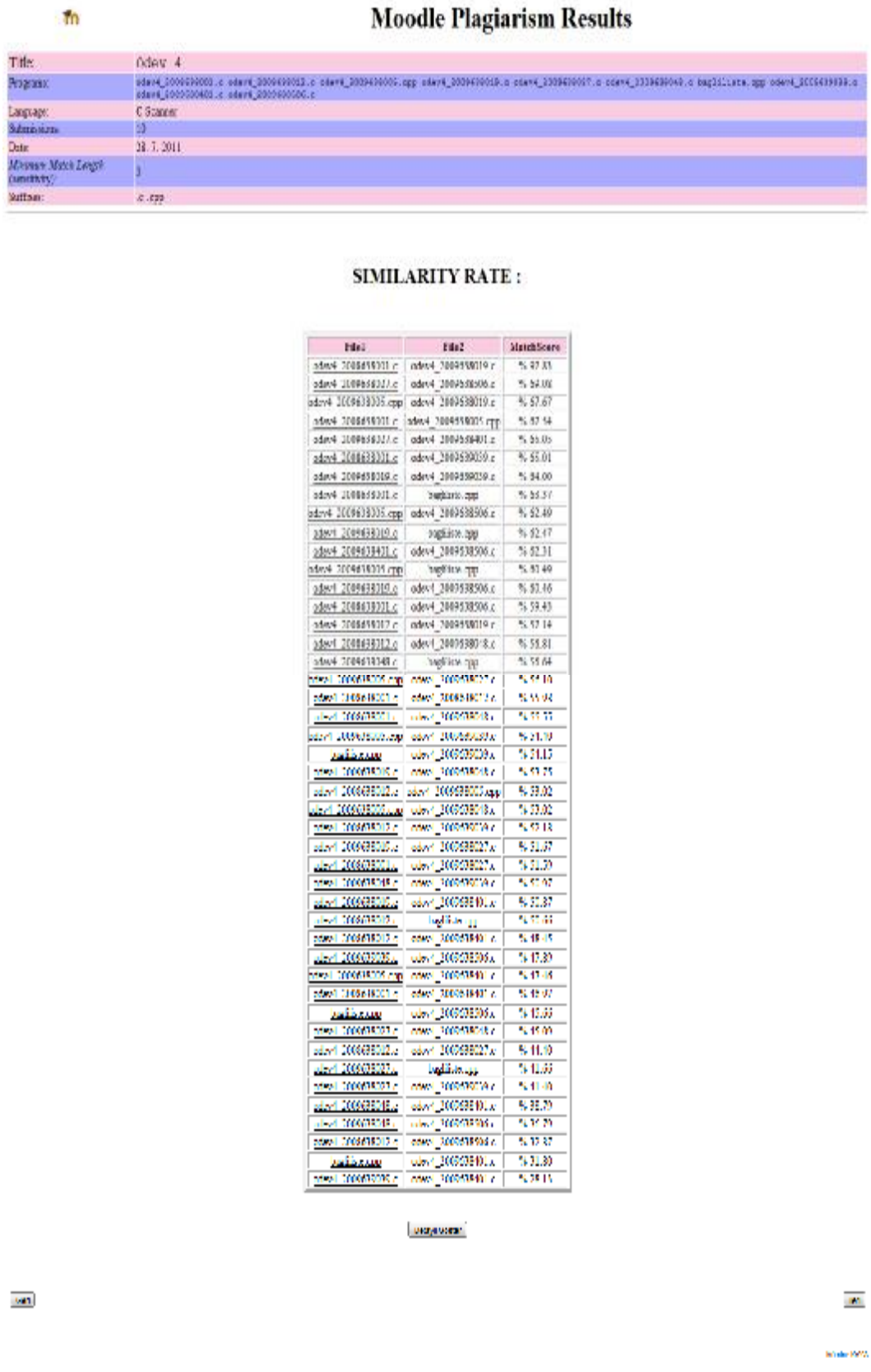


Figure 4.26. JPlag Average Similarity Results For Scenario 3

Finally, the assignments are compared with our hybrid algorithm. The result page is shown in Figure 4.27. It shows that the name of the assignment, the file names, the number of submissions, the minimum match length, the programming language, the suffixes and the date.

The detailed result page is shown in Figure 4.28. It shows that the assignments are grouped and compared with each other. Source code similarity rate, comment line similarity rate, word matching similarity rate, semantic sequence similarity rate and match score are showed in Figure 4.28.



Moodle Plagiarism Results

Title:	Ödev 4
Programs:	ödev4_2009690003.c ödev4_2009690013.c ödev4_2009690005.cpp ödev4_2009690018.c ödev4_2009690007.c ödev4_2009690049.c bag2Liste.cpp ödev4_2009690038.c ödev4_2009690403.c ödev4_2009690006.c
Language:	C Source
Submissions:	10
Date:	28. 7. 2011
Minimum Match Length (sensitivity):	3
Suffixes:	.c .cpp

SIMILARITY RATE :

File1	File2	MatchScore
ödev4_2009690311.c	ödev4_2009690019.c	% 97.83
ödev4_2009690311.c	ödev4_2009690506.c	% 94.08
ödev4_2009690335.cpp	ödev4_2009690019.c	% 97.67
ödev4_2009690311.c	ödev4_2009690005.cpp	% 97.54
ödev4_2009690311.c	ödev4_2009690401.c	% 96.05
ödev4_2009690311.c	ödev4_2009690059.c	% 95.01
ödev4_2009690019.c	ödev4_2009690059.c	% 94.00
ödev4_2009690311.c	bagliste.cpp	% 93.57
ödev4_2009690335.cpp	ödev4_2009690506.c	% 92.49
ödev4_2009690019.c	bagliste.cpp	% 92.47
ödev4_2009690341.c	ödev4_2009690506.c	% 92.31
ödev4_2009690335.cpp	bagliste.cpp	% 91.49
ödev4_2009690019.c	ödev4_2009690506.c	% 91.16
ödev4_2009690311.c	ödev4_2009690506.c	% 90.43
ödev4_2009690317.c	ödev4_2009690019.c	% 87.14
ödev4_2009690312.c	ödev4_2009690018.c	% 86.81
ödev4_2009690349.c	bagliste.cpp	% 86.64
ödev4_2009690506.cpp	ödev4_2009690027.c	% 84.10
ödev4_2009690027.c	ödev4_2009690027.c	% 83.04
ödev4_2009690027.c	ödev4_2009690018.c	% 82.75
bagliste.cpp	ödev4_2009690018.c	% 81.13
ödev4_2009690506.c	ödev4_2009690018.c	% 80.75
ödev4_2009690312.c	ödev4_2009690027.cpp	% 80.00
ödev4_2009690335.cpp	ödev4_2009690018.c	% 79.00
ödev4_2009690027.c	ödev4_2009690018.c	% 78.13
ödev4_2009690312.c	ödev4_2009690027.c	% 78.37
ödev4_2009690027.c	ödev4_2009690027.c	% 78.29
ödev4_2009690506.c	ödev4_2009690027.c	% 77.92
ödev4_2009690312.c	ödev4_2009690018.c	% 77.87
ödev4_2009690027.c	bagliste.cpp	% 77.83
ödev4_2009690312.c	ödev4_2009690403.c	% 76.15
ödev4_2009690335.cpp	ödev4_2009690018.c	% 75.89
ödev4_2009690506.cpp	ödev4_2009690403.c	% 75.16
ödev4_2009690311.c	ödev4_2009690403.c	% 75.07
bagliste.cpp	ödev4_2009690018.c	% 75.00
ödev4_2009690506.c	ödev4_2009690506.c	% 74.89
ödev4_2009690312.c	ödev4_2009690027.c	% 74.10
ödev4_2009690027.c	bagliste.cpp	% 73.83
ödev4_2009690506.c	ödev4_2009690403.c	% 73.49
ödev4_2009690018.c	ödev4_2009690018.c	% 72.79
ödev4_2009690506.c	ödev4_2009690506.c	% 72.29
ödev4_2009690312.c	ödev4_2009690506.c	% 72.07
bagliste.cpp	ödev4_2009690018.c	% 71.80
ödev4_2009690335.c	ödev4_2009690403.c	% 71.13

Figure 4.27. The Result of This Hybrid Algorithm For Scenario 3

SIMILARITY RATE :

Dosyalar:ceken4_2008638001.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev1_2008638001.c	odev1_2008638010.c	% 97.83	% 0.00	% 93.91	% 6.00	% 97.83	101	230	% 97.83
ceken4_2008638001.c	ceken4_2008638001.cpp	% 69.46	% 0.00	% 51.92	% 0.61	% 64.91	104	137	% 67.54
ceken4_2008638001.c	ceken4_2008638035.c	% 68.90	% 0.00	% 36.36	% 4.47	% 61.29	104	288	% 63.01
ceken4_2008638001.c	bagliste.cpp	% 68.01	% 0.00	% 22.22	% 3.11	% 58.59	104	249	% 60.37
odev1_2008638001.c	odev1_2008638006.c	% 62.72	% 0.00	% 30.30	% 0.30	% 56.77	101	330	% 58.70
ceken4_2008638001.c	ceken4_2008638012.c	% 57.90	% 0.00	% 43.43	% 2.79	% 54.07	104	110	% 55.98
ceken4_2008638001.c	ceken4_2008638048.c	% 58.44	% 0.00	% 30.43	% 2.79	% 53.31	104	170	% 55.55
odev1_2008638001.c	odev1_2008638027.c	% 53.00	% 0.00	% 40.00	% 2.50	% 49.65	101	158	% 51.50
ceken4_2008638001.c	ceken4_2008638401.c	% 50.86	% 0.00	% 30.00	% 5.00	% 46.94	104	343	% 48.97

Dosyalar:odev1_2008638012.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
ceken4_2008638012.c	ceken4_2008638010.c	% 58.26	% 0.00	% 48.48	% 2.75	% 55.38	110	130	% 57.14
ceken4_2008638012.c	ceken4_2008638048.c	% 57.97	% 0.00	% 47.83	% 0.34	% 53.01	110	170	% 55.81
ceken4_2008638012.c	ceken4_2008638001.cpp	% 54.87	% 0.00	% 37.70	% 10.91	% 50.79	110	137	% 53.00
odev1_2008638012.c	odev1_2008638035.c	% 56.70	% 0.00	% 13.95	% 2.70	% 47.80	110	288	% 52.13
ceken4_2008638012.c	bagliste.cpp	% 53.85	% 0.00	% 22.22	% 3.11	% 47.21	110	249	% 50.61
odev1_2008638012.c	odev1_2008638002.c	% 18.01	% 0.00	% 61.11	% 8.33	% 28.13	110	342	% 38.76
odev1_2008638012.c	odev1_2008638021.c	% 31.31	% 0.00	% 30.00	% 1.25	% 31.05	110	158	% 31.70
ceken4_2008638012.c	ceken4_2008638006.c	% 34.36	% 0.00	% 20.00	% 1.50	% 31.13	110	330	% 32.87

Dosyalar:odev1_2008638001.cpp

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev1_2008638001.cpp	odev1_2008638010.c	% 65.69	% 0.00	% 51.52	% 0.61	% 63.02	137	130	% 67.07
ceken4_2008638001.cpp	ceken4_2008638036.c	% 63.13	% 0.00	% 40.00	% 0.50	% 59.36	137	370	% 62.49
odev1_2008638001.cpp	bagliste.cpp	% 63.02	% 0.00	% 33.33	% 0.82	% 58.95	137	249	% 60.79
odev1_2008638001.cpp	odev1_2008638027.c	% 55.65	% 0.00	% 40.00	% 2.50	% 51.00	137	158	% 53.10
odev1_2008638001.cpp	odev1_2008638039.c	% 58.72	% 0.00	% 18.60	% 7.88	% 50.21	137	288	% 54.70
ceken4_2008638001.cpp	ceken4_2008638048.c	% 55.69	% 0.00	% 30.43	% 2.42	% 50.06	137	170	% 53.02
odev1_2008638001.cpp	odev1_2008638021.c	% 42.03	% 0.00	% 30.00	% 1.67	% 42.22	137	158	% 44.06

Dosyalar:ceken4_2008638010.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
ceken4_2008638010.c	ceken4_2008638035.c	% 67.18	% 0.00	% 36.36	% 4.40	% 60.39	130	288	% 64.00
odev1_2008638010.c	bagliste.cpp	% 67.01	% 0.00	% 22.22	% 3.11	% 57.05	130	249	% 62.17
ceken4_2008638010.c	ceken4_2008638006.c	% 63.38	% 0.00	% 33.33	% 4.50	% 56.97	130	330	% 60.46
odev1_2008638010.c	odev1_2008638018.c	% 55.97	% 0.00	% 37.08	% 2.20	% 51.20	130	170	% 53.05
ceken4_2008638010.c	ceken4_2008638027.c	% 52.74	% 0.00	% 43.33	% 1.25	% 50.00	130	158	% 51.67
odev1_2008638010.c	odev1_2008638002.c	% 39.93	% 0.00	% 35.56	% 1.67	% 41.95	130	342	% 50.87

Dosyalar:odev1_2008638027.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev1_2008638027.c	odev1_2008638006.c	% 53.52	% 0.00	% 30.00	% 1.25	% 47.32	158	330	% 49.08
ceken4_2008638027.c	ceken4_2008638401.c	% 51.37	% 0.00	% 36.67	% 1.67	% 47.09	158	343	% 48.90
odev1_2008638027.c	odev1_2008638018.c	% 38.70	% 0.00	% 17.85	% 2.50	% 39.58	158	170	% 43.09
ceken4_2008638027.c	bagliste.cpp	% 43.59	% 0.00	% 33.33	% 1.25	% 40.18	158	249	% 41.56
ceken4_2008638027.c	ceken4_2008638010.c	% 42.75	% 0.00	% 30.00	% 1.00	% 39.68	158	288	% 41.40

Dosyalar:odev1_2008638018.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev1_2008638018.c	bagliste.cpp	% 59.83	% 0.00	% 27.08	% 2.08	% 53.03	170	249	% 56.01
odev1_2008638018.c	odev1_2008638035.c	% 51.76	% 0.00	% 37.09	% 0.91	% 47.02	170	288	% 50.07
ceken4_2008638048.c	ceken4_2008638401.c	% 35.61	% 0.00	% 44.44	% 8.33	% 38.33	170	343	% 38.79
ceken4_2008638048.c	ceken4_2008638006.c	% 39.54	% 0.00	% 8.70	% 0.90	% 33.59	170	330	% 36.79

Dosyalar:bagliste.cpp

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
bagliste.cpp	odev1_2008638010.c	% 67.06	% 0.00	% 22.22	% 3.11	% 58.71	249	288	% 62.26
bagliste.cpp	odev1_2008638006.c	% 48.93	% 0.00	% 16.67	% 3.80	% 42.25	249	330	% 45.26
bagliste.cpp	ceken4_2008638401.c	% 52.92	% 0.00	% 22.22	% 6.67	% 50.42	249	343	% 51.50

Dosyalar:odev1_2008638039.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev1_2008638039.c	odev1_2008638006.c	% 50.72	% 0.00	% 28.57	% 1.60	% 45.42	288	330	% 47.89
odev1_2008638039.c	odev1_2008638002.c	% 29.46	% 0.00	% 16.67	% 1.67	% 26.81	288	342	% 28.23

Dosyalar:ceken4_2008638401.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
ceken4_2008638401.c	ceken4_2008638006.c	% 66.01	% 0.00	% 27.78	% 13.33	% 58.18	343	330	% 60.31

Figure 4.28. The Detailed Result of This Hybrid Algorithm For Scenario 3

The numbers of matched tokens for every file are calculated as shown in Figure 4.29. and written to the database.

NUMBERS of MATCHED TOKENS:

File1	File2	SourceToken	CommentToken	WordToken	MaxSemanticToken
odev4_2009639039.c	odev4_2009638506.c	423	0	10	4
odev4_2009639039.c	odev4_2009638401.c	214	0	3	1
odev4_2009638401.c	odev4_2009638506.c	240	0	5	8
odev4_2009638048.c	odev4_2009638506.c	390	0	2	1
odev4_2009638048.c	odev4_2009638401.c	220	0	8	5
odev4_2009638048.c	odev4_2009639039.c	734	0	4	1
odev4_2009638048.c	bagliliste.cpp	703	0	5	4
odev4_2009638027.c	odev4_2009638048.c	369	0	11	2
odev4_2009638027.c	bagliliste.cpp	279	0	6	1
odev4_2009638027.c	odev4_2009639039.c	351	0	9	4
odev4_2009638027.c	odev4_2009638401.c	208	0	12	1
odev4_2009638027.c	odev4_2009638506.c	336	0	9	1
odev4_2009638019.c	odev4_2009638027.c	347	0	13	1
odev4_2009638019.c	odev4_2009639039.c	702	0	12	8
odev4_2009638019.c	odev4_2009638048.c	659	0	8	4
odev4_2009638019.c	bagliliste.cpp	588	0	4	5
odev4_2009638019.c	odev4_2009638401.c	225	0	10	1
odev4_2009638019.c	odev4_2009638506.c	433	0	11	5
odev4_2009638005.cpp	odev4_2009638019.c	604	0	17	1
odev4_2009638005.cpp	odev4_2009638506.c	434	0	14	1
odev4_2009638005.cpp	odev4_2009638401.c	228	0	9	1
odev4_2009638005.cpp	odev4_2009639039.c	602	0	8	13
odev4_2009638005.cpp	bagliliste.cpp	549	0	6	1
odev4_2009638005.cpp	odev4_2009638048.c	647	0	7	4
odev4_2009638005.cpp	odev4_2009638027.c	371	0	13	4
odev4_2008638012.c	odev4_2009638005.cpp	790	0	23	18
odev4_2008638012.c	odev4_2009638019.c	848	0	16	5
odev4_2008638012.c	odev4_2009638027.c	386	0	15	1
odev4_2008638012.c	odev4_2009638048.c	1015	0	11	1
odev4_2008638012.c	bagliliste.cpp	782	0	4	5
odev4_2008638012.c	odev4_2009639039.c	914	0	6	5
odev4_2008638012.c	odev4_2009638401.c	215	0	11	5
odev4_2008638012.c	odev4_2009638506.c	431	0	7	2
odev4_2008638001.c	odev4_2009638005.cpp	598	0	17	1
odev4_2008638001.c	bagliliste.cpp	592	0	4	5
odev4_2008638001.c	odev4_2009638019.c	856	0	31	12
odev4_2008638001.c	odev4_2009638027.c	345	0	12	2
odev4_2008638001.c	odev4_2009638048.c	684	0	7	5
odev4_2008638001.c	odev4_2009639039.c	709	0	12	8
odev4_2008638001.c	odev4_2009638401.c	222	0	9	3
odev4_2008638001.c	odev4_2009638506.c	423	0	10	5
odev4_2008638001.c	odev4_2008638012.c	830	0	15	5
bagliliste.cpp	odev4_2009638401.c	184	0	4	4
bagliliste.cpp	odev4_2009638506.c	331	0	3	4
bagliliste.cpp	odev4_2009639039.c	601	0	4	5

Figure 4.29. Numbers of Matched Tokens For Scenario 3

The process of this hybrid algorithm takes 460 seconds shown in Figure 4.30. Source Code Matching Algorithm process is taken 457 seconds. Comment Line Matching, Semantic Sequence Matching, Word Matching Algorithms and Tokenization processes take 2 seconds. The Match Score process takes 1 second. So all algorithms and processes take 460 seconds.

PROCESS TIME:

SourceTime	CommentTime	WordTime	SemanticTime	TokenTime	MatchScoreTime	TotalTime
457 second(s)	0 second(s)	1 second(s)	1 second(s)	0 second(s)	1 second(s)	460 second(s)

Figure 4.30. Process Time of This Hybrid Algorithm For Scenario 3

JPlag, Moss and this Hybrid Algorithm are compared with respect to similarity rate in a Table 4.11. There are 45 matches in this scenario. The random ten matches are selected from the Scenario 3 assignments. As shown in the table, our hybrid algorithm is more successful than Moss and JPlag. Especially Moss cannot detect plagiarism on most assignments.

Table 4.11. The Comparison of the Similarity Rate Computed By Moss JPlag and Hybrid For Scenario 3

FILE1	FILE2	MOSS	JPLAG	HYBRID
odev4_2008638001.c	odev4_2008638012.c	15 %	39.6 %	**55.98 %**
odev4_2008638001.c	odev4_2009638005.cpp	11 %	49.6 %	**67.54 %**
odev4_2008638001.c	odev4_2009638019.c	84 %	95.5 %	**97.83 %**
odev4_2008638012.c	odev4_2009638019.c	12 %	37.2 %	**57.14 %**
odev4_2009638005.cpp	odev4_2009638027.c	0 %	**60.0 %**	56.10 %
odev4_2009638019.c	odev4_2009638048.c	0 %	**56.1 %**	53.75 %
odev4_2009638027.c	odev4_2009639039.c	0 %	34.8 %	**41.40 %**
odev4_2009638027.c	odev4_2009638506.c	0 %	68.9 %	**69.08 %**
odev4_2009638048.c	odev4_2009638401.c	0 %	30.4 %	**38.79 %**
odev4_2009638401.c	odev4_2009638506.c	59 %	**68.4 %**	62.31 %

The process time of Moss, JPlag and Hybrid Algorithm are compared in a Table 4.12. As shown in the table, our hybrid algorithm is the slowest.

Table 4.12. The Comparison of the Process Time For Scenario 3

MOSS	**JPLAG**	**HYBRID**
7 seconds	23 seconds	460 seconds

4.4. The Fourth Scenario

The fourth scenario consist of 25 long source codes which are submitted by different students of the same assignment. These files are compared with each other using Moss, JPlag and The Hybrid Algorithm. Source files have between 71 and 775 lines and they have comment lines.

The assignments are compared with Moss firstly. The process of Moss takes 30 seconds. The command line for running Moss is shown in Figure 4.31. The result page is shown in Figure 4.32.

```
perl Moss.pl -l c c:\xampp\scenario4\odev4_2008638001.c
c:\xampp\scenario3\odev4_2008638012.c c:\xampp\scenario3\odev4_2009638019.c
c:\xampp\scenario3\odev4_2009638005.cpp c:\xampp\scenario3\odev4_2009638027.c
c:\xampp\scenario3\odev4_2009638048.c c:\xampp\scenario3\bagliliste.cpp
c:\xampp\scenario3\odev4_2009639039.c c:\xampp\scenario3\odev4_2009638401.c
c:\xampp\scenario3\odev4_2009638506.c c:\xampp\scenario3\odev4_2008638001.c
c:\xampp\scenario3\odev4_2008638012.c c:\xampp\scenario3\odev4_2009638019.c
c:\xampp\scenario3\odev4_2009638005.cpp c:\xampp\scenario3\odev4_2009638027.c
c:\xampp\scenario3\odev4_2009638048.c c:\xampp\scenario3\bagliliste.cpp
c:\xampp\scenario3\odev4_2009639039.c c:\xampp\scenario3\odev4_2009638401.c
c:\xampp\scenario3\odev4_2009638506.c c:\xampp\scenario3\odev4_2008638001.c
c:\xampp\scenario3\odev4_2008638012.c c:\xampp\scenario3\odev4_2009638019.c
c:\xampp\scenario3\odev4_2009638005.cpp c:\xampp\scenario3\odev4_2009638027.c
```

Figure 4.31. Moss Script For Scenario 4

Moss Results

Sun Jul 24 03:20:56 PDT 2011

Options -l c -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
c:\xampp\scenario3\odev4_2008638001.c (84%)	c:\xampp\scenario3\odev4_2009638019.c (83%)	156
c:\xampp\scenario3\odev4_2009638401.c (79%)	c:\xampp\scenario3\odev4_2009638506.c (40%)	59
c:\xampp\scenario3\odev4_2009638019.c (10%)	c:\xampp\scenario3\odev4_2009638506.c (18%)	26
c:\xampp\scenario3\odev4_2008638012.c (3%)	c:\xampp\scenario3\odev4_2009638005.cpp (8%)	28
c:\xampp\scenario3\odev4_2009638019.c (5%)	c:\xampp\scenario3\odev4_2009638401.c (18%)	13
c:\xampp\scenario3\odev4_2008638012.c (2%)	c:\xampp\scenario3\odev4_2009638019.c (4%)	6
c:\xampp\scenario3\odev4_2008638001.c (4%)	c:\xampp\scenario3\odev4_2008638012.c (2%)	6
c:\xampp\scenario3\odev4_2008638012.c (1%)	c:\xampp\scenario3\odev4_2009638506.c (6%)	10
c:\xampp\scenario3\odev4_2008638012.c (1%)	c:\xampp\scenario3\odev4_2009638401.c (13%)	10

Any errors encountered during this query are listed below.

Figure 4.32. Moss Results For Scenario 4

The assignments are then compared with JPlag. The process of JPlag takes 70 seconds. The result page is shown in Figure 4.33. It shows the average similarity.

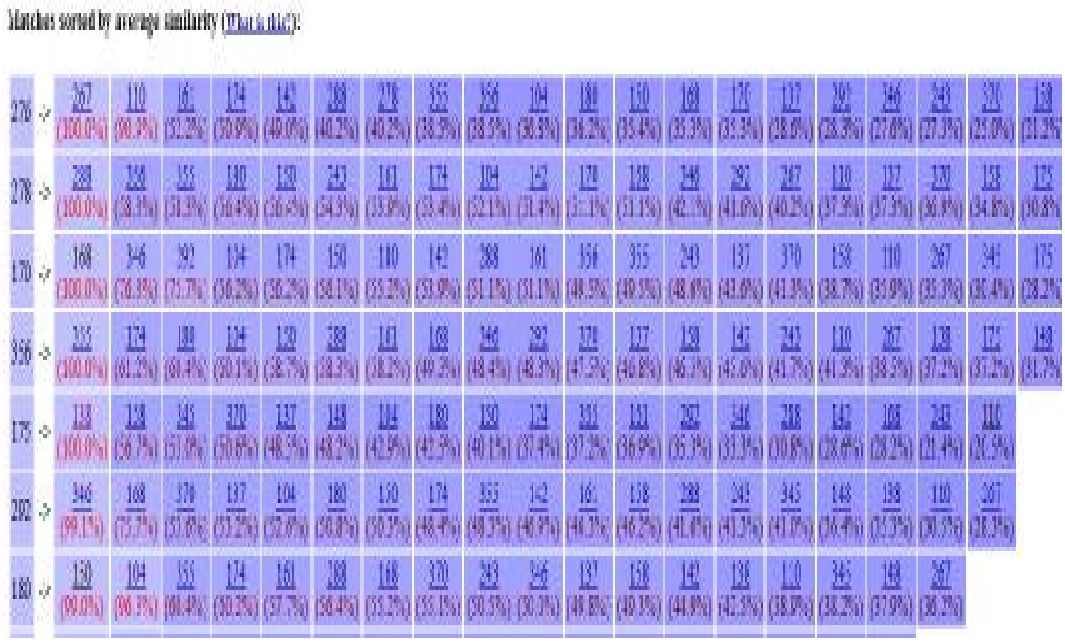


Figure 4.33. JPlag Average Similarity Results For Scenario 4

Finally, the assignments are compared with our hybrid algorithm. The result page is shown in Figure 4.34. It shows that the name of the assignment, the file names, the number of submissions, the minimum match length, the programming language, the suffixes and the date.

The detailed result page is shown in Figure 4.35. It shows that the assignments are grouped and compared with each other. Source code similarity rate, comment line similarity rate, word matching similarity rate, semantic sequence similarity rate and match score are showed in Figure 4.35.

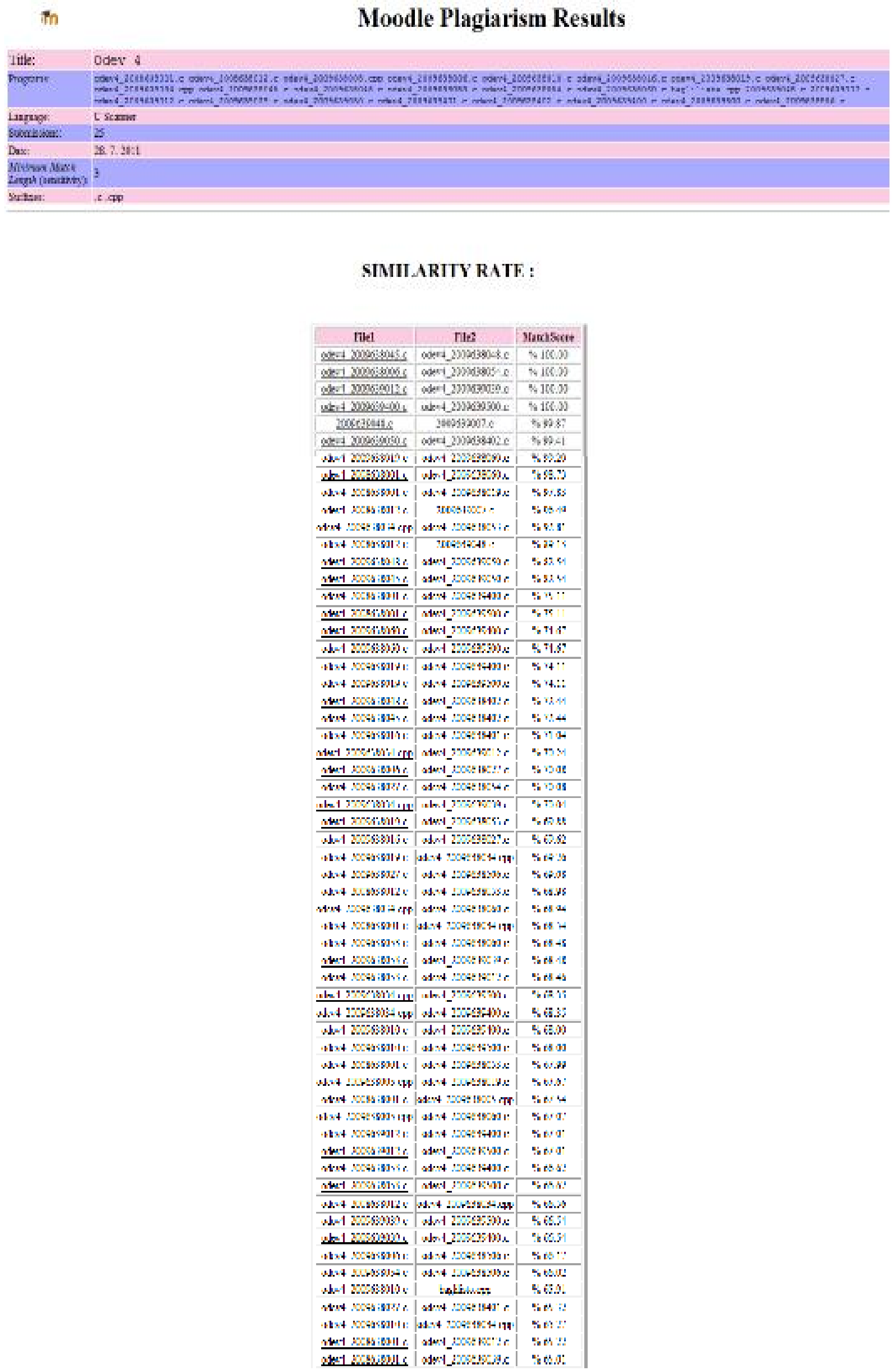


Figure 4.34. The Result of This Hybrid Algorithm For Scenario 4

SIMILARITY RATE :

Dosyalar:odev1_2008638001.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev1_2008638001.c	odev1_2009638060.c	% 98.73	% 0.00	% 100.00	% 54.29	% 99.52	% 104	% 180	% 98.73
odev1_2008638001.c	odev1_2009638019.c	% 97.83	% 0.00	% 95.91	% 6.70	% 95.25	% 104	% 150	% 97.83
odev1_2008638001.c	odev1_2009639100.c	% 78.35	% 0.00	% 45.45	% 2.79	% 71.12	% 104	% 255	% 75.11
odev1_2008638001.c	odev1_2009639500.c	% 78.35	% 0.00	% 45.45	% 2.79	% 71.12	% 104	% 256	% 75.11
odev1_2008638001.c	odev1_2009638034.cpp	% 70.92	% 0.00	% 48.48	% 5.59	% 65.36	% 104	% 161	% 68.54
odev4_2008638001.c	odev4_2009638053.c	% 69.98	% 0.00	% 51.52	% 5.59	% 65.35	% 104	% 174	% 67.99
odev4_2008638001.c	odev4_2009638005.cpp	% 69.49	% 0.00	% 51.52	% 0.61	% 64.91	% 104	% 137	% 67.54
odev4_2008638001.c	odev4_2009639012.c	% 68.21	% 0.00	% 39.39	% 5.03	% 61.77	% 104	% 278	% 65.22
odev1_2008638001.c	odev1_2009639020.c	% 68.30	% 0.00	% 36.36	% 4.47	% 61.29	% 104	% 288	% 65.01
odev4_2008638001.c	odev4_2009638006.c	% 67.15	% 0.00	% 43.75	% 0.94	% 61.65	% 104	% 138	% 64.68
odev4_2008638001.c	odev4_2009638094.c	% 67.15	% 0.00	% 39.39	% 0.94	% 60.88	% 104	% 173	% 64.20
odev1_2008638001.c	odev1_2009638010.c	% 65.96	% 0.00	% 41.67	% 0.57	% 60.32	% 104	% 142	% 63.41
odev4_2008638001.c	baglliste.cpp	% 68.01	% 0.00	% 22.22	% 3.11	% 58.53	% 104	% 243	% 63.37
odev4_2008638001.c	odev4_2009639050.c	% 64.14	% 0.00	% 31.82	% 3.02	% 57.10	% 104	% 293	% 61.00
odev1_2008638001.c	odev1_2009638402.c	% 63.88	% 0.00	% 31.82	% 3.03	% 56.93	% 104	% 345	% 60.58
odev4_2008638001.c	odev4_2009638306.c	% 62.76	% 0.00	% 30.30	% 4.50	% 55.77	% 104	% 370	% 59.43
odev1_2008638001.c	odev1_2008638012.c	% 57.30	% 0.00	% 45.45	% 2.79	% 54.07	% 104	% 110	% 55.58
odev1_2008638001.c	odev1_2009638078.c	% 58.44	% 0.00	% 30.43	% 2.79	% 52.31	% 104	% 170	% 55.55
odev1_2008638001.c	odev1_2009638076.c	% 58.44	% 0.00	% 30.43	% 2.79	% 52.31	% 104	% 168	% 55.55
odev4_2008638001.c	odev4_2009638027.c	% 53.00	% 0.00	% 40.00	% 2.50	% 49.65	% 104	% 158	% 51.59
odev1_2008638001.c	odev1_2009638401.c	% 39.89	% 0.00	% 50.00	% 5.00	% 40.94	% 104	% 345	% 47.47
odev1_2008638001.c	2009638007.c	% 47.76	% 0.00	% 45.45	% 2.79	% 45.12	% 104	% 275	% 47.10
odev4_2008638001.c	2009639045.c	% 47.61	% 0.00	% 42.42	% 2.79	% 45.76	% 104	% 267	% 46.97
odev1_2008638001.c	odev1_2009638016.c	% 42.89	% 0.00	% 45.45	% 2.86	% 42.52	% 104	% 148	% 44.81

Dosyalar:odev1_2008638012.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev1_2008638012.c	2009638007.c	% 85.96	% 0.00	% 100.00	% 5.35	% 88.77	% 110	% 275	% 86.40
odev1_2008638012.c	2009638045.c	% 85.91	% 0.00	% 90.20	% 55.30	% 85.52	% 110	% 267	% 89.13
odev4_2008638012.c	odev4_2009638053.c	% 71.48	% 0.00	% 47.83	% 1.96	% 65.56	% 110	% 174	% 68.98
odev1_2008638012.c	odev1_2009638034.cpp	% 68.80	% 0.00	% 47.83	% 2.10	% 61.51	% 110	% 161	% 66.56
odev1_2008638012.c	odev1_2009638010.c	% 61.38	% 0.00	% 62.50	% 2.27	% 60.37	% 110	% 142	% 62.22
odev4_2008638012.c	odev4_2009638019.c	% 58.70	% 0.00	% 43.48	% 2.73	% 53.48	% 110	% 150	% 57.14
odev1_2008638012.c	odev1_2009638076.c	% 57.97	% 0.00	% 47.83	% 0.54	% 53.01	% 110	% 168	% 56.81
odev4_2008638012.c	odev4_2009638048.c	% 57.97	% 0.00	% 47.83	% 0.54	% 53.01	% 110	% 170	% 56.81
odev4_2008638012.c	odev4_2009638060.c	% 57.51	% 0.00	% 45.88	% 2.79	% 54.50	% 110	% 180	% 56.11
odev1_2008638012.c	odev1_2009639100.c	% 57.24	% 0.00	% 43.78	% 2.13	% 51.61	% 110	% 255	% 56.25
odev1_2008638012.c	odev1_2009639500.c	% 57.24	% 0.00	% 43.78	% 2.13	% 51.61	% 110	% 256	% 56.25
odev4_2008638012.c	odev4_2009638005.cpp	% 54.82	% 0.00	% 37.70	% 10.91	% 50.79	% 110	% 137	% 53.02
odev4_2008638012.c	odev4_2009639012.c	% 56.47	% 0.00	% 19.51	% 2.70	% 48.79	% 110	% 278	% 52.72
odev1_2008638012.c	odev1_2009638402.c	% 46.83	% 0.00	% 51.55	% 0.76	% 47.28	% 110	% 345	% 52.82
odev4_2008638012.c	odev4_2009639050.c	% 46.69	% 0.00	% 54.55	% 3.05	% 47.19	% 110	% 293	% 52.58
odev4_2008638012.c	odev4_2009639039.c	% 56.47	% 0.00	% 13.95	% 2.70	% 47.80	% 110	% 288	% 52.18
odev1_2008638012.c	bagliliste.cpp	% 53.89	% 0.00	% 22.22	% 3.11	% 47.21	% 110	% 243	% 50.66
odev4_2008638012.c	odev4_2009638401.c	% 18.91	% 0.00	% 61.11	% 8.33	% 26.13	% 110	% 345	% 50.56
odev4_2008638012.c	odev4_2009638027.c	% 31.34	% 0.00	% 50.00	% 1.25	% 34.05	% 110	% 158	% 41.34
odev1_2008638012.c	odev1_2009638016.c	% 21.75	% 0.00	% 50.00	% 2.86	% 26.37	% 110	% 148	% 42.04
odev1_2008638012.c	odev1_2009638006.c	% 35.82	% 0.00	% 45.75	% 0.94	% 35.52	% 110	% 138	% 41.27
odev1_2008638012.c	odev1_2009638051.c	% 35.82	% 0.00	% 33.24	% 0.94	% 35.54	% 110	% 175	% 37.83
odev1_2008638012.c	odev1_2009638506.c	% 31.16	% 0.00	% 31.00	% 1.80	% 31.13	% 110	% 120	% 32.87

Dosyalar:odev4_2009638005.cpp

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev1_2009638005.cpp	odev1_2009638019.c	% 69.63	% 0.00	% 51.52	% 0.61	% 65.02	% 137	% 150	% 67.67
odev4_2009638005.cpp	odev4_2009638060.c	% 68.80	% 0.00	% 51.14	% 0.61	% 64.04	% 117	% 180	% 67.02
odev1_2009638005.cpp	odev1_2009638053.c	% 67.95	% 0.00	% 39.13	% 5.03	% 61.49	% 137	% 174	% 64.95
odev1_2009638005.cpp	odev1_2009638091.cpp	% 65.81	% 0.00	% 36.95	% 2.42	% 59.38	% 137	% 161	% 62.82
odev1_2009638005.cpp	odev1_2009639050.c	% 66.78	% 0.00	% 27.27	% 3.03	% 58.41	% 137	% 292	% 62.73
odev4_2009638005.cpp	odev4_2009638402.c	% 66.67	% 0.00	% 27.27	% 3.79	% 58.36	% 117	% 346	% 62.66
odev4_2009638005.cpp	odev4_2009638306.c	% 65.12	% 0.00	% 40.00	% 0.90	% 59.36	% 117	% 373	% 62.49
odev1_2009638005.cpp	bagliliste.cpp	% 63.62	% 0.00	% 33.33	% 0.62	% 56.06	% 137	% 243	% 60.40
odev4_2009638005.cpp	odev4_2009639090.c	% 62.66	% 0.00	% 34.15	% 0.61	% 56.34	% 117	% 335	% 59.71
odev4_2009638005.cpp	odev4_2009639400.c	% 62.66	% 0.00	% 34.15	% 0.61	% 56.34	% 117	% 333	% 59.71
odev4_2009638005.cpp	odev4_2009638006.c	% 62.63	% 6.50	% 31.25	% 3.77	% 55.00	% 117	% 138	% 58.41
odev1_2009638005.cpp	odev1_2009638051.c	% 62.63	% 6.67	% 26.47	% 3.77	% 55.05	% 137	% 175	% 58.95
odev1_2009638005.cpp	odev1_2009638027.c	% 57.65	% 0.00	% 43.33	% 5.00	% 51.00	% 137	% 158	% 55.10
odev1_2009638005.cpp	odev1_2009638012.c	% 58.32	% 0.00	% 21.95	% 7.88	% 50.76	% 137	% 278	% 54.63
odev1_2009638005.cpp	odev1_2009638039.c	% 58.42	% 0.00	% 18.60	% 7.88	% 50.21	% 137	% 288	% 54.40
odev1_2009638005.cpp	odev1_2009638010.c	% 51.96	% 0.00	% 15.83	% 2.12	% 33.26	% 137	% 142	% 33.91
odev1_2009638005.cpp	odev1_2009638075.c	% 55.63	% 0.00	% 30.43	% 2.42	% 50.05	% 137	% 168	% 53.02
odev1_2009638005.cpp	odev1_2009638078.c	% 55.63	% 0.00	% 30.43	% 2.42	% 50.05	% 137	% 170	% 53.02
odev4_2009638005.cpp	odev4_2009638401.c	% 41.58	% 0.00	% 50.00	% 1.67	% 43.33	% 117	% 345	% 47.88
odev1_2009638005.cpp	odev1_2009638016.c	% 46.47	% 0.00	% 45.45	% 5.21	% 45.42	% 137	% 148	% 45.24
odev4_2009638005.cpp	2009638048.c	% 45.27	% 0.00	% 50.00	% 10.91	% 43.84	% 117	% 267	% 44.35
odev4_2009638005.cpp	2009638007.c	% 45.08	% 0.00	% 37.70	% 3.48	% 43.93	% 117	% 275	% 44.23

Figure 4.35. The Detailed Result of This Hybrid Algorithm For Scenario 4

The numbers of matched tokens for every file are calculated as shown in Figure 4.36 and written to the database.

NUMBERS of MATCHED TOKENS:

File1	File2	SourceToken	CommentToken	WordToken	MaxSemanticToken
odev4_2009639500.c	odev4_2009638506.c	404	0	6	6
odev4_2009639400.c	odev4_2009638506.c	404	0	6	6
odev4_2009639400.c	odev4_2009639500.c	1011	0	41	188
odev4_2009639050.c	odev4_2009639400.c	570	0	11	2
odev4_2009639050.c	odev4_2009639500.c	570	0	11	2
odev4_2009639050.c	odev4_2009638506.c	371	0	2	5
odev4_2009639050.c	odev4_2009638401.c	212	0	8	7
odev4_2009639050.c	odev4_2009638402.c	849	23	22	5
odev4_2009639039.c	odev4_2009639500.c	794	0	9	4
odev4_2009639039.c	odev4_2009638402.c	585	0	4	4
odev4_2009639039.c	odev4_2009638401.c	214	0	3	1
odev4_2009639039.c	odev4_2009639050.c	585	0	4	4
odev4_2009639039.c	odev4_2009639400.c	794	0	9	4
odev4_2009639039.c	odev4_2009638506.c	423	0	10	4
odev4_2009639012.c	odev4_2009638402.c	583	0	6	4
odev4_2009639012.c	odev4_2009639400.c	794	0	11	6
odev4_2009639012.c	odev4_2009639500.c	794	0	11	6
odev4_2009639012.c	odev4_2009638506.c	423	0	10	6
odev4_2009639012.c	odev4_2009638401.c	214	0	5	1
odev4_2009639012.c	odev4_2009639050.c	583	0	6	4
odev4_2009639012.c	odev4_2009639039.c	1208	0	37	156
odev4_2009638402.c	odev4_2009639500.c	568	0	11	6
odev4_2009638402.c	odev4_2009638506.c	368	0	2	3
odev4_2009638402.c	odev4_2009639400.c	568	0	11	6
odev4_2009638401.c	odev4_2009638402.c	217	0	8	5
odev4_2009638401.c	odev4_2009638506.c	240	0	5	8
odev4_2009638401.c	odev4_2009639500.c	191	0	10	4
odev4_2009638401.c	odev4_2009639400.c	191	0	10	4
odev4_2009638060.c	bagliliste.cpp	586	0	4	5
odev4_2009638060.c	2009639048.c	840	0	14	5
odev4_2009638060.c	2009639007.c	842	0	15	5
odev4_2009638060.c	odev4_2009639012.c	695	0	13	9
odev4_2009638060.c	odev4_2009639039.c	693	0	12	8
odev4_2009638060.c	odev4_2009639050.c	564	0	7	5
odev4_2009638060.c	odev4_2009638506.c	430	0	10	5
odev4_2009638060.c	odev4_2009639500.c	732	0	15	5
odev4_2009638060.c	odev4_2009639400.c	732	0	15	5
odev4_2009638060.c	odev4_2009638402.c	560	0	7	4
odev4_2009638060.c	odev4_2009638401.c	225	0	9	3

Figure 4.36. Numbers of Matched Tokens For Scenario 4

The process of this hybrid algorithm takes 4134 seconds as shown in Figure 4.37. Source Code Matching Algorithm takes 4102 seconds. Comment Line Matching, Semantic Sequence Matching, Word Matching Algorithms and Tokenization take 15 seconds in total. The Match Score process takes 17 seconds. So all algorithms and processes take 4134 seconds.

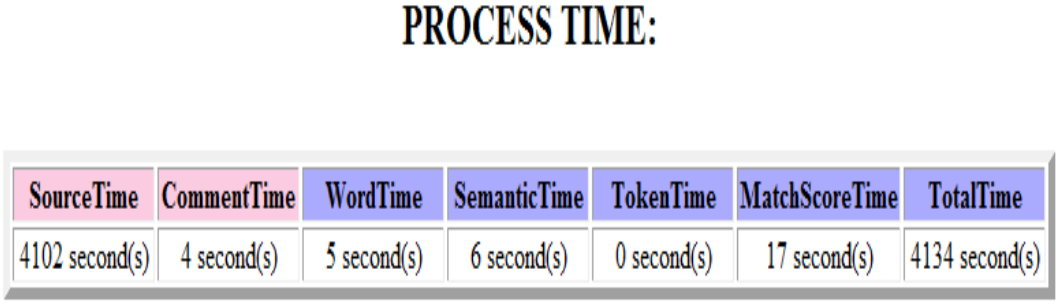


Figure 4.37. Process Time of This Hybrid Algorithm For Scenario 4

JPlag, Moss and this Hybrid Algorithm are compared with respect to similarity rate in Table 4.13. There are 300 matches in this scenario. The random ten matches are selected from the Scenario 4 assignments. As shown in the table, our hybrid algorithm is more successful than Moss and JPlag. Especially Moss cannot detect plagiarism on some assignments.

Table 4.13. The Comparison of the Similarity Rate Computed By Moss JPlag and Hybrid For Scenario 4

FILE1	**FILE2**	**MOSS**	**JPLAG**	**HYBRID**
odev4_2008638001.c	odev4_2008638012.c	15 %	39.6 %	**55.98 %**
odev4_2009638006.c	odev4_2009638010.c	9 %	28.1 %	**45.33 %**
odev4_2009638019.c	odev4_2009638060.c	98 %	99.0 %	**99.20 %**
odev4_2009638027.c	odev4_2009638045.c	0 %	38.7 %	**45.55 %**
odev4_2009638045.c	odev4_2009639050.c	79.0 %	75.7 %	**82.54 %**
odev4_2009638045.c	odev4_2009638048.c	99.0 %	**100.0 %**	**100.00 %**
odev4_2009638048.c	odev4_2009639050.c	79.0 %	75.7 %	**82.54 %**
odev4_2009638053.c	odev4_2009638054.c	9 %	37.4 %	**49.86 %**
odev4_2009638060.c	odev4_2009639400.c	0 %	60.4 %	**74.67 %**
odev4_2009639012.c	odev4_2009639039.c	97 %	**100.00%**	**100.00 %**

The process time of Moss, JPlag and Hybrid Algorithm are compared in a Table 4.14. As shown in the table, our hybrid algorithm is the slowest.

Table 4.14. The Comparison of the Process Time For Scenario 4

MOSS	**JPLAG**	**HYBRID**
30 seconds	70 seconds	4134 seconds

4.4. The Fifth Scenario

The fifth scenario consist of 30 long source codes which are submitted by different students for the same assignment. These files are compared with each other using Moss, JPlag and The Hybrid Algorithm. Source files have between 180 and 420 lines and they have comment lines.

The assignments are compared with Moss firstly. The process of Moss takes 53 seconds. Figure 4.38 shows the Moss result page of Scenario 5.

Moss Results

Sun Jul 31 03:20:58 PDT 2011

Options -l c -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

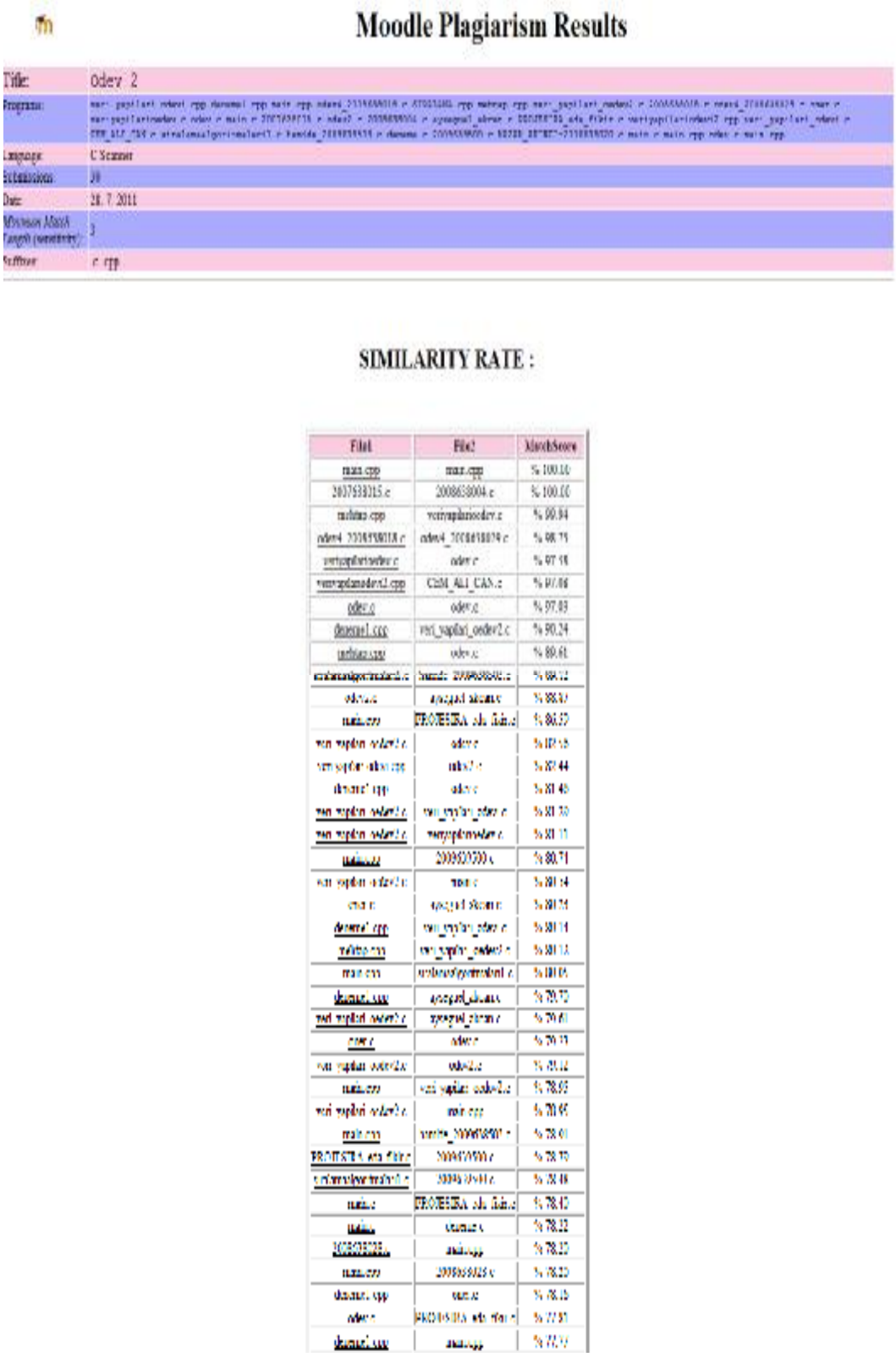
File 1	File 2	Lines Matched
c:/xampp/scenario5/2007638015.c (99%)	c:/xampp/scenario5/2008638004.c (99%)	329
c:/xampp/scenario5/odev.c (71%)	c:/xampp/scenario5/mehtap.cpp (85%)	286
c:/xampp/scenario5/main.cpp (31%)	c:/xampp/scenario5/deneme1.cpp (36%)	111
c:/xampp/scenario5/mehtap.cpp (34%)	c:/xampp/scenario5/deneme1.cpp (34%)	100
c:/xampp/scenario5/odev.c (29%)	c:/xampp/scenario5/deneme1.cpp (34%)	107
c:/xampp/scenario5/2008638028.c (35%)	c:/xampp/scenario5/deneme.c (43%)	109
c:/xampp/scenario5/mehtap.cpp (31%)	c:/xampp/scenario5/deneme.c (43%)	105
c:/xampp/scenario5/2009639500.c (34%)	c:/xampp/scenario5/deneme.c (43%)	94
c:/xampp/scenario5/odev.c (25%)	c:/xampp/scenario5/deneme.c (40%)	97
c:/xampp/scenario5/deneme.c (40%)	c:/xampp/scenario5/deneme1.cpp (29%)	97
c:/xampp/scenario5/2008638028.c (32%)	c:/xampp/scenario5/mehtap.cpp (29%)	91
c:/xampp/scenario5/main.cpp (25%)	c:/xampp/scenario5/mehtap.cpp (29%)	80
c:/xampp/scenario5/2009639500.c (31%)	c:/xampp/scenario5/odev.c (24%)	96
c:/xampp/scenario5/2009639500.c (31%)	c:/xampp/scenario5/mehtap.cpp (28%)	89
c:/xampp/scenario5/main.cpp (23%)	c:/xampp/scenario5/deneme.c (37%)	83
c:/xampp/scenario5/main.cpp (23%)	c:/xampp/scenario5/odev.c (22%)	74
c:/xampp/scenario5/2008638028.c (30%)	c:/xampp/scenario5/odev.c (22%)	84
c:/xampp/scenario5/2008638028.c (29%)	c:/xampp/scenario5/deneme1.cpp (25%)	84
c:/xampp/scenario5/2009639500.c (27%)	c:/xampp/scenario5/main.c (24%)	67
c:/xampp/scenario5/2009639500.c (27%)	c:/xampp/scenario5/deneme1.cpp (24%)	72

Figure. 4.38. Moss Results For Scenario 5

The assignments are then compared with JPlag. The process of JPlag takes 106 seconds. The result page is shown in Figure 4.39. It shows the average similarity.

Figure. 4.39. JPlag Average Similarity Results For Scenario 5

Finally, the assignments are compared at last with our hybrid algorithm. The result page is shown in Figure 4.40. It shows that the name of the assignment, the file names, the number of submissions, the minimum match length, the programming language, the suffixes and the date. The detailed result page is shown in Figure 4.41. It shows that the assignments are grouped and compared with each other. Source code similarity rate, comment line similarity rate, word matching similarity rate, semantic sequence similarity rate and match score are showed in Figure 4.41.



Moodle Plagiarism Results

Title:	Odev 2
Programs:	[illegible]
Language:	C Sconner
Submissions:	10
Date:	28. 7. 2011
Minimum Match Length (sentences):	3
Suffixes:	c cpp

SIMILARITY RATE :

File1	File2	MatchScore
main.cpp	main.cpp	% 100.00
20075938215.c	20086638004.c	% 100.00
mainmo.cpp	vesiyapilarinodev.c	% 99.84
odev4_20086938018.c	odev4_20086838079.c	% 98.75
vesiyapilarinodev.c	odev.c	% 97.98
vesiyapilarinodev2.cpp	CbM_All_CAN.c	% 97.86
odev.c	odev.c	% 97.83
dosyam1.cpp	ves_yapilari_odev2.c	% 90.24
mebitan.cpp	odev.c	% 89.61
[illegible]	[illegible]	% 88.12
odev.c	[illegible]	% 88.03
main.cpp	[illegible]	% 86.37
ves yapilari odev1.c	odev.c	% 85.18
ves yapilari odev.cpp	main2.c	% 85.04
dosyam1.cpp	odev.c	% 84.46
ves yapilari odev1.c	ves_yapilari_odev.c	% 84.30
ves yapilari odev1.c	vesiyapilarinodev.c	% 84.17
main.cpp	20086638004.c	% 80.71
ves yapilari odev1.c	main.c	% 80.14
main.c	[illegible]	% 80.14
dosyam1.cpp	ves_yapilari_odev.c	% 80.14
mebitan.cpp	ves_yapilari_odev2.c	% 80.13
main.cpp	[illegible]	% 80.04
dosyam.cpp	[illegible]	% 79.72
ves yapilari odev1.c	[illegible]	% 79.61
main.c	odev.c	% 79.19
ves yapilari odev2.c	main2.c	% 79.12
main.cpp	ves_yapilari_odev2.c	% 78.93
ves yapilari odev2.c	main.cpp	% 78.66
main.cpp	vesiyapilarinodev2.c	% 78.61
[illegible]	20086638004.c	% 78.59
[illegible]	[illegible]	% 78.48
main.c	[illegible]	% 78.43
[illegible]	[illegible]	% 78.22
[illegible]	[illegible]	% 78.22
main.cpp	20086638004.c	% 78.20
dosyam.cpp	main.c	% 78.16
odev.c	[illegible]	% 77.93
dosyam1.cpp	[illegible]	% 77.77

Figure 4.40. The Result of This Hybrid Algorithm For Scenario 5

SIMILARITY RATE :

Dosyalar:veri yapilari odevi.cpp

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1 No	User2 No	Detailed MatchScore
veri yapilari odevi.cpp	odev2.c	% 85.79	% 0.00	% 53.85	% 2.01	% 78.41	101	205	% 82.41
veri yapilari odevi.cpp	SIRALAMA.cpp	% 75.42	% 31.30	% 73.68	% 2.01	% 73.93	104	113	% 73.06
veri yapilari odevi.cpp	veri yapilari oedev2.c	% 77.17	% 16.67	% 52.50	% 4.32	% 71.45	101	120	% 71.57
veri yapilari odevi.cpp	veriyapilariodevi2.cpp	% 77.80	% 0.00	% 45.00	% 5.03	% 70.30	101	254	% 71.22
veri yapilari odevi.cpp	2008638004.c	% 74.32	% 0.00	% 47.50	% 4.02	% 68.10	101	207	% 71.50
veri yapilari odevi.cpp	2007638015.c	% 74.32	% 0.00	% 47.50	% 4.02	% 68.10	104	192	% 71.50
veri yapilari odevi.cpp	main.cpp	% 73.73	% 0.00	% 50.00	% 4.02	% 68.07	104	114	% 71.22
veri yapilari odevi.cpp	main.cpp	% 73.73	% 0.00	% 50.00	% 4.02	% 68.07	101	110	% 71.22
veri yapilari odevi.cpp	main.cpp	% 73.73	% 0.00	% 50.00	% 4.02	% 68.07	104	126	% 71.22
veri yapilari odevi.cpp	CEM ALI CAN.c	% 74.26	% 0.00	% 42.50	% 0.50	% 67.13	101	303	% 70.06
veri yapilari odevi.cpp	aysegul_akcan.c	% 72.30	% 0.00	% 57.50	% 2.51	% 68.24	104	243	% 70.63
veri yapilari odevi.cpp	veri yapilari odevi.c	% 72.41	% 0.00	% 45.00	% 2.01	% 66.13	101	280	% 69.57
veri yapilari odevi.cpp	veriyapilarioedev.c	% 71.23	% 19.05	% 55.00	% 2.51	% 67.13	101	172	% 69.47
veri yapilari odevi.cpp	mehtap.cpp	% 71.04	% 3.83	% 52.50	% 3.02	% 66.39	104	118	% 69.04
veri yapilari odevi.cpp	main.c	% 71.75	% 4.62	% 42.50	% 0.50	% 65.16	101	255	% 68.70
veri yapilari odevi.cpp	main.c	% 68.73	% 4.62	% 42.50	% 0.50	% 65.16	104	179	% 68.70
veri yapilari odevi.cpp	deneme.c	% 71.38	% 0.00	% 45.00	% 0.50	% 65.26	101	253	% 68.61
veri yapilari odevi.cpp	deneme1.cpp	% 71.23	% 7.89	% 45.00	% 7.54	% 65.29	101	109	% 68.49
veri yapilari odevi.cpp	oner.c	% 69.73	% 0.00	% 50.00	% 2.51	% 64.85	104	125	% 67.61
veri yapilari odevi.cpp	NAZAN AKINCI 2006638020.c	% 70.53	% 8.82	% 32.50	% 2.51	% 62.78	101	85	% 67.00
veri yapilari odevi.cpp	PROJESIRA eda fikir.c	% 68.78	% 0.00	% 52.50	% 2.51	% 64.53	101	253	% 67.00
veri yapilari odevi.cpp	hamide 2006638503.c	% 69.22	% 0.00	% 27.50	% 4.02	% 60.45	101	349	% 64.97
veri yapilari odevi.cpp	2009638500.c	% 66.84	% 0.00	% 47.50	% 2.01	% 62.08	104	356	% 64.77
veri yapilari odevi.cpp	odev.c	% 65.41	% 0.00	% 55.00	% 2.01	% 62.76	104	176	% 64.76
veri yapilari odevi.cpp	odev.c	% 66.02	% 0.00	% 55.00	% 2.01	% 62.76	101	178	% 64.76
veri yapilari odevi.cpp	siralamaalgoritmalari1.c	% 68.98	% 0.00	% 27.50	% 4.02	% 60.76	104	339	% 64.76
veri yapilari odevi.cpp	2008638028.c	% 66.42	% 12.20	% 42.50	% 2.51	% 60.98	104	121	% 63.92
veri yapilari odevi.cpp	odev4_2008638018.c	% 42.42	% 0.00	% 12.50	% 0.56	% 36.25	104	113	% 39.39
veri yapilari odevi.cpp	odev1 2008638029.c	% 42.33	% 0.00	% 12.12	% 0.56	% 36.11	101	122	% 38.27

Dosyalar:deneme1.cpp

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1 No	User2 No	Detailed MatchScore
deneme1.cpp	veri yapilari oedev2.c	% 92.03	% 20.00	% 76.00	% 8.59	% 87.62	109	120	% 90.24
deneme1.cpp	odev.c	% 83.45	% 0.00	% 46.94	% 1.51	% 76.90	109	178	% 81.46
deneme1.cpp	odev.c	% 83.83	% 0.00	% 46.94	% 1.51	% 76.90	109	176	% 81.46
deneme1.cpp	veri_yapilari_odevi.c	% 83.65	% 0.00	% 30.00	% 1.45	% 76.00	109	280	% 80.14
deneme1.cpp	aysegul akcan.c	% 81.98	% 0.00	% 50.98	% 3.92	% 76.61	109	243	% 79.70
deneme1.cpp	oner.c	% 81.88	% 0.00	% 46.00	% 2.11	% 73.88	109	125	% 78.15
deneme1.cpp	main.cpp	% 79.90	% 0.00	% 39.32	% 2.68	% 74.77	109	126	% 77.77
deneme1.cpp	main.cpp	% 79.99	% 0.00	% 39.32	% 2.68	% 74.77	109	110	% 77.77
deneme1.cpp	main.cpp	% 79.99	% 0.00	% 39.32	% 2.68	% 74.77	109	114	% 77.77
deneme1.cpp	veriyapilarioedev.c	% 79.80	% 6.25	% 47.92	% 2.16	% 72.89	109	172	% 76.57
deneme1.cpp	mehtap.cpp	% 80.01	% 0.00	% 41.67	% 2.16	% 71.61	109	118	% 76.05
deneme1.cpp	main.c	% 73.89	% 0.00	% 38.64	% 1.56	% 71.10	109	179	% 75.81
deneme1.cpp	main.c	% 80.06	% 0.00	% 38.64	% 1.56	% 71.10	109	255	% 75.81
deneme1.cpp	odev2.c	% 75.22	% 0.00	% 61.54	% 1.47	% 71.30	109	205	% 73.67
deneme1.cpp	SIRALAMA.cpp	% 74.98	% 2.11	% 60.53	% 2.05	% 70.85	109	113	% 73.36
deneme1.cpp	PROJESIRA_eda_fikir.c	% 75.67	% 0.00	% 42.00	% 1.88	% 68.18	109	253	% 73.18
deneme1.cpp	2009638500.c	% 73.16	% 0.00	% 48.78	% 1.44	% 67.37	109	356	% 70.58
deneme1.cpp	2008638028.c	% 72.66	% 3.23	% 40.48	% 4.18	% 65.55	109	121	% 69.33
deneme1.cpp	2008638004.c	% 71.53	% 0.00	% 45.83	% 3.14	% 65.56	109	207	% 68.83
deneme1.cpp	2007638015.c	% 71.53	% 0.00	% 45.83	% 3.14	% 65.56	109	192	% 68.83
deneme1.cpp	hamide_2006638503.c	% 72.40	% 0.00	% 28.00	% 2.83	% 63.08	109	349	% 67.88
deneme1.cpp	siralamaalgoritmalari1.c	% 71.53	% 0.00	% 28.00	% 2.64	% 62.28	109	339	% 67.10
deneme1.cpp	deneme.c	% 69.06	% 0.00	% 34.00	% 2.16	% 61.46	109	253	% 65.46
deneme1.cpp	veriyapilariodevi2.cpp	% 68.10	% 0.00	% 38.00	% 3.79	% 61.42	109	254	% 64.98
deneme1.cpp	CEM ALI CAN.c	% 63.98	% 0.00	% 36.00	% 2.06	% 57.74	109	303	% 61.08
deneme1.cpp	NAZAN AKINCI 2006638020.c	% 62.12	% 8.33	% 30.00	% 1.94	% 55.26	109	85	% 58.83
deneme1.cpp	odev4_2008638018.c	% 41.31	% 0.00	% 18.75	% 1.68	% 36.65	109	113	% 39.15
deneme1.cpp	odev1 2008638029.c	% 41.51	% 0.00	% 18.18	% 1.68	% 36.54	109	122	% 38.13

Figure 4.41. The Detailed Result of This Hybrid Algorithm For Scenario 5

The numbers of matched tokens for every file are calculated as shown in Figure 4.42 and written to the database.

NUMBERS of MATCHED TOKENS:

File1	File2	SourceToken	CommentToken	WordToken	MaxSemanticToken
veri_yapilari_oedev2.c	veriyapilariodevi2.cpp	1248	0	24	6
veri_yapilari_oedev2.c	main.c	1373	0	23	4
veri_yapilari_oedev2.c	odev.c	1599	0	32	4
veri_yapilari_oedev2.c	odev.c	1558	0	32	4
veri_yapilari_oedev2.c	ayseguel_akcan.c	1617	0	27	5
veri_yapilari_oedev2.c	hamide_2009638503.c	1382	0	14	7
veri_yapilari_oedev2.c	2009639500.c	1302	0	23	4
veri_yapilari_oedev2.c	odev4_2008638029.c	614	0	5	3
veri_yapilari_oedev2.c	oner.c	1551	0	25	4
veri_yapilari_oedev2.c	main.cpp	1520	0	26	6
veri_yapilari_oedev2.c	2007638015.c	1231	0	26	5
veri_yapilari_oedev2.c	odev2.c	1317	0	26	4
veri_yapilari_oedev2.c	2008638004.c	1231	0	26	5
veri_yapilari_oedev2.c	veriyapilarioedev.c	1430	8	32	5
veri_yapilari_oedev2.c	NAZAN_AKINCI-2006638020.c	989	3	18	4
veri_yapilari_oedev2.c	deneme.c	1141	0	24	5
veri_yapilari_oedev2.c	hamide_2009638503.c	1382	0	14	7
veri_yapilari_oedev2.c	siralamaalgoritmalari1.c	1327	0	13	6
veri_yapilari_oedev2.c	CEM_ALI_CAN.c	1121	0	23	5
veri_yapilari_oedev2.c	veri_yapilari_odevi.c	1440	0	21	5
veri_yapilari_oedev2.c	main.c	1436	0	23	4
veri_yapilari_oedev2.c	PROJESIRA_eda_fikir.c	1493	0	27	5
veri_yapilari_oedev2.c	2008638028.c	1245	2	18	5
veri_yapilari_oedev2.c	2009639500.c	1302	0	23	4
veri_yapilari_oedev2.c	PROJESIRA_eda_fikir.c	1493	0	27	5
veri_yapilari_oedev2.c	odev4_2008638029.c	614	0	5	3
veri_yapilari_oedev2.c	siralamaalgoritmalari1.c	1327	0	13	6
veri_yapilari_oedev2.c	oner.c	1551	0	25	4
veri_yapilari_oedev2.c	main.cpp	1520	0	26	6
veri_yapilari_oedev2.c	veriyapilarioedev.c	1430	8	32	5
veri_yapilari_oedev2.c	odev.c	1558	0	32	4
veri_yapilari_oedev2.c	odev.c	1599	0	32	4
veri_yapilari_oedev2.c	veri_yapilari_odevi.c	1440	0	21	5
veri_yapilari_oedev2.c	main.c	1373	0	23	4
veri_yapilari_oedev2.c	veriyapilariodevi2.cpp	1248	0	24	6
veri_yapilari_oedev2.c	odev2.c	1317	0	26	4
veri_yapilari_oedev2.c	2008638004.c	1231	0	26	5

Figure 4.42. Numbers of Matched Tokens For Scenario 5

The process of this hybrid algorithm takes 21807 seconds as shown in Figure 4.43. Source Code Matching Algorithm takes 21732 seconds. Comment Line Matching takes 8 seconds. Semantic Sequence Matching takes 18 seconds. Word Matching Algorithm takes 12 seconds and Tokenization processes take 1 second. The Match Score process takes 36 seconds. So all algorithms and processes take 21807 seconds.

PROCESS TIME:

SourceTime	CommentTime	WordTime	SemanticTime	TokenTime	MatchScoreTime	TotalTime
21732 second (s)	8 second(s)	12 second(s)	18 second(s)	1 second(s)	36 second(s)	21807 second (s)

Figure 4.43. Process Time of This Hybrid Algorithm For Scenario 5

JPlag, Moss and this Hybrid Algorithm are compared with respect to similarity rate in Table 4.15. There are 435 matches in this scenario. The random ten matches are selected from the Scenario 5 assignments and compared in Table 4.15. As shown in the table, our hybrid algorithm is more successful than Moss and JPlag.

Table 4.15. The Comparison of the Similarity Rate Computed By Moss JPlag and Hybrid For Scenario 5

FILE1	**FILE2**	**MOSS**	**JPLAG**	**HYBRID**
2007638015.c	2008638004.c	99 %	**100 %**	**100 %**
deneme1.cpp	veri_yapilari_oedev2.c	96 %	**99.1 %**	90.24 %
veriyapilarioedev.c	odev.c	84 %	85.9 %	**97.58 %**
oner.c	ayseguel_akcan.c	30 %	66.5%	**80.23 %**
veri_yapilari_oedev2.c	main.c	16 %	49.7 %	**80.54 %**
main.c	veri_yapilari_odevi.c	22 %	53.6 %	**75.40 %**
2008638028.c	hamide_2009638503.c	10 %	56.8%	**71.35 %**
odev.c	main.c	18 %	47.7%	**75.81 %**
ayseguel_akcan.c	PROJESIRA_eda_fikir.c	30 %	51.1%	**72.54 %**
odev.c	2007638015.c	23 %	69.0 %	**69.30 %**

The process time of Moss, JPlag and Hybrid Algorithm are compared in Table 4.16. As shown in the table, our hybrid algorithm is the slowest.

Table 4.16. The Comparison of the Process Time For Scenario 5

MOSS	**JPLAG**	**HYBRID**
53 seconds	106 seconds	21807 seconds

4.3. The Sixth Scenario

The sixth scenario consist of 25 long source codes which are submitted by different students of the same assignment. These files are compared with each other using Moss, JPlag and The Hybrid Algorithm. Source files have between 84 and 601 lines and they do not have comment lines.

The assignments are compared with Moss firstly. The process of Moss takes 36 seconds. The command line for running Moss is shown in Figure 4.44. The result page is shown as in Figure 4.45.

```
perl Moss.pl -l c c:\xampp\scenario6\odev3_2009639010.c
c:\xampp\scenario6\odev3_2009639061.c c:\xampp\scenario6\odev3_2008639012.c
c:\xampp\scenario6\odev3_2009639018.c c:\xampp\scenario6\odev3_2008639005.c
c:\xampp\scenario6\odev3_2009639006.c c:\xampp\scenario6\2008638057.c
c:\xampp\scenario6\odev3_2009638045.c c:\xampp\scenario6\odev3_2009638050.c
c:\xampp\scenario6\odev3_2009638048.c c:\xampp\scenario6\odev3_2009638006.c
c:\xampp\scenario6\odev3_2009638033.c c:\xampp\scenario6\odev3_2008638001.c
c:\xampp\scenario6\odev3_2008638017.cpp c:\xampp\scenario6\odev3_2009638050.c
c:\xampp\scenario6\odev3_2009638018.c c:\xampp\scenario6\2009638504.cpp
c:\xampp\scenario6\odev3_2009638506.c c:\xampp\scenario6\odev3_2009639500.c
c:\xampp\scenario6\odev3_2009639504.c c:\xampp\scenario6\odev3_2008638001.c
c:\xampp\scenario6\odev3_2009638402.c c:\xampp\scenario6\odev3_2009638019.c
c:\xampp\scenario6\odev3_26567644358.c c:\xampp\scenario6\ 2008638004.c
```

Figure 4.44. Moss Script For Scenario 6

Moss Results

Sun Jul 24 03:20:56 PDT 2011

Options -l c -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
c:\xampp\scenario3\odev4_2008638001.c (84%)	c:\xampp\scenario3\odev4_2009638019.c (83%)	156
c:\xampp\scenario3\odev4_2009638401.c (79%)	c:\xampp\scenario3\odev4_2009638506.c (40%)	59
c:\xampp\scenario3\odev4_2009638019.c (10%)	c:\xampp\scenario3\odev4_2009638506.c (18%)	26
c:\xampp\scenario3\odev4_2008638012.c (3%)	c:\xampp\scenario3\odev4_2009638005.cpp (8%)	28
c:\xampp\scenario3\odev4_2009638019.c (5%)	c:\xampp\scenario3\odev4_2009638401.c (18%)	13
c:\xampp\scenario3\odev4_2008638012.c (2%)	c:\xampp\scenario3\odev4_2009638019.c (4%)	6
c:\xampp\scenario3\odev4_2008638001.c (4%)	c:\xampp\scenario3\odev4_2008638012.c (2%)	6
c:\xampp\scenario3\odev4_2008638012.c (1%)	c:\xampp\scenario3\odev4_2009638506.c (6%)	10
c:\xampp\scenario3\odev4_2008638012.c (1%)	c:\xampp\scenario3\odev4_2009638401.c (13%)	10

Any errors encountered during this query are listed below.

Figure 4.45. Moss Results For Scenario 6

The assignments are compared with JPlag. The process of JPlag takes 71 seconds. The result page is shown as in Figure 4.46. It shows the average similarity.

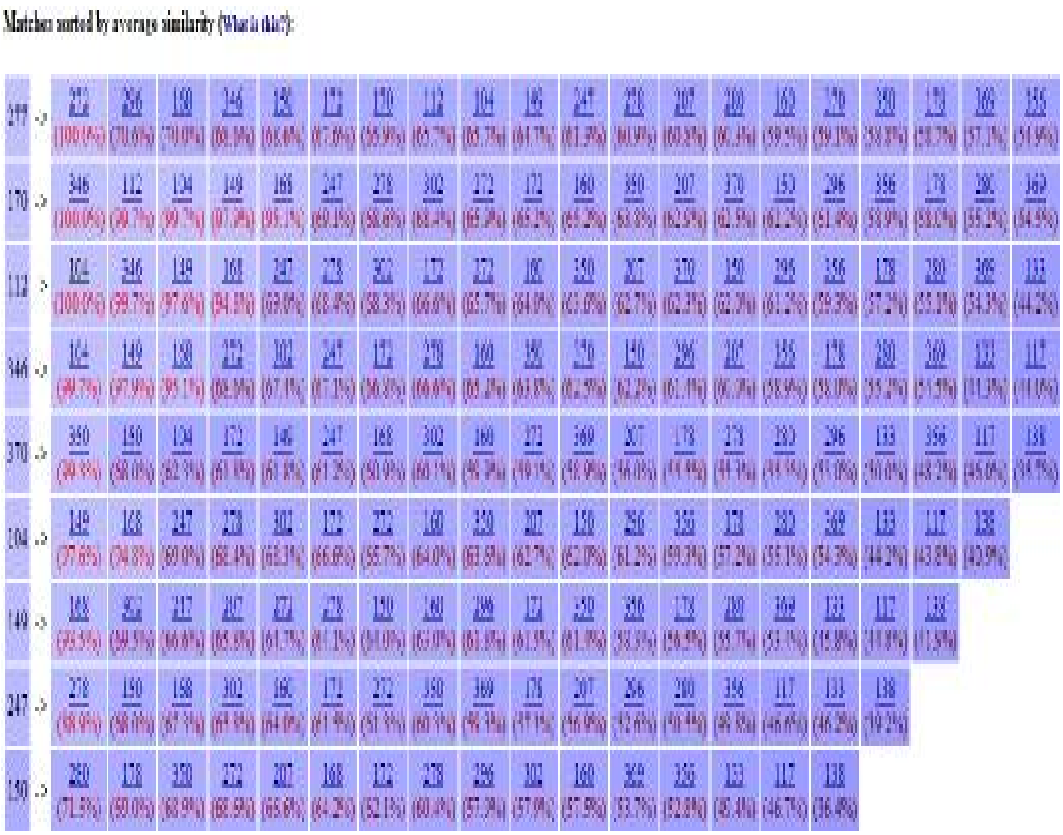
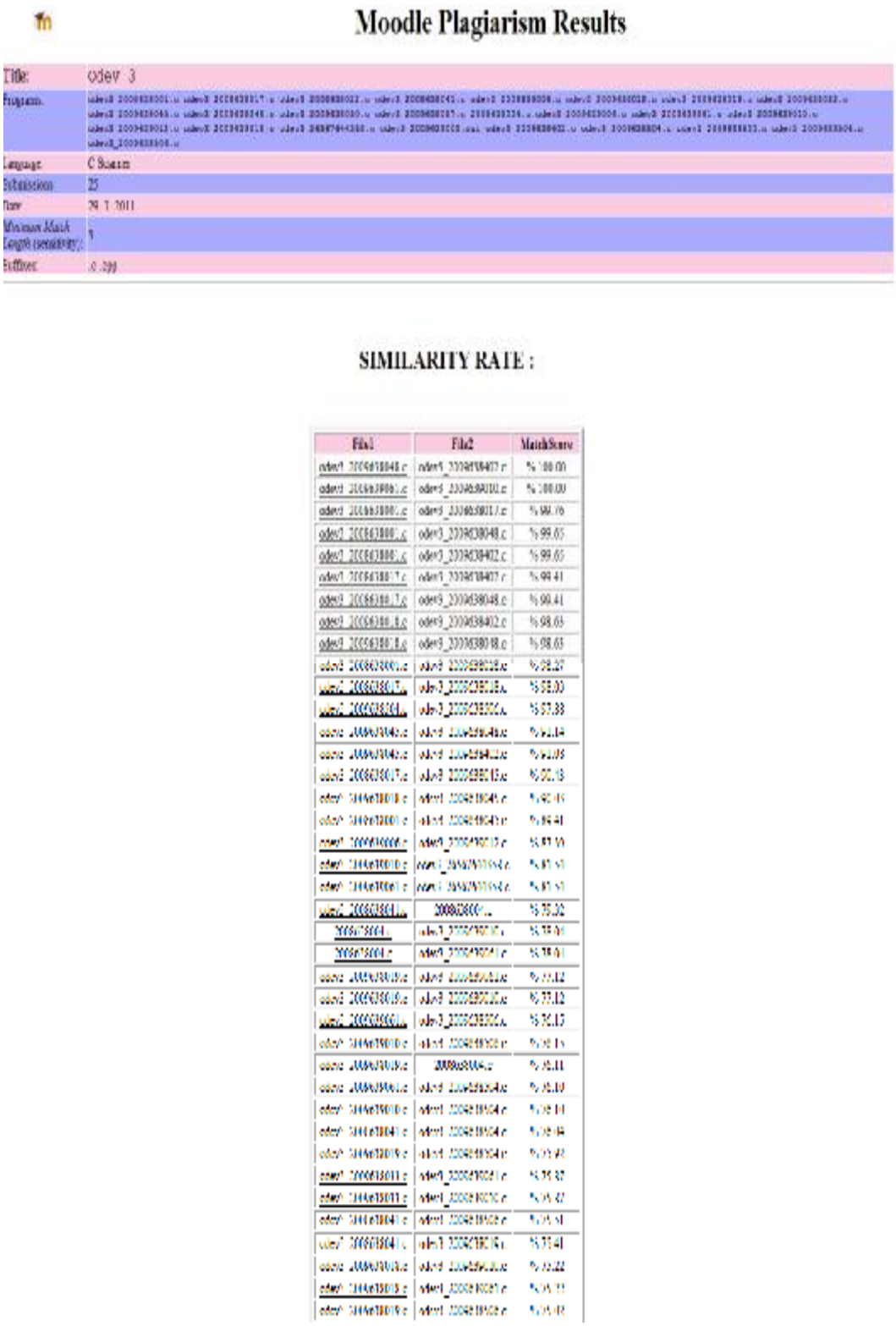


Figure 4.46. JPlag Average Similarity Results For Scenario 6

Finally, the assignments are compared with our hybrid algorithm. The result page is shown in Figure 4.47. It shows that the name of the assignment, the file names, the number of submissions, the minimum match length, the programming language, the suffixes and the date.

The detailed result page is shown in Figure 4.48. It shows that the assignments are grouped and compared with each other. Source code similarity rate, comment line similarity rate, word matching similarity rate, semantic sequence similarity rate and match score are showed in Figure 4.48.



Moodle Plagiarism Results

Title:	odev 3
Programs:	[illegible]
Language:	C Source
Submissions:	25
Date:	29 1 2011
Minimum Match Length (sensitivity):	9
Prefixes:	0 .jpg

SIMILARITY RATE :

File1	File2	MatchScore
[illegible]	[illegible]	% 100.00
[illegible]	[illegible]	% 100.00
[illegible]	[illegible]	% 99.76
[illegible]	[illegible]	% 99.65
[illegible]	[illegible]	% 99.65
[illegible]	[illegible]	% 99.41
[illegible]	[illegible]	% 99.41
[illegible]	[illegible]	% 98.65
[illegible]	[illegible]	% 98.65
[illegible]	[illegible]	% 98.27
[illegible]	[illegible]	% 98.00
[illegible]	[illegible]	% 97.88
[illegible]	[illegible]	% 94.14
[illegible]	[illegible]	% 94.06
[illegible]	[illegible]	% 92.13
[illegible]	[illegible]	% 92.13
[illegible]	[illegible]	% 89.41
[illegible]	[illegible]	% 89.39
[illegible]	[illegible]	% 87.51
[illegible]	[illegible]	% 87.51
[illegible]	[illegible]	% 79.32
[illegible]	[illegible]	% 78.00
[illegible]	[illegible]	% 78.00
[illegible]	[illegible]	% 77.12
[illegible]	[illegible]	% 77.12
[illegible]	[illegible]	% 76.13
[illegible]	[illegible]	% 76.13
[illegible]	[illegible]	% 76.11
[illegible]	[illegible]	% 76.10
[illegible]	[illegible]	% 75.94
[illegible]	[illegible]	% 75.94
[illegible]	[illegible]	% 75.87
[illegible]	[illegible]	% 75.87
[illegible]	[illegible]	% 75.41
[illegible]	[illegible]	% 77.41
[illegible]	[illegible]	% 72.22
[illegible]	[illegible]	% 72.22
[illegible]	[illegible]	% 72.22

Figure 4.47. The Result of This Hybrid Algorithm For Scenario 6

SIMILARITY RATE :

Duygularının3_20060938001.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev3_20060938001.c	odev3_20060938017.c	%99.76	%0.00	%77.90	%23.26	%99.76	104	112	%99.76
odev3_20060938001.c	odev3_20060938102.c	%99.65	%0.00	%78.05	%23.26	%99.65	101	316	%99.65
odev3_20060938001.c	odev3_20060938015.c	%99.65	%0.00	%78.05	%20.00	%99.65	101	149	%99.65
odev3_20060938001.c	odev3_20060938015.c	%98.27	%0.00	%78.05	%1.55	%98.27	101	149	%98.27
odev3_20060938001.c	odev3_20060938015.c	%91.22	%0.00	%68.15	%21.26	%91.07	101	188	%89.41
odev3_20060938001.c	odev3_20060938010.c	%78.04	%0.00	%52.26	%0.95	%95.05	104	277	%74.18
odev3_20060938001.c	odev3_20060938061.c	%78.04	%0.00	%52.26	%0.95	%95.05	101	272	%74.18
odev3_20060938001.c	odev3_20060938019.c	%77.57	%0.00	%52.14	%0.81	%97.70	104	130	%72.66
odev3_20060938001.c	odev3_20060938001.c	%73.66	%0.00	%25.72	%0.81	%97.03	101	350	%72.57
odev3_20060938001.c	20060938004.c	%77.05	%0.00	%50.05	%3.10	%97.10	104	207	%72.33
odev3_20060938001.c	odev3_20060938050.c	%78.57	%0.00	%55.14	%0.78	%97.43	101	172	%72.05
odev3_20060938001.c	odev3_20060938033.c	%75.09	%0.00	%51.03	%0.88	%95.05	101	150	%71.89
odev3_20060938001.c	odev3_20060938026.cut	%77.18	%0.00	%50.00	%0.78	%97.47	101	302	%71.10
odev3_20060938001.c	odev3_20060938006.c	%76.92	%0.00	%53.71	%0.63	%95.95	104	370	%71.96
odev3_20060938001.c	odev3_26567644958.c	%74.46	%0.00	%53.71	%2.70	%95.10	104	266	%70.48
odev3_20060938001.c	odev3_20060938017.c	%77.42	%0.00	%52.90	%0.78	%92.98	104	278	%69.85
odev3_20060938001.c	odev3_20060938006.c	%77.35	%0.00	%52.90	%0.65	%92.99	104	347	%69.78
odev3_20060938001.c	odev3_20060938041.c	%69.68	%0.00	%48.58	%1.60	%92.38	104	193	%69.27
odev3_20060938001.c	odev3_20060938018.c	%68.15	%0.00	%58.10	%1.16	%97.44	104	280	%69.02
odev3_20060938001.c	odev3_20060938051.c	%68.65	%0.00	%50.69	%1.05	%98.95	101	178	%68.97
odev3_20060938001.c	odev3_20060938101.c	%68.81	%0.00	%51.50	%0.78	%98.53	101	389	%68.66
odev3_20060938001.c	odev3_20060938102.c	%62.41	%0.00	%52.50	%3.20	%95.12	101	316	%68.50
odev3_20060938001.c	odev3_20060938022.c	%54.10	%0.00	%49.31	%1.27	%90.99	104	117	%53.80
odev3_20060938001.c	odev3_20060938005.c	%52.20	%0.00	%55.78	%2.89	%98.22	101	138	%50.15

Duygularının3_20060938017.c

File1	File2	Source	Comment	Word	Semantic	MatchScore	User1_No	User2_No	Detailed_MatchScore
odev3_20060938017.c	odev3_20060938015.c	%99.71	%0.00	%78.95	%23.26	%99.71	112	149	%99.41
odev3_20060938017.c	odev3_20060938102.c	%99.71	%0.00	%78.95	%21.49	%99.71	112	316	%99.41
odev3_20060938017.c	odev3_20060938015.c	%98.03	%0.00	%78.95	%0.88	%98.03	112	149	%98.03
odev3_20060938017.c	odev3_20060938015.c	%91.67	%0.00	%53.83	%1.57	%93.53	112	188	%90.13
odev3_20060938017.c	odev3_20060938061.c	%79.07	%0.00	%52.09	%3.81	%98.02	112	272	%73.69
odev3_20060938017.c	odev3_20060938010.c	%79.07	%0.00	%52.09	%3.81	%98.02	112	277	%73.69
odev3_20060938017.c	odev3_20060938004.c	%78.63	%0.00	%52.57	%0.81	%98.13	112	330	%73.54
odev3_20060938017.c	odev3_20060938019.c	%77.57	%0.00	%53.71	%3.23	%98.08	112	130	%73.26
odev3_20060938017.c	20060938007.c	%78.18	%0.00	%51.01	%3.88	%98.22	112	207	%72.70
odev3_20060938017.c	odev3_20060938050.c	%78.71	%0.00	%53.14	%2.64	%97.80	112	172	%72.45
odev3_20060938017.c	odev3_20060938005.c	%77.20	%0.00	%28.57	%1.85	%97.10	112	350	%72.31
odev3_20060938017.c	odev3_20060938033.c	%77.10	%0.00	%52.57	%1.77	%98.04	112	150	%72.16
odev3_20060938017.c	odev3_20060938026.cut	%77.18	%0.00	%50.00	%0.88	%97.47	112	302	%71.80
odev3_20060938017.c	odev3_26567644958.c	%74.63	%0.00	%53.71	%1.60	%95.98	112	266	%70.61
odev3_20060938017.c	odev3_20060938005.c	%72.25	%0.00	%55.00	%0.85	%94.26	112	247	%68.51
odev3_20060938017.c	odev3_20060938012.c	%72.19	%0.00	%55.00	%1.36	%94.14	112	278	%68.37
odev3_20060938017.c	odev3_20060938018.c	%68.15	%0.00	%57.62	%1.16	%93.13	112	280	%66.91
odev3_20060938017.c	odev3_20060938013.c	%68.81	%0.00	%50.41	%3.81	%91.28	112	133	%65.72
odev3_20060938017.c	odev3_20060938057.c	%68.69	%0.00	%50.69	%1.05	%92.02	112	178	%64.10
odev3_20060938017.c	odev3_20060938104.c	%68.64	%0.00	%50.00	%0.68	%98.78	112	369	%63.90
odev3_20060938017.c	odev3_20060938100.c	%67.81	%0.00	%52.50	%2.72	%96.19	112	356	%63.69
odev3_20060938017.c	odev3_20060938022.c	%53.15	%0.00	%52.31	%1.27	%90.81	112	117	%52.66
odev3_20060938017.c	odev3_20060938005.c	%52.66	%0.00	%55.78	%1.40	%95.14	112	138	%51.15

Figure 4.48. The Detailed Result of This Hybrid Algorithm For Scenario 6

The numbers of matched tokens for every file are calculated as shown in Figure 4.49. and written to the database.

NUMBERS of MATCHED TOKENS:

File1	File2	SourceToken	CommentToken	WordToken	MaxSemanticToken
odev3_26567644358.c	odev3_2009638504.c	584	0	4	1
odev3_26567644358.c	odev3_2009639005.out	612	3	9	1
odev3_26567644358.c	odev3_2009639500.c	616	0	12	1
odev3_26567644358.c	odev3_2009639504.c	672	6	11	1
odev3_26567644358.c	odev3_2009638506.c	563	0	4	1
odev3_26567644358.c	odev3_2009638402.c	627	0	10	2
odev3_2009639504.c	odev3_2009638506.c	654	0	11	4
odev3_2009639500.c	odev3_2009638506.c	573	0	8	1
odev3_2009639500.c	odev3_2009639504.c	776	0	14	4
odev3_2009639061.c	odev3_26567644358.c	654	24	18	5
odev3_2009639061.c	odev3_2009639010.c	737	60	31	105
odev3_2009639061.c	odev3_2009639012.c	604	0	17	1
odev3_2009639061.c	odev3_2009638506.c	590	0	6	2
odev3_2009639061.c	odev3_2009639504.c	636	3	13	1
odev3_2009639061.c	odev3_2009639500.c	568	0	12	1
odev3_2009639061.c	odev3_2009638504.c	602	0	6	1
odev3_2009639061.c	odev3_2009639018.c	396	0	10	1
odev3_2009639061.c	odev3_2009638402.c	622	0	8	2
odev3_2009639061.c	odev3_2009639005.out	624	1	11	1
odev3_2009639018.c	odev3_2009638402.c	439	0	10	1
odev3_2009639018.c	odev3_2009639005.out	377	0	8	1
odev3_2009639018.c	odev3_26567644358.c	362	0	10	1
odev3_2009639018.c	odev3_2009638504.c	368	0	7	1
odev3_2009639018.c	odev3_2009639500.c	383	0	10	1
odev3_2009639018.c	odev3_2009639504.c	391	0	9	2
odev3_2009639018.c	odev3_2009638506.c	368	0	7	1
odev3_2009639012.c	odev3_26567644358.c	578	0	12	1
odev3_2009639012.c	odev3_2009639005.out	678	0	12	1
odev3_2009639012.c	odev3_2009639018.c	417	0	12	1
odev3_2009639012.c	odev3_2009638506.c	560	0	7	1
odev3_2009639012.c	odev3_2009638402.c	644	0	13	4
odev3_2009639012.c	odev3_2009638504.c	566	0	7	1
odev3_2009639012.c	odev3_2009639500.c	600	0	13	1
odev3_2009639012.c	odev3_2009639504.c	658	0	12	4
odev3_2009639010.c	odev3_2009639500.c	568	0	12	1
odev3_2009639010.c	odev3_2009639012.c	604	0	17	1
odev3_2009639010.c	odev3_2009639018.c	396	0	10	1
odev3_2009639010.c	odev3_2009639005.out	624	1	11	1
odev3_2009639010.c	odev3_2009638402.c	622	0	8	2
odev3_2009639010.c	odev3_2009638504.c	602	0	6	1
odev3_2009639010.c	odev3_2009639504.c	636	3	13	1
odev3_2009639010.c	odev3_26567644358.c	654	24	18	5
odev3_2009639010.c	odev3_2009638506.c	590	0	6	2
odev3_2009639006.c	odev3_2009639504.c	650	0	11	1
odev3_2009639006.c	odev3_2009638504.c	562	0	7	3
odev3_2009639006.c	odev3_2009638402.c	619	0	14	2
odev3_2009639006.c	odev3_2009639005.out	679	0	10	5
odev3_2009639006.c	odev3_26567644358.c	550	0	12	1
odev3_2009639006.c	odev3_2009639018.c	419	0	11	4
odev3_2009639006.c	odev3_2009639500.c	594	0	14	1
odev3_2009639006.c	odev3_2009639061.c	604	0	14	2
odev3_2009639006.c	odev3_2009639012.c	807	0	24	6
odev3_2009639006.c	odev3_2009639010.c	604	0	14	2
odev3_2009639006.c	odev3_2009638506.c	559	0	7	1
odev3_2009639005.out	odev3_2009638402.c	696	0	8	2
odev3_2009639005.out	odev3_2009638506.c	598	0	7	2
odev3_2009639005.out	odev3_2009639500.c	646	0	10	2
odev3_2009639005.out	odev3_2009638504.c	617	0	7	4
odev3_2009639005.out	odev3_2009639504.c	713	3	12	1
odev3_2009638504.c	odev3_2009638506.c	693	0	35	94
odev3_2009638504.c	odev3_2009639504.c	676	1	11	4
odev3_2009638504.c	odev3_2009639500.c	587	0	8	1
odev3_2009638402.c	odev3_2009638504.c	612	0	9	1
odev3_2009638402.c	odev3_2009639500.c	631	0	11	4
odev3_2009638402.c	odev3_2009639504.c	698	0	7	1
odev3_2009638402.c	odev3_2009638506.c	590	0	9	1
odev3_2009638057.c	odev3_2009639012.c	451	0	9	6
odev3_2009638057.c	odev3_2009639010.c	428	0	8	1
odev3_2009638057.c	odev3_2009639061.c	428	0	8	1
odev3_2009638057.c	odev3_2009639006.c	461	0	7	1

Figure 4.49. Numbers of Matched Tokens For Scenario 6

The process of this hybrid algorithm takes 2302 seconds as shown in Figure 4.50. Source Code Matching Algorithm takes 2269 seconds. Comment Line Matching, Semantic Sequence Matching, Word Matching Algorithms and Tokenization take 15 seconds. The Match Score process takes 18 seconds. So all algorithms and processes take 2302 seconds.

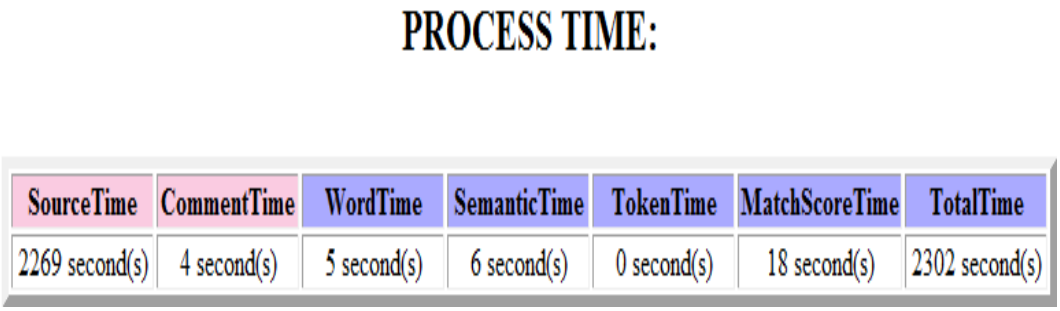


Figure 4.50. Process Time of This Hybrid Algorithm For Scenario 6

JPlag, Moss and this Hybrid Algorithm are compared with respect to similarity rate in Table 4.17. There are 300 matches in this scenario. The random ten matches are selected from the Scenario 6 assignments. As shown in the table, our hybrid algorithm is more successful than Moss and JPlag.

Table 4.17. The Comparison of the Similarity Rate Computed By Moss JPlag and Hybrid For Scenario 6

FILE1	**FILE2**	**MOSS**	**JPLAG**	**HYBRID**
odev3_2009639010.c	odev3_2009639061.c	99 %	100.00 %	**100.00 %**
odev3_2008639012.c	odev3_2009639018.c	7 %	64.1 %	**68.79 %**
odev3_2008639005.c	odev3_2009639006.c	0 %	65.8 %	**69.16 %**
odev3_2008638057.c	odev3_2009638050.c	10 %	52.6 %	**64.10 %**
odev3_2009638045.c	odev3_2009638048.c	75 %	**95.1 %**	91.14 %
odev3_2009638006.c	odev3_2009638033.c	8 %	34.7 %	**61.21 %**
odev3_2008638001.c	odev3_2008638017.c	97 %	**100.00 %**	99.76 %
odev3_2009638050.c	odev3_2009638048.c	18 %	65.2 %	**73.14 %**
odev3_2009638504.c	odev3_2009638506.c	91 %	**99.3 %**	97.88 %
odev3_2009639500.c	odev3_2009639504.c	4 %	49.8 %	**63.03 %**

The process time of Moss, JPlag and Hybrid Algorithm are compared in a Table 4.18. As shown in the table, our hybrid algorithm is the slowest.

Table 4.18. The Comparison of the Process Time For Scenario 3

MOSS	**JPLAG**	**HYBRID**
36 seconds	71 seconds	2302 seconds

4.7. The Seventh Scenario

The seventh scenario is tested that the weight coefficients, α_1, α_2, α_3 and α_4, affect the match score to what extent. α_1 is for Word Matching Algorithm. α_2 is for Semantic Sequence Matching Algorithm. α_3 is for Greedy String Tiling Based Source Lines Matching Algorithm. α_4 is for Greedy String Tiling Based Comment Lines Matching Algorithm. Two source files are compared in this scenario. These files are oedev1_2008638008.c and oedev1_2008638026.c. They have 88.76% similarity rate of source code matching, 0% similarity rate of comment line matching, 76.92% similarity rate of word matching and 12.24% similarity rate semantic sequence matching. If the weight coefficients change, the match score changes, too.

The new match scores according to the changed weight coefficients are showed in Table 4.19. The last coefficients give the best result, because we observed that students do not use comment often. So, as the comment matching rate is lower, the result is better. When source matching rate is higher, the result is better, too. That's why α_1 is selected as 0.178; α_2 is selected as 0.01; α_3 is selected as 0.8 and α_4 is selected as 0.01.

Table 4.19. According To The Changes In The Weight Coefficients

α_1	**α_2**	**α_3**	**α_4**	**MatchScore**
0,1	0,03	0,77	0,1	76.40
0,15	0,01	0,79	0,05	81.78
0,2	0,03	0,75	0,02	82.32
0,18	0,01	0,8	0,01	84.98
0,178	0,01	0,802	0,01	85.00

4.8. The Eighth Scenario

When creating this algorithm, some threshold values are used, because without using threshold values the result cannot be accurate. This eighth scenario is tested the match score results with and without using threshold. Four threshold values are used to take an optimum result.

Figure 4.51 shows the pseudocode of hybrid algorithm with using threshold values.

```
if ( source>=95 ){
    matchscore=(source*1) + (comment*0) +(word*0) +(semantic*0);
}
else{
    if ( word>source ){
        if ( ( comment<=10) || (semantic<=10) ){
            matchscore=(source*0.25) + (comment*0) + (word*0.75)
                    + (semantic*0);
        }
        else{
            matchscore=(source*0.85) + (comment*0.015) +
                    (word*0.12) + (semantic*0.015);
        }
    }
    elsif ( comment>source ){
        if ( semantic<=10 ){
            matchscore=(source*0.2) + (comment*0,71) +(word*0.09)
                    + (semantic*0);
        }
        else{
            matchscore=(source*0.85) + (comment*0.015) +
                    (word*0.12) + (semantic*0.015);
        }
    }
    else{
        matchscore=(source*0.9) + (comment*0.001) + (word*0.097) +
                (semantic*0.002);
        }
}
```

Figure 4.51. The Pseudocode of Hybrid Algorithm With Using Threshold Values

The first threshold is that if source code matching similarity rate is higher than 95%, then the match score is calculated according to α_3 weight coefficient, which is Greedy String Tiling Based Source Lines Matching Algorithm coefficient. In this case, α_3 becomes equal to 1.

The second threshold is that if word matching similarity rate is bigger than source code matching similarity rate, then the match score is calculated according to α_1 weight coefficient. If the similarity rates of the comment matching algorithm and semantic sequence matching algorithm are less than 10%, then α_1 equals to 0.75 and α_3 equals to 0.25. If the similarity rates of the comment matching algorithm and semantic sequence matching algorithm are bigger than 10%, then α_1 equals to 0.12; α_2 equals to 0.015; α_3 equals to 0.85 and α_4 equals to 0.015.

The third threshold is that if comment matching similarity rate is bigger than source code matching similarity rate, then the match score is calculated according to α_4 weight coefficient. If the similarity rate of the semantic sequence matching algorithm is less than 10%, then α_1 equals to 0.09; α_3 equals to 0.2 and α_4 equals to 0.71. If the similarity rate of the semantic sequence matching algorithm is bigger than 10%, then α_1 equals to 0.12; α_2 equals to 0.015; α_3 equals to 0.85 and α_4 equals to 0.015.

The last threshold is that if source code matching similarity rate is less than 95% similarity rate, then the match score is calculated according to α_1, α_2, α_3 and α_4 weight coefficients. So α_1 equals to 0.097; α_2 equals to 0.002; α_3 equals to 0.9 and α_4 equals to 0.001.

```
if ( ( source<=50 ) || ( comment<=10 ) || ( word<=10) ||
   (semantic<=10) ){
matchscore=(source*0.802) + (comment*0.01) +(word*0.178) +
                (semantic*0.01);
}
else{
matchscore =(source*0.8) + (comment*0.02) +(word*0.15) +
                 (semantic*0.03);
}
```

Figure 4.52. The Pseudocode of Hybrid Algorithm Without Using Threshold Values

According to Figure 4.52, for without using threshold values, if the similarity rates of the comment matching, word matching and semantic sequence matching algorithms are less than 10% and if the source matching algorithm is less than 50%, then α_1 equals to 0,178; α_2 equals to 0,01; α_3 equals to 0,802 and α_4 equals to 0,01. In other cases, α_1 equals to 0,15; α_2 equals to 0,03; α_3 equals to 0,8 and α_4 equals to 0,02.

The affects of these threshold values are showed in Table 4.20. According to Table 4.20, the similarity scores computed by using the threshold values become more meaningful.

If the similarity rate of the comment matching is higher than 0%, then the match score increases, but if the similarity rate of the word matching algorithm is higher than the similarity rate of source code matching algorithm and the similarity rate of the comment matching is higher than 0%, then the match score decreases a little.

Table 4.20. The Affects of Threshold Values

Files	Source	Comment	Word	Semantic	Without Treshold	With Treshold
oedev1_2008638008.c oedev1_2008638018.c	100.00	0.00	91.67	100.00	100%	100%
oedev1_2008638008.c oedev1_2009638060.c	95.26	0.00	84.62	2.04	95.26%	95.26%
oedev1_2008638008.c odev1_2008638025.c	64.43	0.00	53.85	2.04	61.28%	63.21%
oedev1_2008638018.c odev1_2009638066.c	81.05	0.00	44.44	2.33	72.94%	77.26%
oedev1_2008638018.c odev1_2008638025.c	64.43	0.00	66.67	2.04	63.56%	66.00%
oedev1_2008638026.c oedev1_2009638060.c	93.71	0.00	78.57	1.85	89.16%	91.96%
odev1_2008638025.c 2008639022.c	48.01	0.00	83.33	3.45	53.37%	72.73 %
odev1_2008638031.c odev1_2008639007.c	61.50	0.00	66.67	3.33	61.22%	65.12%
oedev1_2009638060.c odev1_2009639027.c	66.67	0.00	80.00	2.78	67.74%	76.00%
odev1_2009638048.c odev1_2008639007.c	61.50	0.00	66.67	3.33	61.22%	65.12%

4.9. The Ninth Scenario

In the ninth scenario, the effect of using our online C compiler on grading is tested. For this purpose, 57 source files which are different submissions of the same assignment compiled on Moodle using our online GCC compiler. The time needed for the compiling process is compared with the time of the compiling submissions by lecturers (i.e., without using our online compiler).

One assignment takes 50 seconds with online compiling on Moodle. The same assignment takes at least 5 minutes if our online compiler is not used since in this case, lecturer first downloads the source code, then opens the code and compiles using any C compiler. That's why all 57 submissions take 2850 seconds, namely 47.5 minutes with compiling on Moodle. On the other hand all 57 submissions take 17100 seconds, namely 285 minutes with manual compilation. It corresponds to 2 hours 45 minutes. All calculations of reading, controlling and compiling assignments by lecturers are by assuming that they do this process without intervention. All the same calculations are done on Moodle that lecturers compile the assignments without a break. Because sometimes lecturers get tired and have a break to read and compile the assignments. So the reading of assignments takes days in some cases. In these calculations this condition does not take into consideration. The comparison result is showed in Table 4.21.

Table 4.21. The Comparison Time of Compiled Assignments

	Reading, Controlling and Compiling Assignments By Lecturers Manually	**Reading, Controlling and Compiling Assignments With GCC On Moodle**
TIME	2 hours 45 minutes	47.5 minutes

4.10. The Tenth Scenario

In the previous scenarios, students of Data Structures course did not know that a plagiarism detection module was installed on Moodle to detect the similarities between the assignments. But for the last assignment of Data

Structures students know that a plagiarism detection module was installed on Moodle. The tenth scenario tested the effect of plagiarism detection module on the plagiarism behavior of students.

Before they learn the existence of this module, most students plagiarize from each other. The assignments which has 100 % similarity rates are plagiarized by 22.22 % of the students. The assignments which has between 76 % and 99 % similarity rates are plagiarized by 26.67 % of the students. The assignments which has between 51 % and 75 % similarity rates are plagiarized by 51.11 % of the students. The assignments which has between 26 and 50 % similarity rates are plagiarized by 0 % of the students. The assignments which has between 0 and 25 % similarity rates are plagiarized by 0 % of the students.

After they learn the existence of this module, plagiarism rates are decreased. The assignments which has 100 % similarity rates are plagiarized by 5.23 % of the students. The assignments which has between 76 and 99 % similarity rates are plagiarized by 14.38 % of the students. The assignments which has between 51 and 75 % similarity rates are plagiarized by 69.28 % of the students. The assignments which has between 26 and 50 % similarity rates are plagiarized by 11.11 % of the students. The assignments which has between 0 and 25 % similarity rates are plagiarized by 0 % of the students. According to these results, it can be said that the detection of plagiarism has a deterrent effect on students.

The comparison result is showed in Table 4.22.

Table 4.22. The Plagiarism Information of Assignments

Matchscore	**Before Plagiarism Information**		**After Plagiarism Information**	
100	10 Matches	22.22 %	8 Matches	5.23 %
76-99	12 Matches	26.67 %	22 Matches	14.38 %
51-75	23 Matches	51.11 %	106 Matches	69.28 %
26-50	0 Matches	0 %	17 Matches	11.11 %
0-25	0 Matches	0 %	0 Matches	0 %
TOTAL	45 MATCHES		153 MATCHES	

5. CONCLUSION

In this thesis an online C Compiler and a plagiarism detection algorithm for C programming language assignments are developed. Such an online compiler and a plagiarism detection module are not available for Moodle system. That's why these additions are implemented in this thesis.

This online compiler reduces the time of assignments assessment. Our plagiarism detection algorithm is a hybrid string matching algorithm which consists of Greedy String Tiling, Source Line Matching, Comment Line Matching, Word Matching and Semantic Sequence Matching Algorithms and it is included into Moodle distance education system. In our hybrid algorithm, all the matching algorithms produce a match score and each match score is weighted and the Total Match Score is calculated to give the optimal results. Experimental evaluation shows that, the proposed algorithm is effective on detecting similarities.

We observe that with respect to processing time, our algorithm is fast for short and small number of assignments but it gives more successful and accurate results on short assignments up to 300 lines. For long assignments which have more than 300 lines, our algorithm gives successful results with respect to similarity scores, but the processing time is long when compared to Moss and JPlag.

As future work, we plan to study for reducing of the processing time of our hybrid plagiarism detection algorithm so that we can use it effectively. Algorithms, which are written in this study, can be weighted with respect to their importance, and this method can improve the performance of our hybrid algorithm. We plan to add a choice for programming language to our algorithm, because it runs only for C programming language assignments. This hybrid algorithm can be easily used to detect plagiarism on the programming assignments that are written in other programming languages, and this helps to increase the quality of the computer engineering education by forcing the students to make their own homeworks.

REFERENCES

AHTIAINEN, A., SURAKKA, S. and RAHIKAINEN, M., 2007. Plaggie: GNU-licensed Source Code Plagiarism Detection Engine for Java Exercises, In Proceedings of the 6th Baltic Sea Conference on Computing Education Research, Koli Calling, pp. 141-142.

AIKEN A., 1998. MOSS (Measure Of Software Similarity) Plagiarism Detection System, http://www.cs.berkeley.edu/?moss/ (as of April 2000) and Personal Communication, 1998. University of Berkeley, CA.

AJLAN, A. A. and ZEDAN, H., 2007. Why Moodle, 12th IEEE International Workshop on Future Trends of Distributed Computing Systems.

AL, U. and MADRAN R. O., 2004, Web Tabanlı Uzaktan Eğitim Sistemleri: Sahip Olması Gereken Özellikler ve Standartlar.

ALZAHRANI, S., SALIM, N., ABRAHAM, A., 2011. Understanding Plagiarism Linguistic Patterns, Textual Features and Detection Methods, IEEE Transactions On Systems, Man, And Cybernetics—Part C: Applications And Reviews, Vol. Xx, No. Xx, Mmm 2011.

APACHE, 2011. http://www.apache.org.

ARWIN, C. and TAHAGHOGHI, S. M. M., 2006. Plagiarism Detection Across Programming Languages, In Proceedings of the 29th Australasian Computer Science Conference, pp. 277-286.

BARKER, B., FRISBIE, A. and PATRICK, K. (1989). Broadening The Definition of Distance Education In The Light Of The New Telecommunications Technologies, The American Journal of Distance Education, 3(1), 20-29.

BELKHOUCHE, B., NIX, A. and HASSELL, J., 2004. Plagiarism Detection In Software Designs. In Proceedings of the 42nd Annual Southeast Regional Conference, pp. 207-211.

BERGHEL, H. L. and SALLACH, D. L., 1984. Measurements of Program Similarity in Identical Task Environments. ACM SIGPLAN Notices, 19(8):65–76.

BIRNBAUM, B.W., 2001. Foundations and Practices In The Use of Distance Education. Mellen Studies in Education, Vol. 66, pp. 1-174, Lewiston: The Edwin Mellen Press.

BOWYER, K. W. and HALL, L. O., 1999. Experience Using ”MOSS” to Detect Cheating On Programming Assignments, fie, vol. 3, pp.13B3/18-13B3/22vol.3, Frontiers in Education Conference, 1999. FIE '99. 29th Annual.

BULL, J., COLLINS, C., COUGHLIN, E. and SHARP, D., 2001. Technical Review of Plagiarism Detection Software Report, CAA, University of Luton and JISC Plagiarism Advisory Service.

BURROWS S., TAHAGHOGHI S. M. M. and ZOBEL J., 2006. Efficient Plagiarism Detection for Large Code Repositories. School of Computer Science and Information Technology, RMIT University, Melbourne, Australia, pp 158-159.

CARBAJOSA, A. D. and BAUSELA, J. F., 2010. Using Plagiarism Detection Systems to avoid Research Pathologies in Computer Science – University of Valladolid MIN Conference 2010 – 15 Nov 2010.

CARNEGIE COMMISSION, 1972. The Fourth Revolution: Instructional Technology in Higher Education, New York, McGraw-Hill.

CHEN, X., FRANCIA, B., LI, M., MCKINNON, B. and SEKER, A., 2003. Shared Information and Program Plagiarism Detection**,** IEEE Transactions on Information Theory, vol. 50, pp. 1545-1551

CHESTER, G., 2001. Pilot of Free-text Electronic Plagiarism Detection Software, Joint Information Systems Committee.

CHEUNG, K. S., 2006. A Comparison Of WEBCT, BLACKBOARD And MOODLE For The Teaching And Learning Of Continuing Education Courses, International Conference On ICT In Teaching And Learning, Enhancing Learning Through Technology, pp 219-228.

CLOUGH, P., 2000. Plagiarism in Natural and Programming Languages: an Overview of Current Tools and Technologies, University of Sheffield.

CODEMATCH, 2011. http://www.safe-corp.biz/products_codematch.htm.

COLE, J., 2005. Using Moodle: Teaching With The Popular Open Source Course Management System, O'Reilly Community Press, Sebastopol, CA.

COLE J. R. and FOSTER H., 2008. Using Moodle: Teaching with the Popular Open Source Course Management System, 2nd Edition, O'Reilly Community Press, p. ix.

COSMA G., JOY M., 2006. Source-code Plagiarism: A UK Academic Perspective. Research Report No. 422. Department of Computer Science, University of Warwick. http://www.dcs.warwick.ac.uk/reports/422.html.

CPD, 2011. http://pmd.sourceforge.net/cpd.html.

CULWIN, F., MACLEOD, A. and LANCASTER T., 2001. Source code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools, Technical Report No. SBU-CISM-01-02, South Bank University.

DELLING, R. M., 1966. Versuch Der Grundlegung Zu Einer Systematischen Theorie Des Fernunterrichts, in L. Sroka (Ed.). Fernunterricht, Hamburg: Hamburger Fernlehrinstitut.

DIFFERENCES BETWEEN MOODLE AND WEBCT (BLACKBOARD), 2011. http://skillspark.ca/info/MoodleandWebCTComparison.pdf.

DISTANCE LEARNING IN HIGHER EDUCATION, 2011. http://education.stateuniversity.com/pages/1917/Distance-Learning-in-Higher-Education.html.

DOHMEN, G., 1967. Das Fernstudium, Ein neues padagogisches Forschungsund Arbeitsfeld, Tubingen: DIFF.

DUZER, J. V. And MUNOZ K., 2005. A Comparison of Satisfaction with Online Teaching and Learning Tools, Blackboard vs. Moodle.

ELLIS, M. G., ANDERSON, C. W., 2005. Plagiarism Detection in Computer Code, www.rose-hulman.edu/class/cs/faculty-staff/csse-department/seniorTheses/Matt%2520Ellis.pdf.

FAIDHI, J. A. W. and ROBINSON, S. K., 1987. An Empirical Approach For Detecting Program Similarity Within A University Programming Environment. Computers and Education, 11(1):11–19.

FINKEL, R. A., ZASLAVSKY, A., MONOSTORI K. and SCHMIDT H., 2002. Signature Extraction for Overlap Detection in Documents, Monash University, Australia.

GARRISON, D. and SHALE, D., 1987. Mapping The Boundaries of Distance Education: Problems In Defining The Field, The American Journal of Distance Education, 1(1), 4-13.

GITCHELL, D. and TRAN, N., 1999. Sim: A Utility for Detecting Similarity in Computer Programs, In the Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education, pp. 266-270.

GOMES L., 2006. Some Students Use Net To Hire Experts to Do Their School Work. The Wall Street Journal [Online], http://online.wsj.com/public/us.

GÜNDÜZ M., 2000. Uzaktan Eğitim, Yüksek Lisans Tezi, Konya.

GÜNDÜZ M., BAYKAN Ö.K. and YILDIZ F., 2007. "Elektronik Deneyleri İçin Sanal Laboratuar Uygulaması", Journal of Technical-Online, Volume 6, Number: 2.

HACKER, D., 2007. A Writer's Reference, 6th Edition, Boston: Bedford/St. Martin's.

HACKER, D., 2008. A Pocket Style Manual (5 ed.). Boston: Bedford/St. Martin's. p. 107. ISBN 0312559933.

HARRISON, J. C., 2011. Different Types of Distance Education, http://ezinearticles.com/?Different-Types-of-Distance-Education&id=6073038.

HECKEL, P., 1978. A Technique For Isolating Differences Between Files, Communication of the ACM, Volume: 21, Number: 4.

HOAD, T. C. and ZOBEL J., 2003. Methods for Identifying Versioned and Plagiarised Documents. Journal of the American Society for Information Science and Technology 54(3) pp. 203-215.

HOLMBERG, B., 1977. Distance Education: A Survey and Bibliography. London: Kogan Page.

HOLMBERG, B., 2005. The Evolution, Principles And Practices of Distance Education, Studien und Berichte der Arbeitsstelle Fernstudienforschung

der Carl von Ossietzky Universität Oldenburg [ASF], 11. Bibliotheks-und Informationssystem der Universitat Oldenburg, p. 13, ISBN 3814209338, Retrieved 23 January 2011.

HONEYMAN, M. and MILLER, G., 1993. Agriculture Distance Education: A Valid Alternative For Higher Education, Proceedings of the 20th Annual National Agricultural Education Research Meeting: 67–73.

HONOR COUNCIL, 2011. http://orgs.odu.edu/hc/pages/What_is_the_Honor_Council.shtml.

HUSTON, C. E., 2008, Fingerprinting Jar Files Using Winnowing, http://www.catehuston.com/files/fingerprinting%20jar%20files%20using%20winnowing.pdf.

JEFFRIES, M., 2008. Research in Distance Education, http://www.digitalschool.net/edu/DL_history_mJeffries.html.

JPLAG, 2011. https://www.ipd.uni-karlsruhe.de/jplag/.

KARP R. M. and RABIN M. O.. 1987. Efficient Randomized Pattern Matching Algorithms. IBM J. of Research and Development, 31(2):249-260, March 1987.

KEEGAN, D., 1996. Foundations of Distance Education , 3rd ed., London: Routledge.

LIAQAT, A. G. and AIJAZ, A., 2011. Plagiarism Detection in Java Code, Linnaeus University, Faculty of Science and Engineering, School of Computer Science, Physics and Mathematics, Student Thesis.

LIU, C., CHEN, C., HAN, J. and YU, P. S., 2006. GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis, In the Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06), pages 872--881.

LONG-DISTANCE TEACHING, 1990. The Futurist, 24, 48.

MANN, S. and FREW, Z., 2006. Similarity and Originality in Code: Plagiarism and Normal Variation in Student Assignments, In Proceedings of the 8th Australasian Conference on Computing Education, pp. 143-150.

MEHROTRA, C. M., HOLLISTER, C.D., and MCGAHEY, L. (2001). Distance Learning: Principles for Effective Design, Delivery, and Evaluation, Thousand Oaks, CA: Sage Publications, Inc.

MISHNE, G., 2003. Source Code Retrieval using Conceptual Graphs, Master of Logic Thesis, Institute for Logic, Language and Computation (ILLC) University of Amsterdam Plantage Muidergracht 24, 1018 TV Amsterdam The Netherlands.

MISHNE, G. And RIJKE, M., 2004. Source Code Retrieval Using Conceptual Similarity, In Proceeding of the 2004 Conference on Computer Assisted Information Retrieval (RIAO'04), pp. 539-554, Avignon (Vaucluse), France.

MONOSTORI, K., ZASLAVSKY, A. and BIA, A., 2001. Using the MatchDetectReveal System for Comparative Analysis of Texts Monash University, Australia.

MOODLE, 2011. http://moodle.org/.

MOORE, M., 1973. Toward a Theory of Independent Learning and Teaching, Journal of Higher Education, 44, 661-679.

MOORE, M., 1977. On a Theory of Independent Study, Hagen: Fernuniversitat (ZIFF).

MOORE, M., 1990. Background and Overview of Contemporary American Distance Education, In M. Moore (Ed.) Contemporary Issues In American Distance Education (pp. xii-xxvi). New York: Pergamon.

MOORE, M. and KEARSLEY, G., 1996. Distance Education: A Systems View, Belmont, CA: Wadsworth.

MOSS, 2011. http://theory.stanford.edu/~aiken/moss/.

MUGRIDGE, I., 1991. Distance Education and The Teaching Of Science, Impact of Science on Society 41 4, 313-320.

MYSQL, 2011. www.mysql.com.

NIEZGODA, S. and WAY, T. P., 2006. SNITCH: A Software Tool for Detecting Cut and Paste Plagiarism, In Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '06), pp. 51-55.

NORTH, D., 1993. Have video won't travel, Canadian Business, 68 4, 43.

NOZAWA, K., 2011. To Moodle or Not To Moodle: Can It Be an Ideal e-Learning Environment?, Ritsumeikan University Journal of Policy Science, Bulletin of Universities and Institutes , 289-312.

ODABAŞ, H., 2004. İnternet Tabanlı Uzaktan Öğrenim Modelinin Bilgi Hizmetlerine Yönelik Yüksek Öğretim Programlarında Kullanımı, Kütüphaneciliğin Destanı Uluslararası Sempozyumu: Saga of Librarianship International Symposium,Ankara (Turkey),21-24 October 2004, Ankara Üniversitesi Dil ve Tarih-Coğrafya Fakültesi. pp.121-139. (Published) [Conference Paper].

PARAPAR, J., BARREIRO, A., 2008. Winnowing Text Based Clustering, CIKM'08, October 26–30, 2008, Napa Valley, California, USA, ACM 978-1-59593-991-3/08/10.

PERRATON, H. (1988). A Theory For Distance Education, In D. Stewart, D. Keegan, & B. Holmberg (Ed.) Distance Education: International Perspectives (pp. 34-45). New York: Routledge.

PETERS, O., 1973. Die Didaktische Struktur des Fernunterrichts, Weinheim: Beltz.

PHP, 2011. www.php.net/.

PICCIANO, A. (2001). Distance Learning: Making Connections Across Virtual Space And Time, Upper Saddle River, NJ: Merrill Prentice Hall.

PLAGIARISM, 2011. http://www.plagiarism.org/plag_article_what_is_plagiarism.html.

PLAGIARISM TUTORIAL, 2011. www.mtlsd.org/highschool/stuff/10%20plagiarism%20tutorial.ppt.

PORTWAY, P. and LANE, C. (Eds.), 1994. Guide To Teleconferencing and Distance Learning, San Ramon, California: Applied Business Communications.

PRECHELT L., MALPOHL G., and PHILIPPSEN M., 2000, JPlag: Finding Plagiarisms Among A Set Of Programs. Technical Report 20001, Fakultät für Informatik, Universität Karlsruhe, Germany, March 2000. ftp.ira.uka.de.

PRECHELT, L., MALPOHL, G. and PHILIPPSEN, M., 2002. Finding Plagiarisms among a Set of Programs with JPlag. Journal of Universal Computer Science, Vol. 8, No. 11.

QWHATIS.COM, 2011. http://www.qwhatis.com/what-is-gcc/.

REDSHIELD, D., 2011. Why Plagiarism Happens, http://ezinearticles.com/?Why-Plagiarism-Happens---Top-5-Reasons&id=2692125.

REISER, R. A., 1987. Instructional Technology: A History, In R.M. Gagne (Ed.) Instructional Technology: Foundations (Pp. 11-48), Hillsdale, NJ: Lawrence Erlbaum Associates.

SAETTLER, P., 1968. A History of Instructional Technology ,(pp 11-35), New York: Mcgraw-Hill.

SCHLEIMER, S., WILKERSON, D. S. and AIKEN, A., 2003. Winnowing: Local Algorithms For Document Fingerprinting, In Proceedings ACM SIGMOD International Conference On Management of Data, pp. 76-85.

SHAW, M., 1980, Cheating policy in a Computer Science Department. SIGCSE Bulletin 12, 2 (July 1980), 72-76.

SHERLOCK, 2011. http://sydney.edu.au/engineering/it/~scilect/sherlock/.

SID, 2011. http://genome.math.uwaterloo.ca/SID/.

SON, J.-W., PARK, S.-B., and PARK, S.-Y, 2006. Program Plagiarism Detection Using Parse Tree Kernels, In Proceedings of the 9th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2006), Volume 4099 of Lecture Notes in Computer Science, Pages 1000–1004.

THE INSTITUTE FOR HIGHER EDUCATION POLICY, 2000. Quality On The Line: Benchmarks For Success In Internet-Based Distance Education, Page: 15.

VAMPLEW, P. and DERMOUDY, J., 2005. An Anti-plagiarism Editor for Software Development Courses, In Proceedings of the 7th Australasian Conference on Computing Education pp. 83-90.

VERCO, K. and WISE, M. J., 1996. Software for Detecting Suspected Plagiarism: Comparing Structure and Attribute-Counting Systems, In Proceedings of

the 1st Australasian conference on Computer Science Education, pp. 81-88.

WAGNER, N. R., 2010. Plagiarism by Student Programmers, The University of Texas at San Antonio. Division Computer Science. San Antonio, TX 78249, USA.

WHITE, D. and JOY, M., 2004. Sentence-based Natural Language Plagiarism Detection, Journal on Educational Resources in Computing (JERIC) 4(4).

WIEDEMEIER, P. D., 2002. Preventing Plagiarism in Computer Literacy Courses, Journal of Computing Sciences in Colleges, 17(4), pp. 154-163.

WISE, M. J., 1992. Detection of Similarities In Student Programs: YAP'ing May Be Preferable to Plague'ing, ACM SIGSCE Bulletin (Proc. of 23rd SIGCSE Technical Symp.), 24(1):268-271, March 1992.

WISE, M. J., 1993. Running KarpRabin Matching and Greedy String Tiling, Technical Report Number 463, Dept. of CS, University of Sydney, March 1993.

WISE, M. J., 1993. String Similarity via Greedy String Tiling and Running Karp-Rabin Matching, ftp://ftp.cs.su.oz.au/michaelw/doc/RKR_GST.ps, Dept. of CS, University of Sydney, December 1993.

WISE, M. J., 1996. 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.

YAP, 2011. http://pam1.bcs.uwa.edu.au/~michaelw/YAP.html.

ZAINA, L. A. M., BRESSAN, G., SILVEIRA R. M., STIUBIENER, I. and RUGGIERO, W. V., 2001. Analysis and Comparison of Distance Education Environments, International Conference on Engineering Education, Oslo, Norway.

ZEIDMAN, B., 2004. Detecting Source-Code Plagiarism - Tools and Algorithms for Finding Plagiarism in Source Code, Dr Dobb's Journal, http://drdobbs.com/architecture-and-design/184405734.

ZEIDMAN, B., 2004. Detecting Source Code Plagiarism With CodeMatch, Zeidman Consulting, http://www.b2gmarket.com/vendorfiles/Detecting%20Source%20Code%20Plagiarism%20with%20CodeMatch.pdf.

ZEIDMAN, R. M., 2010, Detecting Plagiarism In Computer Source Code, United States Patent Application 20100325614 A1.

CURRICULUM VITAE

She was born in Adana, in 08.11.1986. She has completed her elementary education in Private Adana College. She went to intermediate school and high school at İsmail Safa Özler German Anatolian High School. After these education ended, she achieved Çukurova University Faculty of Engineering and Architecture Department of Computer Engineering in 2004. First year, she get English preparatory course in YADİM. Then she began to her undergraduate education in 2005. She has completed her undergraduate education at Çukurova University Faculty of Engineering and Architecture Department of Computer Engineering in 2009 and she get Computer Engineer title. She began to her graduate education in 2009 at Çukurova University Institute of Natural and Applied Sciences Department of Computer Engineering Computer Sciences Field. She gets to work as a research assistant at Osmaniye Korkut Ata University Faculty of Engineering Department of Computer Engineering in Osmaniye in 2010 and she has been working at the same department currently.