



ugr

Universidad  
de Granada

TRABAJO FIN DE GRADO  
INGENIERÍA EN INFORMÁTICA

# DETECCIÓN DE PLAGIO EN R

---

**Autor**

Antonio Javier Rodríguez Pérez

**Directora**

Rocío Celeste Romero Zaliz



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

Granada, Septiembre de 2018







# Detección de Plagio en R

---

**Autor**

Antonio Javier Rodríguez Pérez

**Directora**

Rocío Celeste Romero Zaliz



# Detección de Plagio en R

Antonio Javier Rodríguez Pérez

**Palabras clave:** detección de plagio, analizador léxico, analizador sintáctico, n-grama, algoritmo String Tiling

## Resumen

El plagio en código fuente y su detección es un problema que encontramos con gran frecuencia en la actualidad dado que hay una gran cantidad de software nuevo y programas en general que se crean cada día.

Este problema es incluso más preponderante en el ámbito educativo, ya que durante los últimos años se ha convertido en una práctica cada vez más habitual entre los estudiantes.

Es por esto que es importante que tengamos alguna forma de saber si estos programas son genuinos o tan solo una copia o una modificación de algo ya existente.

Para detectarlo existen diversas herramientas, pero no existe ninguna creada específicamente para el lenguaje R.

En este trabajo se busca resolver este problema adaptando una de las herramientas disponibles llamada JPLAG para que funcione también con este lenguaje.

JPLAG permite determinar con mayor exactitud si existe plagio entre un conjunto de archivos de un mismo lenguaje que se le proporcione.

Modificar JPLAG para que sirva con R permitirá que la detección de plagio entre programas escritos en este lenguaje sea más rápida y eficiente que con cualquier otro método ya existente.





# Plagiarism Detection in R

Antonio Javier Rodríguez Pérez

**Keywords:** Plagiarism Detection, Parser, Lexer, n-gram, String Tiling Algorithm

## Abstract

Plagiarism in source code and its detection is a problem we frequently find nowadays due to the immense amount of new software and programs in general that are made everyday.

This problem is even more prevalent in the educational field, since in recent years it has become an increasingly common practice among students.

For those reasons it's important to have a way of knowing whether that software is genuine or its just a copy or modification of something already existing.

In order to detect that plagiarism there are many tools available, but there is not yet one created specifically for the language R.

In this project we try to solve this issue by adapting one of the available tools called JPLAG so that it'll also function with R.

JPLAG allows us to determine with great precision whether there is plagiarism among a set of files written in the same language, that you supply JPLAG with.

If JPLAG is modified so that it also detects R files, we will be able to detect plagiarism among programs in this language in a faster and more efficient way than any other existing method.



---

Yo, **Antonio Javier Rodríguez Pérez**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 50643240T, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Antonio Javier Rodríguez Pérez

Granada a 2 de Septiembre de 2018 .



---

Da. **Rocío Celeste Romero Zaliz**, Profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

**Informa:**

Que el presente trabajo, titulado ***DETECCIÓN DE PLAGIO EN R***, ha sido realizado bajo su supervisión por **Antonio Javier Rodríguez Pérez**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 2 de Septiembre de 2018 .

**La directora:**

**Rocío Celeste Romero Zaliz**



# Agradecimientos

Gracias a mi madre y a mi padre, Josefa y Antonio, por insistir en mi educación y ayudarme siempre que lo necesitase.

A mi hermana Laura por sus recomendaciones y por estar siempre ahí.

A Rocío, mi tutora, por guiarme y aconsejarme durante la creación de este proyecto.

A mis amigos de la facultad, con los que he pasado cuatro duros años de estudio. De estos amigos debo mencionar a Javi, Sergio y Jesús ya que son con los que más tiempo he pasado y son los que definen en gran parte quien soy.

Y por último, a Alba, con ella he pasado la mayor parte del tiempo durante estos años de carrera y ha conseguido que vea el mundo de forma diferente.





# Declaración de autoría y originalidad del TFG

Yo, Antonio Javier Rodríguez Pérez, con DNI 50643240T, declaro que el presente documento ha sido realizado por mí y se basa en mi propio trabajo, a menos que se indique lo contrario. No se ha utilizado el trabajo de ninguna otra persona sin el debido reconocimiento. Todas las referencias han sido citadas y todas las fuentes de información y conjuntos de datos han sido específicamente reconocidos.

Fdo: Antonio Javier Rodríguez Pérez

Granada a 2 de Septiembre de 2018 .



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Un poco de historia . . . . .	1
1.2. Motivación . . . . .	2
1.3. Objetivos . . . . .	4
1.4. Estructura del documento . . . . .	4
<b>2. Planificación</b>	<b>7</b>
2.1. Metodología usada. . . . .	7
2.2. Reparto de objetivos. . . . .	7
2.3. Presupuesto . . . . .	9
2.3.1. Recursos materiales . . . . .	10
2.3.2. Recursos humanos . . . . .	10
2.3.3. Total . . . . .	10
<b>3. Análisis</b>	<b>11</b>
3.1. Requisitos Funcionales . . . . .	11
3.2. Requisitos No Funcionales . . . . .	12
3.3. Actores . . . . .	12
3.4. Diagrama de Casos de Uso . . . . .	13
3.5. Casos de Uso . . . . .	13
<b>4. Elección de herramientas</b>	<b>17</b>
4.1. Elección de una herramienta base . . . . .	17
4.2. MOSS . . . . .	18
4.2.1. Algoritmo usado . . . . .	18
4.2.2. Ejemplo de ejecución . . . . .	19
4.3. JPLAG . . . . .	21
4.3.1. Algoritmo usado . . . . .	22
4.3.2. Ejemplo de ejecución . . . . .	23
4.3.3. Razones de su elección . . . . .	25
<b>5. Implementación</b>	<b>27</b>
5.1. Estructura de nuestro frontend . . . . .	27
5.1.1. Apache Maven . . . . .	29

5.2. Analizador utilizado . . . . .	29
5.2.1. ANTLR . . . . .	30
5.3. Adaptación del parser a JPLAG . . . . .	33
<b>6. Ejecución y Pruebas</b>	<b>35</b>
6.1. Ejecución en MOSS . . . . .	35
6.2. Ejecución en JPLAG . . . . .	40
6.2.1. Obtención de resultados . . . . .	40
6.3. Pruebas . . . . .	44
6.3.1. Análisis de resultados . . . . .	44
<b>7. Conclusiones y trabajo futuro</b>	<b>57</b>
7.1. Conclusión . . . . .	57
7.2. Trabajo Futuro . . . . .	57
<b>A. Código</b>	<b>63</b>
A.1. RTokenConstants.java . . . . .	63
A.2. RToken.java . . . . .	64
A.3. Language.java . . . . .	65
A.4. JplagRListener.java . . . . .	67
A.5. Parser.java . . . . .	73

# Índice de figuras

1.1. Tabla comparativa de las herramientas para detección de plagio en código fuente más prominentes . . . . .	3
3.1. Actor 1 . . . . .	12
3.2. Actor 2 . . . . .	12
3.3. Diagrama de casos de uso del usuario del sistema. . . . .	13
3.4. Caso de uso 1. . . . .	13
3.5. Caso de uso 2. . . . .	14
3.6. Caso de uso 3. . . . .	14
3.7. Caso de uso 4. . . . .	15
3.8. Caso de uso 5. . . . .	15
3.9. Caso de uso 6. . . . .	16
4.1. Ejemplo de página principal de resultados (MOSS) . . . . .	20
4.2. Ejemplo de página de comparación de código de MOSS . . . . .	21
4.3. Ejemplo de tokens extraídos de código en java . . . . .	22
4.4. Ejemplo de página principal de resultados (JPLAG) . . . . .	24
4.5. Ejemplo de página de comparación de código de JPLAG . . . . .	25
5.1. Árbol de archivos del frontend creado . . . . .	28
5.2. Archivo ejemplo pom.xml . . . . .	29
5.3. Árbol de sintaxis del algoritmo de Euclides. . . . .	31
5.4. Ejemplo de lexer rules. . . . .	32
5.5. Ejemplo de parser rules. . . . .	32
6.1. Página principal de resultados de la segunda entrega con matlab como opción (MOSS) . . . . .	38
6.2. Tabla comparativa entre dos estudiantes en la segunda entrega con matlab como opción (MOSS) . . . . .	39
6.3. Parte comparativa del código fuente de dos estudiantes en la segunda entrega con matlab como opción (MOSS) . . . . .	39
6.4. Página principal de los resultados de la Entrega 2 (JPLAG) parte 1 . . . . .	41

6.5. Página principal de los resultados de la Entrega 2 (JPLAG)	
parte 2 . . . . .	42
6.6. Página de comparación de código entre dos archivos en JPLAG	
(parte de arriba) . . . . .	43
6.7. Página de comparación de código entre dos archivos en JPLAG	
(parte de abajo) . . . . .	43
6.8. Histogramas de valores de similaridad entre todos los pares	
de programas de la primera entrega. . . . .	45
6.9. 10 primeros pares de programas con mayor cantidad de código	
en común de la entrega 1 . . . . .	46
6.10. Ejemplo de plagio que sólo nuestro frontend ha encontrado	
en la primera entrega . . . . .	47
6.11. Histogramas de valores de similaridad entre todos los pares	
de programas de la segunda entrega. . . . .	48
6.12. 10 primeros pares de programas con mayor cantidad de código	
en común de la entrega 2 . . . . .	48
6.13. Histogramas de valores de similaridad entre todos los pares	
de programas de la cuarta entrega. . . . .	49
6.14. 10 primeros pares de programas con mayor cantidad de código	
en común de la entrega 4 . . . . .	49
6.15. Histogramas de valores de similaridad entre todos los pares	
de programas de la tercera entrega. . . . .	50
6.16. 10 primeros pares de programas con mayor cantidad de código	
en común de la entrega 3 . . . . .	51
6.17. Histogramas de valores de similaridad entre todos los pares	
de programas de la quinta entrega. . . . .	51
6.18. 10 primeros pares de programas con mayor cantidad de código	
en común de la entrega 5 . . . . .	52
6.19. Ejemplo de plagio entre alumnos en el que se ha cambiado el	
nombre de variables y archivos(Entrega 5) . . . . .	53
6.20. Histogramas de valores de similaridad entre todos los pares	
de programas de la sexta entrega. . . . .	53
6.21. 10 primeros pares de programas con mayor cantidad de código	
en común de la entrega 6 . . . . .	54
6.22. Histogramas de valores de similaridad entre todos los pares	
de programas de la séptima entrega. . . . .	54
6.23. 10 primeros pares de programas con mayor cantidad de código	
en común de la entrega 7 . . . . .	55

# Capítulo 1

## Introducción

### 1.1. Un poco de historia

Plagiar consiste en hacer uso del trabajo o las ideas de otra persona como si fuesen propias [1].

La palabra plagio proviene del latín "plagiarus" que significa secuestrador y "plaga": "trampa, red" [2].

Esta infracción empezó a cobrar importancia cuando las palabras e ideas empezaron a ser vistas como propiedad a través de publicaciones. Esto ocurrió a comienzos del siglo dieciocho conforme el arte y la literatura empezaron a verse como expresión de individualidad y personalidad de su autor [3].

Por esta razón en 1709 se creó la primera ley inglesa de copyright (el Estatuto de la Reina Ana) que con el tiempo fueron incorporando más y más países en su legislación, siendo Estados Unidos en 1891 el último gran país en incorporar una ley para proteger a sus autores [4].

En el ámbito académico se procede de distintas formas para penalizar el plagio entre los estudiantes dependiendo de la institución.

En universidades estadounidenses por ejemplo, las penalizaciones pueden variar dependiendo de si la copia ha sido accidental (reducción de la calificación del estudiante) o intencional. En el caso de ser intencional normalmente se procederá suspendiendo la entrega o la asignatura e incluso expulsando al alumno de la universidad.

En el caso de la Universidad de Granada según el Artículo 15 "Originalidad de los trabajos y pruebas." de la normativa de evaluación y de calificación de los estudiantes de la Universidad de Granada, en caso de detectarse plagio esto "conllevará automáticamente la calificación numérica de cero en la asignatura en la que se hubiera detectado, independientemente del resto de las calificaciones que el estudiante hubiera obtenido" [5].

En la actualidad el plagio también es un problema que ocurre en archivos en código fuente, aunque su detección es más complicada ya que, por ejemplo,

la copia se puede producir a nivel de estructura del programa. Además de esto, existen una gran cantidad de lenguajes de programación diferentes con sus características propias y con distintos niveles de abstracción, lo que dificulta aún más su detección ya que naturalmente es necesario que se tengan en cuenta las propias características del lenguaje (sus palabras reservadas, su control de flujo, cómo controlan la creación de objetos,...) a la hora de saber si dos programas escritos en el mismo lenguaje son o no similares.

Para resolver este problema existen diversas herramientas que automatizan la detección de plagio en código fuente basándose en los "tokens" del lenguaje, el número de veces que se llama a una función, la distancia entre dos secuencias de strings, etc.

Entre estas herramientas cabe destacar MOSS [6], JPLAG [7], SIM [8] y Plaggie [9].

## 1.2. Motivación

R es un lenguaje de programación de alto nivel y un entorno de software interactivo pensado para computación estadística y gráficas[10]. R está basado en S y en Scheme por lo que también es bastante similar a Matlab.

R es gratis y de código abierto, no tiene restricciones de licencia, es compatible con Linux, Windows y Macintosh y tiene más de 4800 paquetes con funciones fáciles de usar.

Por estas razones R es un lenguaje que cada vez más gente usa y se enseña en más y más universidades.

Aunque su uso se esté expandiendo, todavía no existe una forma para detectar plagio en este lenguaje de forma eficaz y eficiente, ya que ninguna de las herramientas para detección de plagio existentes son compatibles con R tal y como podemos apreciar en la tabla comparativa en la Figura 1.1.



Herramienta	Lenguajes Compatibles	Disponibilidad	Características principales	Compatible con R
MOSS	C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, ensamblador MIPS, ensamblador 8086 y HCL2.	Sólo disponible online, enviando los archivos de los que se quiere saber si ha habido plagio.	Se basa en la sintaxis del programa.  Usa n-gramas a nivel de caracteres como características de los documentos a comparar.	Sólo en modo texto plano (ASCII)
JPLAG	Java, C#, C, C++, Scheme y Python	Disponible online (aunque en una versión más antigua) y offline, descargando tú mismo el código fuente del programa y ejecutándolo en tu propio ordenador	Se basa en la estructura y la sintaxis del programa.  Usa el algoritmo string tiling.	Sólo en modo texto plano.
SIM	C, C++, Java, Pascal, Modula-2, Miranda, Lisp, y ensamblador 8086	Disponible como ejecutable o en archivos fuentes.	Tokeniza el código fuente y compara estos tokens usando algoritmos de emparejamiento de patrones	NO
Plaggie	Java	Es necesario descargar los fuentes del programa y ejecutarlo en tu ordenador	Funciona de forma similar a JPLAG.	NO

Figura 1.1: Tabla comparativa de las herramientas para detección de plagio en código fuente más prominentes

Dado que, como podemos observar, no existe ninguna herramienta directa para R, hasta el momento lo que se ha hecho es detectarlo de forma manual o usando los detectores de plagio en modo texto plano.

La detección de forma manual requiere examinar uno a uno los archivos que mandan los alumnos comparándolos entre sí buscando similitudes en el código. Esto es demasiado lento y no efectivo en caso de que haya una gran cantidad de trabajos de alumnos ya que son demasiadas posibles comparaciones como para que el evaluador pueda tener en cuenta lo que han hecho

todos los demás.

Por otra parte usar los detectores de plagio en modo texto plano, aunque sea rápido, pierde en eficacia ya que no tiene en cuenta las características propias del lenguaje y solo detectará plagio en caso de haberse hecho una copia literal.

## 1.3. Objetivos

El objetivo por tanto de este trabajo es solucionar este problema adaptando una de las herramientas para detección de plagio disponibles a R. Hemos elegido JPLAG como herramienta a adaptar dado que es por norma general la herramienta que da mejores resultados para la detección de plagio en lenguajes en los que sí es compatible y además es la más fácilmente modificable debido a que se estructura en distintos frontends de lenguajes independientes del algoritmo principal de comparación y a que podemos descargar y modificar su código fuente.

Podemos dividir este objetivo en los siguiente sub-objetivos:

- Estudiar y entender el funcionamiento de la estructura de clases de JPLAG
- Añadir un frontend para que JPLAG pueda procesar archivos en R.
- Modificar dicho frontend para que JPLAG reciba correctamente los "tokens" propios del lenguaje.
- Realizar las pruebas necesarias para verificar que se ha adaptado JPLAG correctamente a R y obtiene mejores resultados que en texto plano.

## 1.4. Estructura del documento

Este proyecto de fin de grado está estructurado en ocho capítulos que explican todo el trabajo que se ha llevado a cabo para la adaptación de JPLAG a R:

1. **Introducción:** en este capítulo se explica el problema en cuestión que queremos resolver.
2. **Planificación:** organización de mi tiempo y mis recursos para la realización de este trabajo y presupuesto.
3. **Análisis:** especificación de los requisitos funcionales y no funcionales, y los casos de uso de la tarea en cuestión.

4. **Elección de herramientas:** en este capítulo se muestran las herramientas disponibles para la detección de plagio, se explica por qué se han elegido JPLAG y MOSS y se detalla cómo funcionan éstas.
5. **Implementación:** aquí se explican las partes más importantes del desarrollo, es decir, qué ha sido necesario añadir y modificar para la creación de nuestro frontend y para que éste procese archivos en R.
6. **Ejecución y Pruebas:** instrucciones sobre cómo ejecutar JPLAG y MOSS, pruebas que se han llevado a cabo con nuestra versión nueva de JPLAG, comparándola con la versión antigua y con MOSS, y análisis de estas pruebas.
7. **Conclusiones y trabajo futuro:** conclusiones de lo hecho en el trabajo y posibles mejoras que se puedan hacer al frontend.
8. **Apéndice:** código fuente añadido a JPLAG.



## Capítulo 2

# Planificación

En este capítulo explico la metodología que se ha seguido para la realización de este trabajo, así como la organización de las tareas hecha siguiendo esta metodología.

Por último, concluimos este capítulo con el presupuesto del proyecto.

### 2.1. Metodología usada.

He usado una metodología ágil [11] basándome en SCRUM [12]: esta metodología consiste en mantener un contacto frecuente (cada 2 semanas aproximadamente) con nuestro cliente a través de reuniones para revisar el progreso conseguido en la anterior entrega y plantear los objetivos importantes para la próxima. Después de cada reunión se hace un sprint semanal o bisemanal para cumplir los objetivos planteados y se contacta en cualquier momento con el cliente si surge algún problema para replantear el sprint. Esta metodología permite que haya un seguimiento del avance mucho más completo, acepta cambios durante cualquier etapa durante el desarrollo y soluciona los errores que surgen de forma más eficaz y con menos presión ya que no hay una entrega general a largo plazo.

En mi caso mi tutora de TFG es mi cliente y las reuniones se han llevado a cabo en su despacho y online (cuando no era posible hablar en persona) cada una o dos semanas dependiendo de la cantidad de trabajo asignada al sprint de la semana actual.

### 2.2. Reparto de objetivos.

Usando la metodología mencionada en la sección anterior estos han sido las tareas que se han llevado a cabo en cada sprint desde que comencé el trabajo el 4/4/18 hasta el 27/8/18:

- **Primera reunión (4/4/18):**

- Revisión bibliográfica sobre detección de plagio.
- Revisión de software de plagio en código fuente.
- Repaso de R.
- Búsqueda de software de plagio que funcione con R.
- **Segunda reunión (18/4/18):**
  - Repaso rápido de uso de Latex.
  - Empezar la redacción del proyecto en Latex.
  - Ejecución y uso de la versión actual de JPLAG.
- **Tercera reunión (25/4/18):**
  - Especificación de requisitos.
  - Entender estructura de clases de JPLAG
- **Cuarta reunión (9/5/18):**
  - Adaptar JPLAG para que detecte y use la extensión .r.
  - Aprender cómo funciona en mayor profundidad Apache Maven para modificar correctamente el archivo pom que usa el proyecto en total.
- **Quinta reunión (16/5/18):**
  - Aprender cómo consigue los tokens de los archivos JPLAG.
  - Aprender ANTLR4.
  - Empezar a crear una gramática que define a R para ANTLR4.
- **Sexta reunión (1/6/18):**
  - Estudiar en mayor profundidad la estructura del lenguaje R con sus características específicas para poder crear correctamente su gramática.
  - Redactar progreso hasta el momento.
- **Séptima reunión (13/6/18):**
  - Crear los archivos .java dentro de nuestro frontend que comunican el parser con JPLAG.
  - Implementar un parser inicial básico.
- **Octava reunión (20/6/18):**
  - Conectar el parser creado con JPLAG de forma que reciba los tokens de los archivos que analiza.

- Elegir los tokens más relevantes del lenguaje (primer intento).
- **Novena reunión (4/7/18):**
  - Realizar prueba inicial con versión actual del frontend.
  - Redactar en la memoria los avances.
- **Décima reunión (11/7/18):**
  - Modificar la gramática para poder seleccionar tokens más importantes.
  - Elegir los tokens más relevantes del lenguaje (segundo intento).
  - Probar de nuevo JPLAG con las modificaciones hechas.
- **Undécima reunión (25/7/18):**
  - Última modificación a la gramática.
  - Elegir los tokens más relevantes del lenguaje por última vez.
  - Probar de nuevo JPLAG con las modificaciones hechas con los archivos de alumnos que obtenemos de la tutora.
  - Anonimizar los archivos de alumnos.
- **Duodécima reunión (6/8/18):**
  - Realizar los benchmarks con todas las entregas de los alumnos con nuestro frontend de R.
  - Realizar los benchmarks con todas las entregas de los alumnos con el frontend de texto plano.
  - Realizar los benchmarks con todas las entregas de los alumnos con MOSS.
- **Trigésima reunión (14/8/18):**
  - Escribir en la memoria la comparación de las pruebas realizadas.
  - Terminar de redactar la memoria y corregir errores.
- **Cuadragésima reunión (27/8/18):**
  - Últimas mejoras y corrección de errores en la memoria.

## 2.3. Presupuesto

En este apartado planteamos un análisis del valor de los recursos usados para el desarrollo de este proyecto.

### 2.3.1. Recursos materiales

Aunque las herramientas que hemos estado manipulando para la detección de plagio son gratuitas, hemos hecho uso de una máquina local para realizar todas las pruebas, buscar información y redactar la memoria. Se trata de un ordenador de sobremesa con las siguientes especificaciones:

- Procesador: Intel i5 4670K (200 €)
- Ram: 8 GB DDR3 (75 €)
- Almacenamiento: HDD 1 TB 7200 rpm (50 €)
- F. alimentación: LC-Power LC600H-12 (44 €)
- Placa base: MSI H81M-P33 (60 €)
- Caja: Aerocool V3XAD (30 €)
- Sistema Operativo: Windows 10 (proporcionado por la Universidad de Granada)

Esto asciende a un total de 459 €, aunque a esto hay que añadirle gastos como la electricidad, el material de papelería usado y la conexión a internet.

### 2.3.2. Recursos humanos

Teniendo en cuenta que ha sido un trabajo de unos cinco meses de duración con 4 horas de media de trabajo diario 5 días a la semana, tenemos un total de 400 horas de trabajo.

El salario medio de un ingeniero informático con menos de un año de experiencia es de 17.248€ brutos al año.

400 horas de trabajo equivalen a trabajar durante aproximadamente tres meses por lo que la realización de este trabajo equivale a un total de 4.312€.

### 2.3.3. Total

Juntando los gastos estimados en recursos materiales (600 €) y en recursos humanos (4.312 €) obtenemos que el coste total aproximado de este proyecto ha sido de 4.912 €.



## Capítulo 3

# Análisis

En este capítulo planteo los requisitos funcionales y no funcionales del sistema que he identificado inicialmente y los que han surgido durante su implementación.

Así mismo también muestro los actores involucrados en el sistema, un diagrama de los casos de uso y una descripción detallada de cada uno de ellos.

### 3.1. Requisitos Funcionales

- **R.F. 1.** Gestión de Archivos: El sistema debe saber sobre qué archivos debe o no actuar para realizar el análisis del plagio.
  - RF 1.1. Se podrán especificar los archivos sobre los que se quiera saber si hay plagio.
  - RF 1.2. El sistema sabrá reconocer los archivos con extensión `.r` y `.R`.
  - RF 1.3. El sistema tiene a R como opción de ejecución para aplicar la detección sobre archivos en este lenguaje.
  - RF 1.4. El sistema ignorará los archivos no válidos (que no sean del lenguaje especificado) o con errores.
- **R.F. 2.** Gestión de Resultados: El sistema deberá de gestionar los resultados obtenidos de aplicar el algoritmo de JPLAG a los archivos.
  - RF 2.1. El sistema guardará los resultados en el directorio que se le especifique.
  - RF 2.2. El sistema creará los subdirectorios necesarios en caso de no existir la ruta al directorio en el que almacenar los resultados.
  - RF 2.3. El sistema mostrará los resultados de la ejecución en terminal en todo momento.

### 3.2. Requisitos No Funcionales

- **R.N.F. 1.** Eficiencia. El frontend implantado no deberá demorar en su uso al sistema más que el resto de los frontends para el resto de lenguajes.
- **R.N.F. 2.** Eficacia. El frontend implantado tendrá que obtener resultados iguales o mejores que el resto de frontends con archivos de código en R.
- **R.N.F. 3.** Extensibilidad. El frontend implantado tendrá que ser fácilmente modificable para poder probar con otros tokens del lenguaje.

### 3.3. Actores

<b>Actor</b>	<i>Usuario</i>				ACT-1
<b>Descripción</b>	<i>Entidad que utiliza la herramienta</i>				
<b>Características</b>	<i>Puede ser cualquier persona que tenga el ejecutable de la herramienta.</i>				
<b>Relaciones</b>					
<b>Referencias</b>	<i>CU_01, CU_02, CU_05</i>				
<b>Autor</b>	<i>Antonio Javier Rodríguez Pérez</i>	<b>Fecha</b>	<i>28/4/18</i>	<b>Versión</b>	<i>1.0</i>

Figura 3.1: Actor 1

<b>Actor</b>	<i>Sistema Interno de detección de Plagio</i>				ACT-2
<b>Descripción</b>	<i>Sistema que aplica el algoritmo de detección de plagio</i>				
<b>Características</b>	<i>Se trata de JPLAG.</i>				
<b>Relaciones</b>					
<b>Referencias</b>	<i>CU_03, CU_04, CU_06</i>				
<b>Autor</b>	<i>Antonio Javier Rodríguez Pérez</i>	<b>Fecha</b>	<i>28/4/18</i>	<b>Versión</b>	<i>1.0</i>

Figura 3.2: Actor 2

### 3.4. Diagrama de Casos de Uso

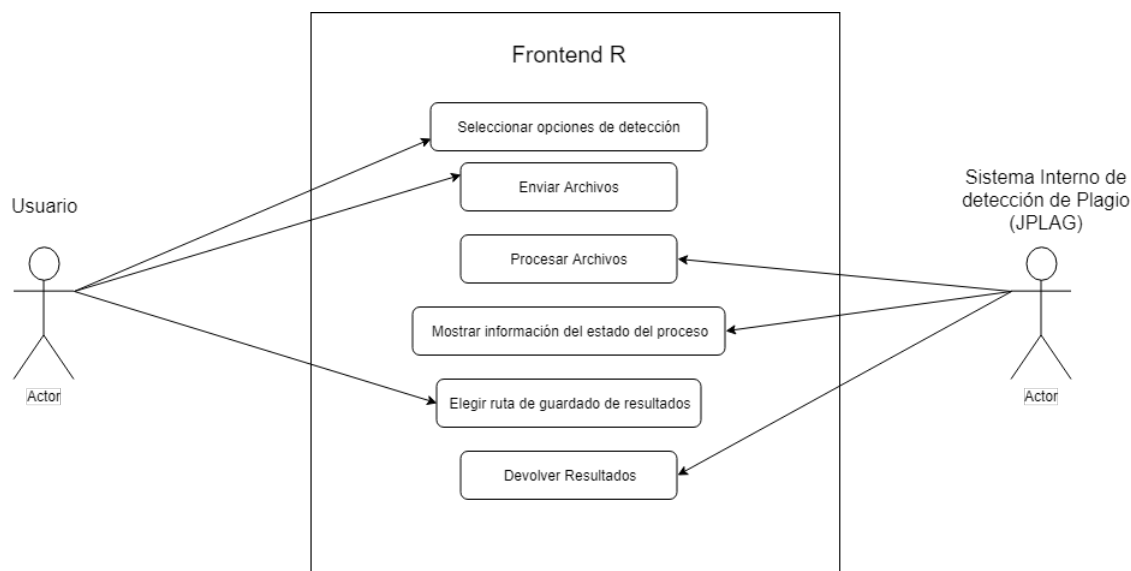


Figura 3.3: Diagrama de casos de uso del usuario del sistema.

### 3.5. Casos de Uso

Caso de Uso	Seleccionar opciones de detección.				CU_01	
Actores	Usuario (principal)					
Tipo	Primario y Esencial					
Referencias	RF_1.3			-		
Precondición	Tener el ejecutable de JPLAG junto con nuestro frontend					
Postcondición	El sistema empieza a procesar las opciones seleccionadas.					
Autor	Antonio Javier Rodríguez Pérez		Fecha	28/4/18	Versión	1.0

Propósito
Permitir que el usuario pueda elegir bajo qué circunstancias debe realizarse la detección de plagio.

Resumen
El usuario escribe en la terminal la orden de ejecución del programa junto con las opciones de ejecución que vea convenientes.

Figura 3.4: Caso de uso 1.

Caso de Uso	Enviar archivos			CU_02	
Actores	Usuario (principal)				
Tipo	Primario y Real				
Referencias	RF_1.1		CU_03		
Precondición	Tener el ejecutable de JPLAG junto con nuestro frontend				
Postcondición	El sistema empieza a aplicar el algoritmo propio a los archivos				
Autor	Antonio Javier Rodríguez Pérez	Fecha	28/4/18	Versión	1.0

<b>Propósito</b>
<i>Permitir que el usuario elija a qué archivos quiere aplicar la detección de plagio.</i>

<b>Resumen</b>
<i>El usuario especifica en la terminal la ruta de los archivos, ya sea poniendo la carpeta en concreto en la que se encuentran o bajo un directorio en general en caso de encontrarse divididos en distintos subdirectorios.</i>

Figura 3.5: Caso de uso 2.

Caso de Uso	Procesar archivos				CU_03
Actores	Sistema interno de detección de Plagio (principal)				
Tipo	Primario y Esencial				
Referencias	RF_1.2, RF_1.4		CU_01, CU_02		
Precondición	El usuario debe de haber dado una ruta válida y los archivos deben de tener la extensión especificada en las opciones seleccionadas.				
Postcondición	El sistema aplica el algoritmo de JPLAG a los archivos válidos				
Autor	Antonio Javier Rodríguez Pérez	Fecha	28/4/18	Versión	1.0

<b>Propósito</b>
<i>Completar la función más importante del sistema, es decir, comprobar con el algoritmo de JPLAG si existe plagio entre los archivos especificados.</i>

<b>Resumen</b>
<i>El sistema comprueba que los archivos se encuentran en la ruta dada y tienen una extensión válida, se pasa sus tokens a tokens estandar de JPLAG y se comprueba la similitud entre las cadenas de tokens.</i>

Figura 3.6: Caso de uso 3.

Caso de Uso	Mostrar información del estado del proceso				CU_04	
Actores	Sistema interno de detección de Plagio (principal)					
Tipo	Primario y Esencial					
Referencias	RF_2.3		-			
Precondición	-					
Postcondición	El sistema muestra cómo progresa el procesado de archivos					
Autor	Antonio Javier Rodríguez Pérez		Fecha	30/4/18	Versión	1.0

<b>Propósito</b>
<i>Informar al usuario del estado del proceso.</i>

<b>Resumen</b>
<i>El sistema muestra en la misma terminal en la que se ha ejecutado el programa el proceso de aplicación del algoritmo de JPLAG los archivos que se eligieron.</i>

Figura 3.7: Caso de uso 4.

Caso de Uso	Elegir ruta de guardado de resultados				CU_05	
Actores	Usuario (principal)					
Tipo	Primario y Esencial					
Referencias	RF_2.1, RF_2.2			-		
Precondición	El usuario debe de especificar una ruta válida.					
Postcondición	El sistema creará los subdirectorios necesarios de la ruta en caso de no existir.					
Autor	Antonio Javier Rodríguez Pérez		Fecha	30/4/18	Versión	1.0

<b>Propósito</b>
<i>Permitir al usuario escoger donde guardar los resultados obtenidos por la herramienta.</i>

<b>Resumen</b>
<i>El sistema comprueba que la ruta de guardado es correcta y en caso de serla y no existir alguna carpeta, de las que se especifica, la crea.</i>

Figura 3.8: Caso de uso 5.

Caso de Uso	Devolver resultados				CU_06	
Actores	Sistema interno de detección de plagio (principal)					
Tipo	Primario y Esencial					
Referencias	RF_2.1, RF_2.2			CU_05		
Precondición	La ejecución del algoritmo debe de haber finalizado sin errores					
Postcondición	El sistema guarda los resultados obtenidos en el directorio por defecto o en una ruta que se haya especificado anteriormente					
Autor	Antonio Javier Rodríguez Pérez		Fecha	30/4/18	Versión	1.0

<b>Propósito</b>
<i>Guardar los resultados obtenidos</i>

<b>Resumen</b>
<i>Una vez completada la ejecución del programa principal con el algoritmo de detección de plagio, el sistema guarda los resultados obtenidos en el directorio por defecto o en una ruta ya especificada, para después poder ser visualizados en el navegador</i>

Figura 3.9: Caso de uso 6.

## Capítulo 4

# Elección de herramientas

### 4.1. Elección de una herramienta base

En la búsqueda de una herramienta para la detección del plagio encontré diversos servicios como Viper [13], Grammarly [14], Ithenticate [15], Plagscan [16] y Turnitin. Entre ellos destaca este último dado que cuenta con una gran base de datos con contenido web, artículos previamente enviados y publicaciones [17].

El acceso a estos servicios se hace a través de su interfaz web y está limitado a universidades y otras instituciones o es necesario tener un plan de suscripción de pago.

Estas herramientas no detectan plagio en sí, sino que devuelven un índice de similitud entre los archivos enviados y el resto de archivos que tienen en su base de datos.

Esta similitud la calculan comparando parámetros como la frecuencia de uso de cada palabra, el parafraseo o la copia literal [18].

Estas herramientas fueron creadas para detectar plagio en trabajos de alumnos escritos en texto plano tales como artículos o memorias de proyectos.

Es por todo esto que estos servicios no son los más indicados para detectar plagio en archivos de código, a menos que se quiera comprobar si el alumno se ha copiado palabra a palabra de algún archivo de código encontrado online.

Se buscaron por tanto herramientas para la detección de plagio en programas en código fuente. Existen numerosas herramientas que cumplen ese propósito. En la tabla de la Figura 1.1 muestro las más usadas aunque también debo mencionar: Sherlock [19], GPlag [20], Marble, Holmes [21] y YAP3 [22].

Estos sistemas de detección tienden a adoptar métricas basadas en diferentes aspectos de un archivo de código tales como la complejidad de los flujos de control que se están usando, el número de estructuras de datos que hay de cada tipo, los nombres de las variables y la cantidad de comentarios [18]. Las herramientas que mejores resultados ofrecen y generalmente usan los

colegios, universidades, academias y otras instituciones para la detección de plagio entre sus alumnos son MOSS y JPLAG.

A continuación se explica en qué consiste cada una de estas.

## 4.2. MOSS

MOSS (Measure Of Software Similarity) es un sistema automático para determinar la similaridad entre programas [6], desarrollado en 1994 por Alex Aiken [23], profesor en UC Berkeley en aquel entonces.

MOSS está disponible como servicio web mediante un servidor que recibe las peticiones que se le envíen a partir de un script en Perl (facilitado por correo automáticamente si lo solicitas). El servidor procesa los archivos enviados con las opciones que se le especifica y devuelve los resultados a través de una interfaz web (devuelve un enlace a los resultados).

Para proteger la confidencialidad del código de los archivos que se envían al servidor, los resultados generados analizando tal código se borran del servidor 14 días después de su envío.

Para enviar una petición al servidor tan solo se tendrá que llamar al script usando una opción de comandos similar a la siguiente (partiendo de que el script y los archivos se encuentran en el directorio actual):

```
moss -l matlab *.m
```

Existen diversas opciones más que podemos usar además de -l. En el primera apartado del capítulo "Ejecución y Pruebas" se explican en más detalle.

En el siguiente subapartado explicamos en qué consiste su algoritmo de detección.

### 4.2.1. Algoritmo usado

MOSS se basa en usar n-gramas a nivel de carácter como criterio de comparación entre archivos [24], esta es una técnica comunmente usada en herramientas de detección de plagio. Un n-grama es una subcadena contigua de n elementos, por ejemplo, con  $n = 5$ , si tenemos la siguiente cadena en un documento:

```
abbcadarrad
```

Los 5-gramas que se obtienen de esta serán:

```
abbca bbcad bcada cadar adarr darra arrad
```



Una vez obtenidos los n-gramas del documento completo, se le calcula a cada uno un identificador propio (hash) basado en la localización del n-grama en el documento y en los caracteres que lo componen.

A continuación, se elige un subconjunto de estos hashes para representar al documento (no podemos coger todos los hashes por motivos de eficiencia ya que habrá casi tantos hashes como caracteres en el archivo).

De esta forma, si la función resumen (hash) usada tiene una probabilidad de colisiones pequeña (una colisión consiste en que dos cadenas diferentes tengan la misma función resumen) cuando dos documentos tengan en común algún hash será extremadamente probable que también compartan n-gramas, esto es lo que usa MOSS para calcular similaridad entre programas.

Para determinar qué funciones resumen escoger normalmente se usa la estrategia de elegir las que sean  $0 \bmod p$ , para algún  $p$  fijo. Esta estrategia es fácil de implementar y nos quedamos únicamente con  $1/p$  de todos los hashes para representar el documento.

Este método no garantiza que se detecten similitudes entre archivos ya que sólo se detectarán n-gramas iguales cuyo hash sea  $0 \bmod p$ .

MOSS soluciona este problema con una técnica llamada windowing (procesar por ventanas).

En lugar de seleccionar n-gramas aleatorios MOSS selecciona como mínimo una función resumen por cada ventana de un cierto tamaño previamente establecido. Además MOSS usa un valor de  $n$  (longitud de n-gramas) grande para evitar ruido en los resultados [21].

#### 4.2.2. Ejemplo de ejecución

MOSS devuelve los resultados en un enlace que nos lleva a una página con los índices de similaridad entre los archivos enviados tal y como podemos ver en la figura 4.1.

Si pinchamos en el nombre de algún archivo de los comparados visualizaremos una página en la que se comparan los dos códigos lado a lado como podemos ver en figura 4.2. En el capítulo de Ejecución y Pruebas se explica con mayor detalle como interpretar estos resultados.

File 1	File 2	Lines Matched
<a href="#">Entrega_5/Entrega_5_Estudiante_1004.R (98%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1028.R (98%)</a>	50
<a href="#">Entrega_5/Entrega_5_Estudiante_1029.R (58%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1034.R (53%)</a>	18
<a href="#">Entrega_5/Entrega_5_Estudiante_1003.R (60%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1023.R (55%)</a>	26
<a href="#">Entrega_5/Entrega_5_Estudiante_1010.R (32%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1034.R (31%)</a>	7
<a href="#">Entrega_5/Entrega_5_Estudiante_1000.R (35%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1008.R (27%)</a>	7
<a href="#">Entrega_5/Entrega_5_Estudiante_1032.R (24%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1034.R (27%)</a>	8
<a href="#">Entrega_5/Entrega_5_Estudiante_1029.R (30%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1032.R (24%)</a>	9
<a href="#">Entrega_5/Entrega_5_Estudiante_1034.R (25%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1036.R (24%)</a>	9
<a href="#">Entrega_5/Entrega_5_Estudiante_1032.R (23%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1036.R (24%)</a>	9
<a href="#">Entrega_5/Entrega_5_Estudiante_1029.R (27%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1036.R (24%)</a>	9
<a href="#">Entrega_5/Entrega_5_Estudiante_1010.R (26%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1029.R (27%)</a>	6
<a href="#">Entrega_5/Entrega_5_Estudiante_1006.R (24%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1034.R (25%)</a>	11
<a href="#">Entrega_5/Entrega_5_Estudiante_1018.R (20%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1034.R (22%)</a>	9
<a href="#">Entrega_5/Entrega_5_Estudiante_1000.R (26%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1027.R (27%)</a>	8
<a href="#">Entrega_5/Entrega_5_Estudiante_1019.R (16%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1031.R (39%)</a>	6
<a href="#">Entrega_5/Entrega_5_Estudiante_1011.R (14%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1028.R (18%)</a>	11
<a href="#">Entrega_5/Entrega_5_Estudiante_1004.R (18%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1011.R (14%)</a>	11
<a href="#">Entrega_5/Entrega_5_Estudiante_1017.R (19%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1018.R (18%)</a>	9
<a href="#">Entrega_5/Entrega_5_Estudiante_1017.R (18%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1034.R (20%)</a>	8
<a href="#">Entrega_5/Entrega_5_Estudiante_1006.R (19%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1017.R (18%)</a>	8
<a href="#">Entrega_5/Entrega_5_Estudiante_1026.R (13%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1037.R (15%)</a>	8
<a href="#">Entrega_5/Entrega_5_Estudiante_1017.R (17%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1026.R (13%)</a>	9
<a href="#">Entrega_5/Entrega_5_Estudiante_1018.R (16%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1029.R (20%)</a>	9
<a href="#">Entrega_5/Entrega_5_Estudiante_1027.R (21%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1029.R (19%)</a>	2
<a href="#">Entrega_5/Entrega_5_Estudiante_1019.R (14%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1034.R (18%)</a>	8
<a href="#">Entrega_5/Entrega_5_Estudiante_1033.R (11%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1036.R (16%)</a>	4
<a href="#">Entrega_5/Entrega_5_Estudiante_1026.R (12%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1034.R (17%)</a>	7
<a href="#">Entrega_5/Entrega_5_Estudiante_1024.R (18%)</a>	<a href="#">Entrega_5/Entrega_5_Estudiante_1031.R (31%)</a>	16

Figura 4.1: Ejemplo de página principal de resultados (MOSS)

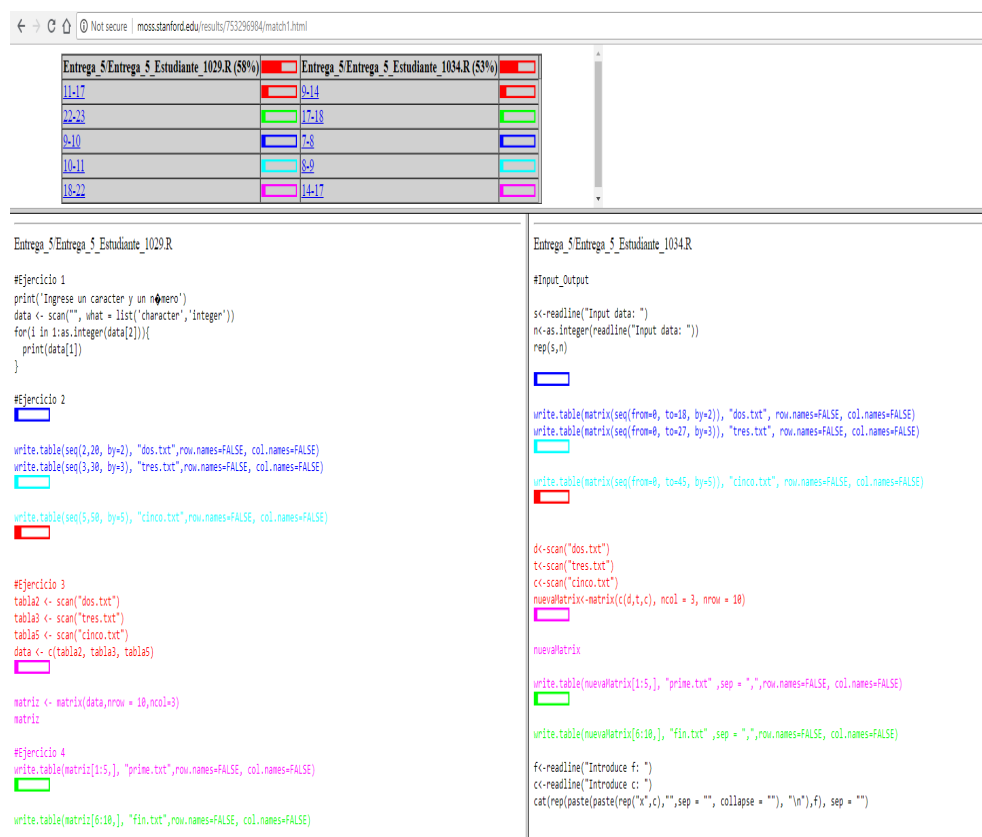


Figura 4.2: Ejemplo de página de comparación de código de MOSS

### 4.3. JPLAG

JPLAG es un sistema que encuentra similitudes entre conjuntos de archivos en código fuente [7]. Fue desarrollado por Guido Malpohl en 1996. JPLAG fue creado con la idea de que comparar programas basándose en un vector de características no es suficiente, ya que esto ignora gran parte de la similitud estructural entre programas [21]. Es por esto que en su lugar intenta emparejar a partir de características estructurales convirtiendo primero el código en una lista de tokens para después aplicar un algoritmo que encuentra emparejamientos entre las listas de tokens obtenidas priorizando emparejamientos lo más largos posible.

Para usar esta herramienta es necesario descargar su código disponible en Github[25] y ejecutarlo localmente en nuestro PC. JPLAG genera entonces un conjunto de páginas en HTML con los resultados.

JPLAG también está disponible como servicio web aunque sus propios desarrolladores no recomiendan su uso ya que usa bibliotecas anticuadas y no

está acabado (además de que ya no aceptan nuevos registros).

#### 4.3.1. Algoritmo usado

JPLAG opera en 2 fases [26]:

1. Se analizan sintácticamente o léxicamente (dependiendo del lenguaje) todos los programas enviados, convirtiéndolos en cadenas de tokens.
2. Se comparan las cadenas de tokens por pares.

##### 1. Análisis Sintáctico

Esta es la única parte que depende del lenguaje en cuestión en el que están escritos los archivos que se envían a JPLAG. Se trata del frontend de JPLAG.

El frontend se encarga de realizar un análisis sintáctico o léxico (parser o scanner) de los archivos para reconocer los tokens que los componen como podemos apreciar en la figura 4.3.

Java source code	Generated tokens
1 public class Count {	BEGINCLASS
2 public static void main(String[] args)	VARDEF,BEGINMETHOD
3 throws java.io.IOException {	
4 int count = 0;	VARDEF,ASSIGN
5	
6 while (System.in.read() != -1)	APPLY,BEGINWHILE
7 count++;	ASSIGN,ENDWHILE
8 System.out.println(count+" chars.");	APPLY
9 }	ENDMETHOD
10 }	ENDCLASS

Figura 4.3: Ejemplo de tokens extraídos de código en java

Los tokens que se escogen deben de ser representativos del lenguaje que estamos tratando para que capturen la esencia del programa (la cual es difícil de cambiar por un plagiador).

Los espacios en blanco y los comentarios nunca se deben de tener en cuenta como tokens.

##### 2. Comparación de los tokens

El algoritmo usado para comparar las cadenas de tokens es básicamente Greedy String Tiling, algoritmo desarrollado por Michael Wise, creador de YAP3, herramienta donde lo aplica [22].

Cuando se comparan 2 cadenas de tokens deben cumplirse tres reglas:

1. Un token de una cadena sólo puede ser asociado a un token de la cadena con la que se compara.
2. Las subcadenas de tokens deben de ser independientes de su posición en la cadena principal de tokens.
3. Tendrán preferencia los emparejamientos largos (de muchos tokens seguidos) sobre los cortos.

Al aplicar la tercera regla estamos ante un algoritmo voraz (greedy) que consiste en tres bucles anidados: el primer bucle itera sobre todos los tokens de la primera cadena (que llamaremos X), el segundo bucle compara el token en el que estamos de X con cada token de la segunda cadena (que llamaremos Y), en caso de ser el mismo token en algún punto del segundo bucle se entra en el tercer bucle, donde se siguen comparando entre X e Y para ver cuántos tokens seguidos son iguales desde ese punto.

Una vez hecho esto marcamos los emparejamientos de mayor longitud encontrados y repetimos el proceso completo una y otra vez pero sin tener en cuenta en cada vuelta los tokens que ya han sido marcados.

Se repite el proceso hasta que no se encuentre ningún emparejamiento.

Este es el algoritmo que usa JPLAG con la diferencia de le aplica algunas optimizaciones en tiempo de ejecución. Aunque la complejidad del algoritmo en el peor caso ( $\mathcal{O}(n^3)$ ) no se puede reducir, sí que es posible reducir la complejidad media de los casos prácticos a  $\mathcal{O}(n)$ .

Esto es lo que consigue JPLAG adaptando al algoritmo de greedy string tiling la idea del algoritmo de Karp-Rabin [27] en el cual se encuentran ocurrencias de una cadena corta en otra larga usando funciones hash.

#### 4.3.2. Ejemplo de ejecución

JPLAG devuelve los resultados en un conjunto de páginas HTML cuya página principal muestra los mayores índices de similaridad entre los archivos enviados junto con un histograma con todos los índices entre los archivos tal y como podemos ver en la figura 4.4.

Si pinchamos en el nombre de algún archivo de los comparados visualizaremos una página en la que se comparan los dos códigos lado a lado como podemos ver en figura 4.5. Podemos apreciar que la forma de devolver los resultados de MOSS y JPLAG son muy similares, esto se debe a que la interfaz web de MOSS también la hizo Guido Malpohl(creador de JPLAG) [28].

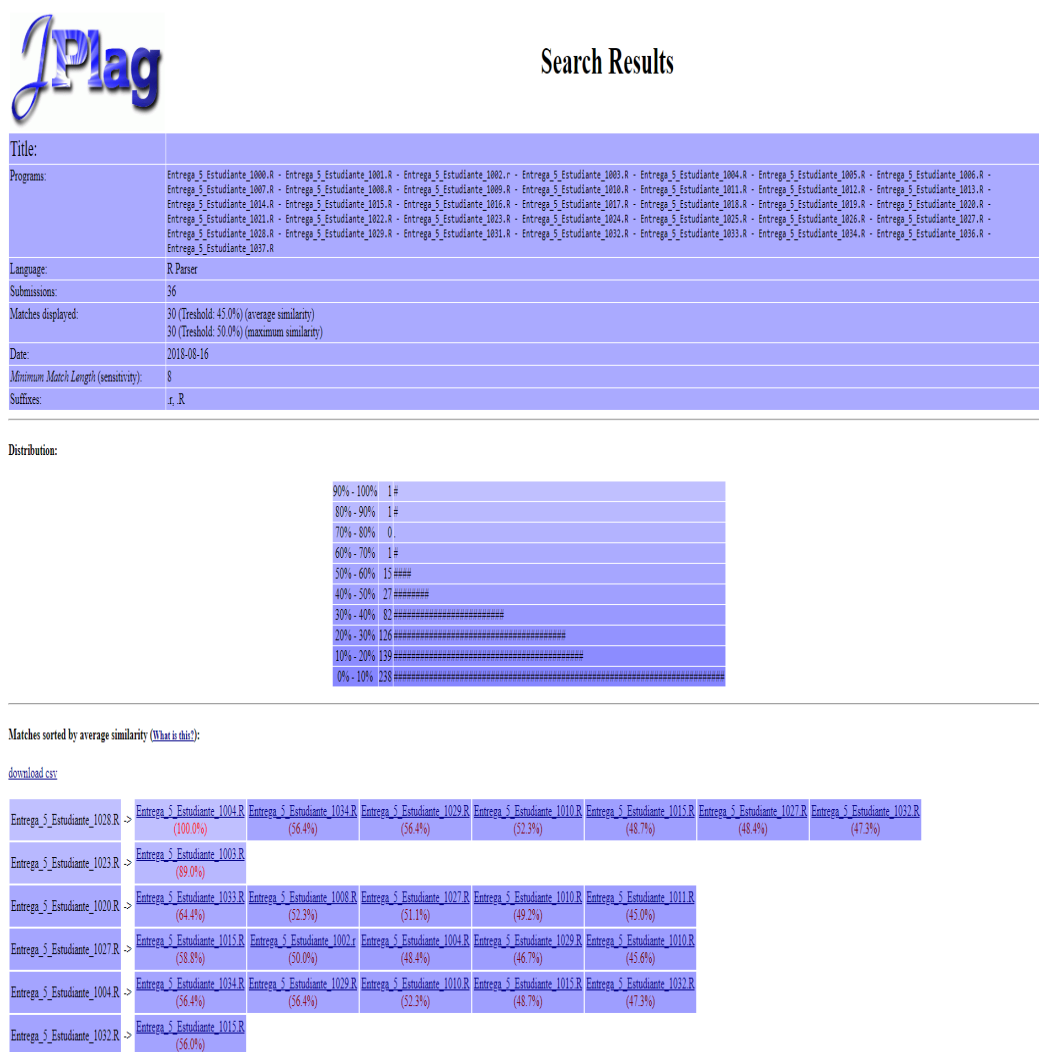


Figura 4.4: Ejemplo de página principal de resultados (JPLAG)



Figura 4.5: Ejemplo de página de comparación de código de JPLAG

### 4.3.3. Razones de su elección

Dado que JPLAG es más fácilmente modificable, debido a que su código está disponible y no es necesario cambiar el algoritmo principal que usa para que funcione con otros lenguajes, (tan solo hay que crear un frontend) JPLAG ha sido la herramienta escogida para adaptar a R.





## Capítulo 5

# Implementación

### 5.1. Estructura de nuestro frontend

Se ha utilizado github y git para tener nuestra propia versión de JPLAG en nuestra cuenta de Github (<https://github.com/AntonioJavierRP>), gracias a esto podremos realizar los cambios que veamos necesarios en nuestra versión y mandar un pull request al repositorio original en [25] por si el autor quiere incorporar el frontend que hemos creado para R a su versión.

Los cambios y las pruebas hechas en dicho código se han realizado en nuestra propia máquina local.

Nuestro frontend consta de la estructura de archivos que podemos ver en la Figura 5.1 basada en los frontends ya existentes, especialmente en el de Java 1.7 y Python 3, ya que usan la misma herramienta que nosotros para crear el analizador sintáctico.

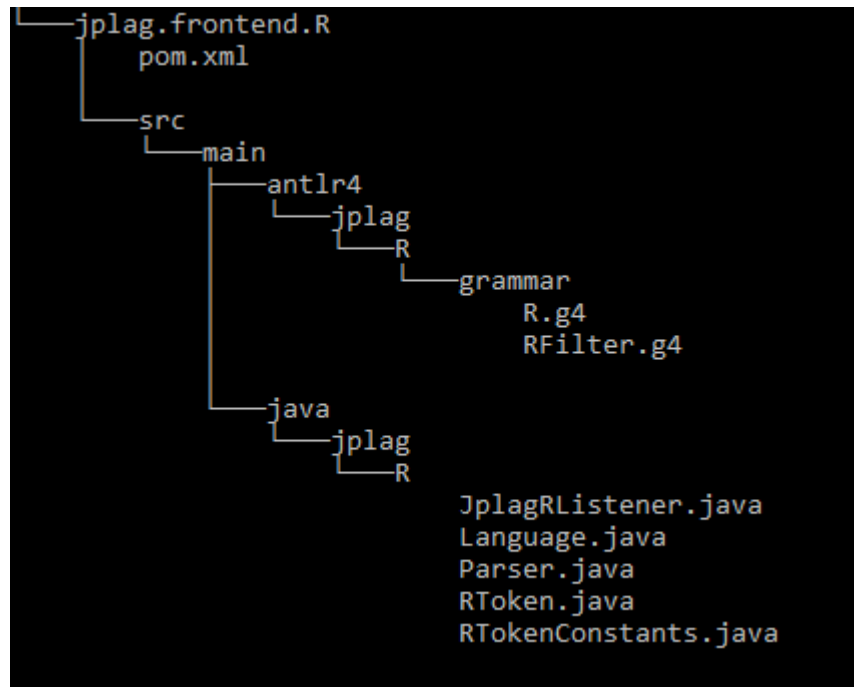


Figura 5.1: Árbol de archivos del frontend creado

Como se puede apreciar, nuestro frontend consta de 8 archivos:

- **pom.xml**: archivo con la opciones de construcción y compilación de nuestro frontend en JPLAG.
- **R.g4**: archivo de gramática que representa a R procesado por ANTLR4.
- **RFilter.g4**: otro archivo de gramática que representa a R procesado por ANTLR4.
- **JplagListener.java**: este archivo se encarga de seleccionar los tokens que mejor representan al lenguaje.
- **Lenguaje.java**: en este archivo constan las características del lenguaje que JPLAG debe de saber antes de procesar los archivos enviados.
- **Parser.java**: este archivo se encarga de extraer los tokens y de enviárselos a JPLAG.
- **RToken.java** y **RTokenConstants.java**: en estos archivo especificamos los tokens importantes de R.

En los siguientes apartados se explica en más profundidad la función de cada uno de estos archivos.

### 5.1.1. Apache Maven

Para la construcción y gestión de JPLAG el autor original ha usado Maven.

Apache Maven es una herramienta de gestión y comprensión de proyectos software en Java [29], es decir, es lo que se encarga de que se compilen todos los fuentes correctamente.

Maven permite la construcción de proyectos a través de archivos POM (project object model) y de un conjunto de plugins que tienen todos los proyectos que usan Maven (tiene que ser instalado previamente).

Un archivo pom.xml tiene una estructura básica similar a la que se muestra en el ejemplo a continuación en la Figura 5.2.

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0"
2.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4.       http://maven.apache.org/xsd/maven-4.0.0.xsd">
5.   <modelVersion>4.0.0</modelVersion>
6.
7.   <groupId>org.codehaus.mojo</groupId>
8.   <artifactId>my-project</artifactId>
9.   <version>1.0</version>
10. </project>
```

Figura 5.2: Archivo ejemplo pom.xml

Para que nuestro frontend se tuviese en cuenta cuando se construye el proyecto se han tenido que modificar tres de los archivos pom.xml ya existentes en el proyecto añadiéndoles dependencias y módulos.

Así mismo se ha tenido también que crear nuestro propio archivo pom.xml dentro de nuestra carpeta del frontend como ya vimos en la Figura 5.1.

## 5.2. Analizador utilizado

Como ya mencionamos anteriormente, el frontend se encarga de reconocer las cadenas de tokens para que se les pueda aplicar el algoritmo. Para poder identificar estos tokens necesitamos un parser (analizador sintáctico). En el resto de frontends los parser creados se han hecho usando JAVACC o ANTLR.

Estas son herramientas para la creación de analizadores de lenguajes basándose en gramáticas. Para la creación de nuestro analizador sintáctico de R hemos escogido ANTLR4.

### 5.2.1. ANTLR

ANTLR (ANother Tool for Language Recognition) es un generador de analizadores sintácticos desarrollado por Terrence Parr [30].

Un analizador sintáctico procesa un trozo de texto y lo transforma en una estructura organizada, en un árbol de sintaxis. Este árbol representa la estructura lógica del contenido del código que ha procesado. Un ejemplo de tal árbol es el que vemos en la Figura 5.3 en el que se representa el código del algoritmo de Euclides.

Para obtener un árbol de sintaxis necesitaremos definir una gramática, invocar a ANTLR, para que genere un parser a partir de esta, y pasarle el código del que quieres obtener el árbol al parser.

ANTLR4 es la versión 4 de ANTLR y aporta numerosas nuevas funcionalidades que reducen la curva de aprendizaje necesaria para desarrollar la gramática de tu lenguaje.

La característica más importante de esta versión es que acepta cualquier gramática que le hagas procesar a excepción de gramáticas con recursión indirecta por la izquierda (esto se da cuando una regla referencia a otra regla a la izquierda y esta segunda regla termina referenciando a la primera sin llegar a un símbolo terminal o token).

Los archivos que contienen la gramática tendrán que tener la extensión .g4, el nombre de la gramática como nombre del archivo y empezar con la línea de código : grammar < *nombredelarchivo* >; . Aunque también puede comenzar siendo un parser grammar o lexer grammar si sólo vamos a definir reglas del analizador léxico o sintáctico. En nuestro caso el archivo se llama R.g4 y el archivo empieza con grammar R;

El resto del archivo contendrá reglas del analizador léxico (lexer rules) y reglas del analizador sintáctico (parser rules).

Las "lexer rules" son las definiciones de los tokens básicos que conforman el lenguaje y siguen la sintaxis de las reglas sintácticas. Las reglas del analizador léxico se escriben en mayúscula y al final del archivo, en Figura 5.4 podemos ver un ejemplo de estas reglas.

Las "parser rules" son las reglas que muestran las combinaciones posibles de los tokens básicos (definidos en las lexer rules) en el lenguaje.

Tenemos una primera regla (en el ejemplo que mostramos es 'chat') que genera el resto de reglas. Estas reglas se escriben en minúsculas y se deben de poner antes que las reglas del analizador léxico.

Podemos ver un ejemplo de estas en la Figura 5.5.

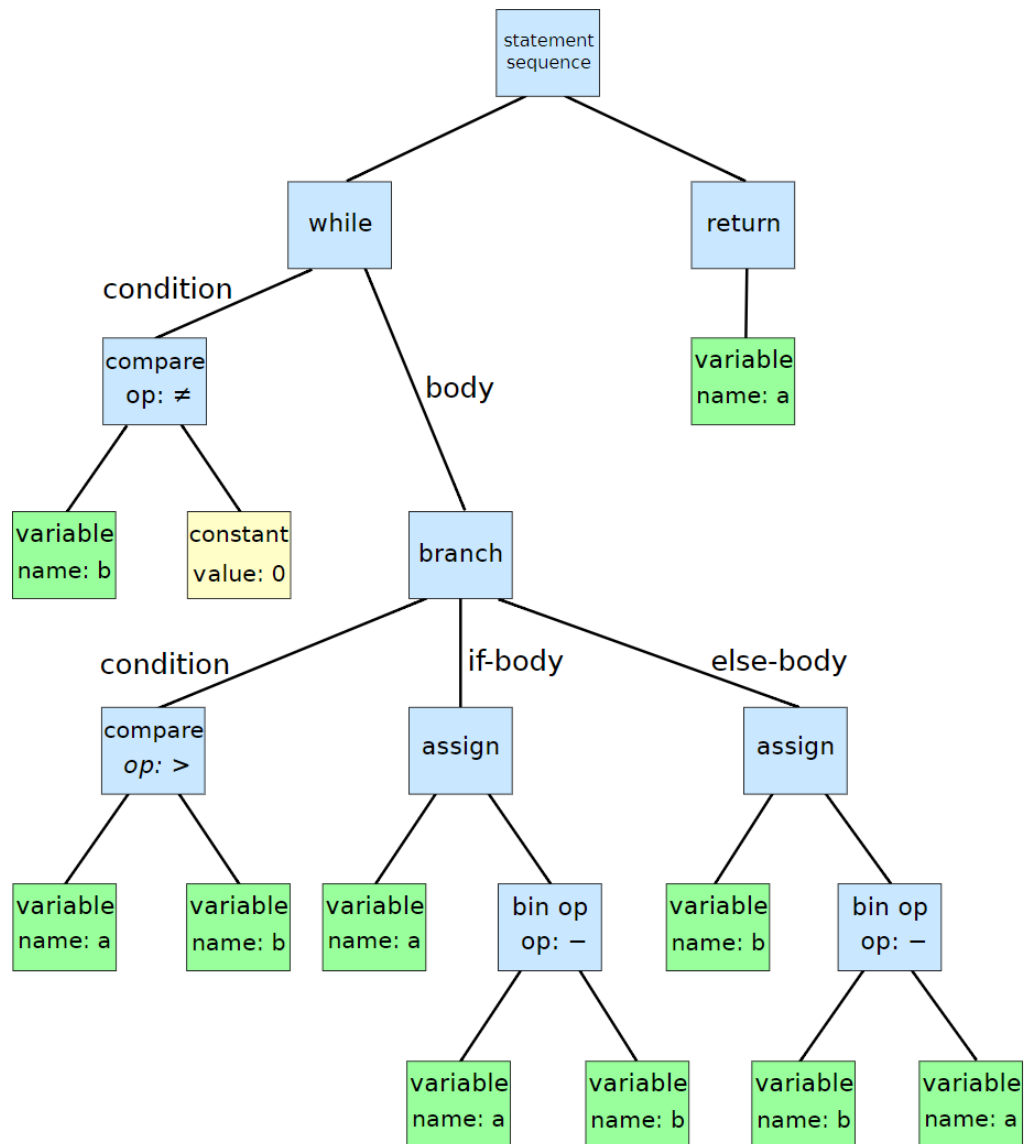


Figura 5.3: Árbol de sintaxis del algoritmo de Euclides.

```

1  /*
2  * Lexer rules
3  */
4
5  fragment A      : ('A' | 'a') ;
6  fragment S      : ('S' | 's') ;
7  fragment Y      : ('Y' | 'y') ;
8  fragment H      : ('H' | 'h') ;
9  fragment O      : ('O' | 'o') ;
10 fragment U      : ('U' | 'u') ;
11 fragment T      : ('T' | 't') ;
12
13 fragment LOWERCASE : [a-z] ;
14 fragment UPPERCASE : [A-Z] ;
15
16
17 SAYS              : S A Y S ;
18
19 SHOUTS            : S H O U T S ;
20
21 WORD              : (LOWERCASE | UPPERCASE | '_' )+ ;
22
23 WHITESPACE        : ( ' ' | '\t' ) ;
24
25 NEWLINE           : ( '\r'? '\n' | '\r' )+ ;
26
27 TEXT              : ( '[' | '(' ) ~[\)]+ ( '[' | '(' ) ;
28
29 fragment DIGIT     : [0-9] ;
30
31 NUMBER            : DIGIT+ ( [.,] DIGIT+ )? ;

```

Figura 5.4: Ejemplo de lexer rules.

```

1  /*
2  * Parser Rules
3  */
4
5  chat              : line+ EOF ;
6
7  line              : name command message NEWLINE;
8
9  message           : (emoticon | link | color | mention | WORD | WHITESPACE)+ ;
10
11 name              : WORD WHITESPACE;
12
13 command           : (SAYS | SHOUTS) ':' WHITESPACE ;
14
15 emoticon          : ':' '-'? ')'
16                   | ':' '-'? '('
17                   ;
18
19 link              : '[' TEXT ']' '(' TEXT ')' ;
20
21 color             : '/' WORD '/' message '/' ;
22
23 mention           : '@' WORD ;

```

Figura 5.5: Ejemplo de parser rules.

### 5.3. Adaptación del parser a JPLAG

La gramática que hemos usado se encuentra en el archivo `R.g4` junto con `RFilter.g4`. Nuestra gramática está basada en una ya existente para R en ANTLR4 encontrada en [31]. Hemos añadido una gran cantidad de reglas del análisis sintáctico nuevas en el archivo `R.g4` para poder quedarnos con los tokens más representativos del lenguaje ya que necesitamos que los tokens que estamos agrupando aparezcan agrupados como parser rule, como por ejemplo, en el caso de los tokens para el flujo de control "if". Hemos creado la siguiente regla:

```
1 if_statement : 'if' '(' expr ')' expr | 'if' '(' expr ')' expr 'else'
               expr ;
```

Esta regla la hemos creado de forma separada para que después en el Listener (que crea ANTLR4 al procesar la gramática) podamos reconocerla. Este Listener contiene métodos que se llaman cuando se entra o sale de cada regla del analizador sintáctico.

Es por eso que en archivo `JplagRListener.java` lo que hacemos es implementar ese Listener (ya que se trata de una interfaz) para que cada vez que se llame a un método de una regla asociada a un token que queramos se añada a la cadena de tokens del archivo que estamos procesando.

El archivo `RFilter.g4` tan solo contiene reglas para eliminar saltos de línea que en realidad son espacios en blanco así que no lo hemos modificado. Las funciones que se encargan de cargar los archivos, llamar al analizador sintáctico para obtener los tokens de estos y mandar los tokens encontrados a JPLAG están en el archivo `parser.java`.

En `lenguaje.java` especificamos las extensiones válidas que requiere que tengan los archivos del lenguaje, el nombre del parser y el número mínimo de emparejamientos seguidos de tokens que deben hacerse para poder marcarlo como emparejamiento (tal y como explicamos en el apartado del algoritmo). El resto de métodos especifican opciones que no vamos a variar respecto de los otros frontends.

Por último, en `RTokenConstants.java` especificamos el nombre de los tokens más importantes del lenguaje en una interfaz llamada `RTokenConstants`, mientras que en `RToken.java` implementamos esta interfaz añadiéndole un método que devuelve los tokens pasados a string.

Una vez hecha toda esta implementación, hemos terminado la creación del frontend, realizamos entonces las pruebas necesarias para comprobar si la lista inicial de tokens seleccionados de R representa suficientemente bien al lenguaje.

Terminamos realizando modificaciones y seleccionamos más tokens basándonos en los seleccionados en el resto de frontends y en la página de referencia del lenguaje R [32].



## Capítulo 6

# Ejecución y Pruebas

### Instrucciones de uso y ejecución

En los siguiente apartados mostramos los pasos a seguir para la obtención de resultados en MOSS y en JPLAG.

#### 6.1. Ejecución en MOSS

Siguiendo las instrucciones en [6] y en [33] sabemos que, para poder comunicarnos en el servidor de MOSS y mandarle los archivos, necesitaremos un script en Perl.

Para obtener este script enviamos un mensaje por correo electrónico a `moss@moss.stanford.edu` con lo siguiente en el cuerpo del mensaje:

```
registeruser  
mail usuario@dominio
```

En nuestro caso pondremos `antoniojrp@correo.ugr.es` en la parte de `usuario@dominio`.

Unos cinco minutos después del envío de este mensaje recibiremos una respuesta automática del servidor en nuestro correo con el script que necesitamos y con instrucciones de su uso. En el script se detalla que podemos compartir este script con otros profesores de programación pero que por favor no lo compartamos en un sitio de acceso público ("Feel free to share this script with other instructors of programming classes, but please do not place the script in a publicly accessible place."), por esta razón no mostramos el contenido completo del script en esta memoria.

Ahora que tenemos el script, lo colocamos en el directorio donde tenemos los archivos o las carpetas de los archivos que queremos comparar, para mayor comodidad a la hora de escribir las rutas de los archivos al llamar al script.

Nos situamos con la terminal (este script funciona en Unix, en caso estar en Windows se tendrá que usar Cygwin) en el directorio donde está el script y ejecutamos los siguientes comandos:

Para modo Matlab:

```
1 $ ./moss -l matlab -c "Entrega X" "Entrega X"/*.r "Entrega X"/*.R
```

Para modo texto plano:

```
1 $ ./moss -l ascii -c "Entrega X" "Entrega X"/*.r "Entrega X"/*.R
```

Entrega X es la entrega de alumnos que queremos evaluar actualmente, ya que en nuestro caso hay un total de 7 entregas diferentes ( siete trabajos diferentes con más de 30 archivos de ejercicios de alumnos en R en cada uno), y hemos nombrado el directorio donde se encuentran cada una de las entregas "Entrega 1", "Entrega 2", "Entrega 3", etc hasta siete entregas.

### Explicación de opciones:

Con la opción `-l` especificamos el lenguaje en el que (supuestamente) están los archivos para que MOSS lo tenga en cuenta en su algoritmo. En nuestro caso hemos decidido realizar las pruebas como si los archivos estuvieran escritos en texto plano y en Matlab (dado que es el más similar a R de todas las posibles opciones). Si esta opción se deja vacía, MOSS supondrá que los archivos están en C.

Con la opción `-c` le asocia un nombre a los resultados que se generan. En nuestro caso le asociamos el número de la entrega.

Entre las opciones que no hemos especificado tenemos:

- `-m`: Especifica el número máximo de veces que un mismo fragmento de código tiene que aparecer para que se ignore en los emparejamientos. Por defecto esta opción tiene el valor 10.
- `-d`: Especifica que los archivos a evaluar se agrupan por directorio y no por archivos individuales, es decir, cada alumno es un directorio completo lleno de archivos y se compara entre directorios.
- `-b`: Nos permite especificar un archivo base. El código que tengan en común el resto de archivos con el archivo base no se tendrá en cuenta en los emparejamientos.

Al ejecutar estos comandos, le enviamos los archivos al servidor y obtenemos un enlace como el siguiente:

<http://moss.stanford.edu/results/169343157>

Es posible que en el momento que se lea esto el enlace de ejemplo no exista ya que el servidor borra los resultados 15 días después de generarlos.

En nuestro caso han tardado de media un minuto en generarse, variando en función del tamaño de los archivos de la entrega (treinta segundos para la entrega más corta y un minuto y cuarenta segundos para la más larga).

Al acceder a este enlace podemos ver en nuestro navegador una página principal con los resultados, en el caso por ejemplo de la Entrega 2, hemos obtenido la página de resultados de la Figura 6.1.

La página muestra la fecha en la que fue generada, las opciones de ejecución elegidas a la hora de ejecutar el script y la etiqueta que le asociamos con la opción -c.

Seguido de esto tenemos seis links de ayuda con las bases de cómo interpretar los resultados, contacto y créditos.

El resto de la página contiene una tabla de pares de archivos con código similar, ordenado por la cantidad de código en común entre estos.

Al final del nombre de cada archivo está el porcentaje del código de ese archivo que se considera coincidente con el del otro en su misma línea.

Por último cabe mencionar que la columna "Lines Matched" contiene el número aproximado de líneas de código emparejadas entre los archivos de la fila.

La mejor estrategia para interpretar estos resultados es empezar evaluando los archivos con mayor porcentaje coincidente e ir bajando en la lista hasta que los archivos que estemos comprobando estén llenos de emparejamientos que sean falsos positivos.

Por cada par de archivos emparejados hay una página con información sobre los fragmentos que coinciden. Esta página está estructurada en tres marcos: uno con una tabla con todos los fragmentos coincidentes (Figura 6.2), y los otros dos con el código de cada archivo (Figura 6.3).

En la tabla de Figura 6.2 se muestra un rango con el número de líneas de cada archivo donde se considera que ha habido copia, junto con un "termómetro" para mostrar gráficamente cómo de grande es este fragmento en comparación con el archivo completo. Estos "termómetros" así mismo son links que te sitúan en la parte del código al que se refieren en los otros dos marcos (Figura 6.3).

Moss Results

Wed Aug 22 11:29:13 PDT 2018

Options -l matlab -m 10

Entrega 2

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

File 1	File 2	Lines Matched
<a href="#">Entrega_2/Entrega_2_Estudiante_1004.R (97%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1028.R (97%)</a>	619
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (79%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1021.R (64%)</a>	179
<a href="#">Entrega_2/Entrega_2_Estudiante_1015.R (52%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1021.R (51%)</a>	109
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (40%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1015.R (34%)</a>	93
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (30%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1000.R (25%)</a>	70
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (29%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1017.R (28%)</a>	95
<a href="#">Entrega_2/Entrega_2_Estudiante_1000.R (24%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1007.R (28%)</a>	103
<a href="#">Entrega_2/Entrega_2_Estudiante_1015.R (21%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1017.R (24%)</a>	64
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (25%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1009.R (24%)</a>	40
<a href="#">Entrega_2/Entrega_2_Estudiante_1017.R (23%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1034.R (22%)</a>	76
<a href="#">Entrega_2/Entrega_2_Estudiante_1015.R (19%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1034.R (20%)</a>	41
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (22%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1032.R (20%)</a>	46
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (22%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1007.R (23%)</a>	78
<a href="#">Entrega_2/Entrega_2_Estudiante_1017.R (21%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1021.R (18%)</a>	73
<a href="#">Entrega_2/Entrega_2_Estudiante_1000.R (19%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1015.R (18%)</a>	51
<a href="#">Entrega_2/Entrega_2_Estudiante_1017.R (20%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1031.R (20%)</a>	75
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (21%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1031.R (20%)</a>	66
<a href="#">Entrega_2/Entrega_2_Estudiante_1015.R (18%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1031.R (20%)</a>	59
<a href="#">Entrega_2/Entrega_2_Estudiante_1009.R (20%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1021.R (17%)</a>	34
<a href="#">Entrega_2/Entrega_2_Estudiante_1000.R (17%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1021.R (17%)</a>	46
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (19%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1034.R (17%)</a>	39
<a href="#">Entrega_2/Entrega_2_Estudiante_1000.R (16%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1009.R (18%)</a>	35
<a href="#">Entrega_2/Entrega_2_Estudiante_1031.R (17%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1034.R (17%)</a>	41
<a href="#">Entrega_2/Entrega_2_Estudiante_1014.R (18%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1033.R (18%)</a>	54
<a href="#">Entrega_2/Entrega_2_Estudiante_1002.r (19%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1024.R (57%)</a>	98
<a href="#">Entrega_2/Entrega_2_Estudiante_1014.R (17%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1026.R (21%)</a>	34
<a href="#">Entrega_2/Entrega_2_Estudiante_1015.R (15%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1025.R (17%)</a>	45
<a href="#">Entrega_2/Entrega_2_Estudiante_1015.R (15%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1016.R (16%)</a>	51
<a href="#">Entrega_2/Entrega_2_Estudiante_1021.R (14%)</a>	<a href="#">Entrega_2/Entrega_2_Estudiante_1032.R (15%)</a>	40

Figura 6.1: Página principal de resultados de la segunda entrega con matlab como opción (MOSS)

Entrega_2/Entrega_2_Estudiante_1002.r (79%)		Entrega_2/Entrega_2_Estudiante_1021.R (64%)	
50-93		58-100	
183-202		204-230	
143-180		156-199	
28-50		34-58	
102-115		106-116	
116-125		119-125	
13-23		15-26	
131-132		144-145	
135-142		149-156	

Figura 6.2: Tabla comparativa entre dos estudiantes en la segunda entrega con matlab como opción (MOSS)

```

y<-matrix(c(1,4,0,0,1,-1),2,3,byrow=TRUE)
#Y calcula los efectos de los siguientes comandos
x[1,]
x[2,]
x[,2]
y[1,2]
y[,2:3]
#Transforma la matriz m que creaste en el ejercicio anterior en un array multidimensional. (Pista: averigua lo
array(data=m,dim=dim(m))
#Crea un array de 5 x 5 y rellénalo con valores del 1 al 25. Investiga la función array(). Llama al array x
xx=array(1:25, dim=c(5,5,1))
#Escribe el array x en un vector y
y<-as.vector(x)
#Dadas las matrices m1 y m2 usa rbind() y cbind() para crear matrices nuevas utilizando estas funciones, llama
m1<-matrix(1, nr = 2, nc = 2)

m2<-matrix(2, nr = 2, nc = 2)
M1<-rbind(m1,m2)
M2<-cbind(m1,m2)
#rbind ha juntado por filas y cbind por columnas
#El operador para el producto de dos matrices es ' %* %'. Por ejemplo, considerando las dos matrices creadas e
M1%M2
#Usa la matriz M1 del ejercicio anterior y aplica la función t(). ¿qué hace esa función?
t(M1)
#Ejecuta los siguientes comandos basados en la función diag() sobre las matrices creadas anteriormente m1 y m2
diag(m1)
diag(rbind(m1,m2)%*%cbind(m1,m2))
diag(m1)<-10
diag(3)
v<-c(10,20,30)
diag(v)
diag(2.1,nr=3,nc=5)
#Se pueden obtener las diagonales de las matrices que deseemos incluso introducirlas
#Ordena la matriz:
x<-matrix(1:100, ncol=10)
#en orden descendente por su segunda columna y asigna el resultado a una nueva matrix x1. Pista: función order
x1<-x[order(x[,2],decreasing=TRUE),]
#en orden descendente por su segunda fila y asigna el resultado a una nueva matrix x2
x2<-x[order(x[2,],decreasing=TRUE)]

```

```

y<-matrix(c(1,4,0,0,1,-1),2,3,byrow=TRUE)
x[1,]
x[2,]
x[,2]
y[1,2]
y[,2:3]
#transforma la matriz m en array multidim
array(data=m,dim=dim(m))
#crea array 5x5 con valores 1 a 25
x<-array(data=c(1:25),dim=c(5,5))
x
#escribe array x en vector y
y<-as.vector(x)
#usa rbind ycbind para crear matrices
m1<-matrix(1,nr=2,nc=2)

m2<-matrix(2,nr=2,nc=2)
M1<-rbind(m1,m2)
M2<-cbind(m1,m2)
#multiplicar M1 M2
M1%M2
#traspuesta de M1
t(M1)
#comandos con diag
diag(m1)
diag(rbind(m1,m2)%*%cbind(m1,m2))
diag(m1)<-10
diag(3)
v<-c(10,20,30)
diag(v)
diag(2.1,nr=3,nc=5)
#ordenar la matriz
x<-matrix(1:100,ncol=10)

```

Figura 6.3: Parte comparativa del código fuente de dos estudiantes en la segunda entrega con matlab como opción (MOSS)

## 6.2. Ejecución en JPLAG

Para usar JPLAG tendremos que descargar el código fuente completo de [25]. Para usar nuestra versión con nuestro frontend de R podemos descargar el código de <https://github.com/AntonioJavierRP/jplag>.

Una vez hecho esto nos situamos con nuestra terminal en la carpeta principal, es decir, la que contiene todos los frontends. Para generar los ejecutables de cada frontend ejecutaremos la orden:

```
1 $ mvn clean generate-sources package
```

En caso de querer generar un único ejecutable usaremos la siguiente orden dentro de la carpeta jplag:

```
1 $ mvn clean generate-sources assembly:assembly
```

Para ejecutar cualquiera de las dos últimas ordenes es necesario tener instalado Apache Maven, las instrucciones de su instalación se encuentran en [34]

Los ejecutables creados estarán en sus respectivas carpetas "target", en el caso de tener un único ejecutable este estará en jplag/target.

### 6.2.1. Obtención de resultados

En nuestro caso hemos llamado a JPLAG con las siguientes opciones (situándonos en jplag/target):

```
1 $ java -jar jplag-2.11.9-SNAPSHOT-jar-with-dependencies.jar -l R -r  
   ../resultados/Entrega_X/ -s <ruta_directorio_entrega_X>
```

Se ha ejecutado esta orden para siete entregas distintas, con unos 30 archivos por entrega, dos veces, una para R y otra para texto plano.

Como se puede apreciar se ha usado la opción -l para especificar el lenguaje, -r para especificar donde guardar los resultados y -s para especificar la ruta donde se encuentran todos los archivos que se quieren evaluar. Podemos visualizar el resto de opciones disponibles con su descripción ejecutando el .jar sin opciones:

```
1 $ java -jar jplag-2.11.9-SNAPSHOT-jar-with-dependencies.jar
```

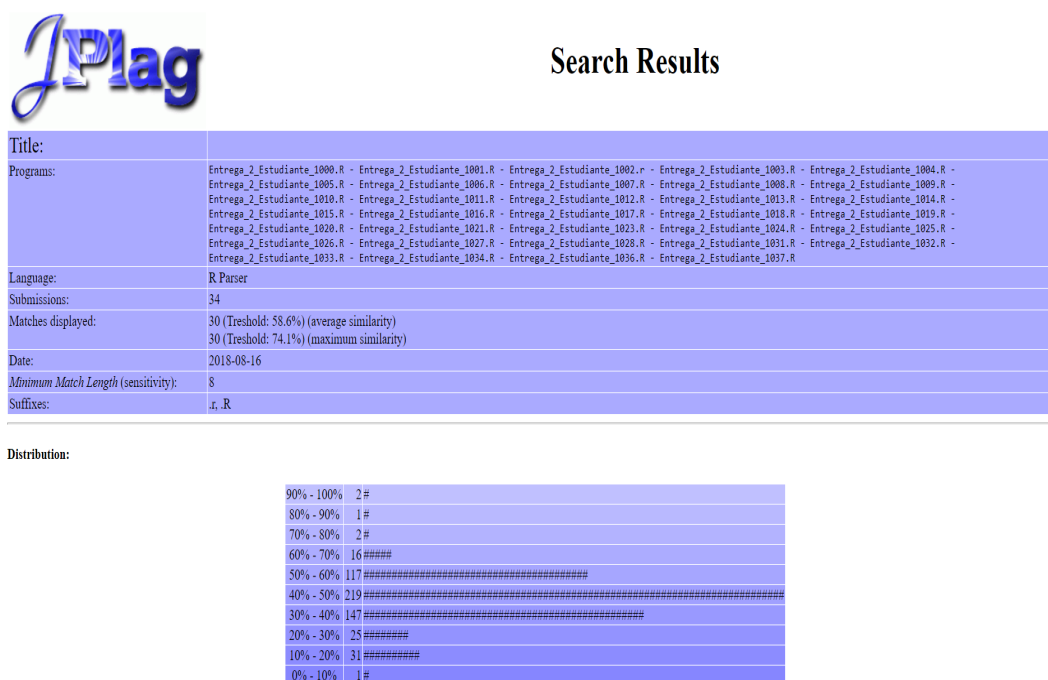


Figura 6.4: Página principal de los resultados de la Entrega 2 (JPLAG) parte 1

Los resultados serán visualizables en el navegador, podemos abrir el archivo `index.html` para situarnos en la página principal de análisis de los resultados (Figuras 6.4 y 6.5). La interpretación de estos resultados es muy similar a la que explicamos en apartados anteriores con MOSS, ya que su interfaz es muy similar.

A continuación explicamos cómo interpretar los elementos de la página principal de JPLAG.

La Figura 6.4 contiene la parte de arriba de la página de resultados de la segunda entrega de los alumnos obtenida por nuestro frontend de R. Se lista en una tabla el nombre de los programas que se han detectado, el idioma del parser usado, el número de archivos totales, el número de archivos emparejados que se muestran, la fecha en la que se obtuvieron los resultados, el número mínimo de tokens iguales que tiene que haber para que se considere un emparejamiento y las extensiones de los archivos procesados. Seguido de esto, JPLAG muestra un histograma de los valores de similaridad encontrados en todos los pares de programas.

Podemos usar este histograma para identificar los casos que son plagio obvio y los que no tienen nada en común.

Los pares que se encuentran entre estos dos lados del espectro se encuentran

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

[download csv](#)

Entrega_2_Estudiante_1028.R ->	<a href="#">Entrega_2_Estudiante_1004.R</a> (100.0%)	<a href="#">Entrega_2_Estudiante_1002.r</a> (59.9%)				
Entrega_2_Estudiante_1027.R ->	<a href="#">Entrega_2_Estudiante_1024.R</a> (91.6%)					
Entrega_2_Estudiante_1002.r ->	<a href="#">Entrega_2_Estudiante_1021.R</a> (87.5%)	<a href="#">Entrega_2_Estudiante_1037.R</a> (70.3%)	<a href="#">Entrega_2_Estudiante_1015.R</a> (67.1%)	<a href="#">Entrega_2_Estudiante_1032.R</a> (64.6%)	<a href="#">Entrega_2_Estudiante_1017.R</a> (64.1%)	<a href="#">Entrega_2_Estudiante_1031.R</a> (63.6%)
Entrega_2_Estudiante_1015.R ->	<a href="#">Entrega_2_Estudiante_1021.R</a> (76.8%)	<a href="#">Entrega_2_Estudiante_1031.R</a> (60.7%)	<a href="#">Entrega_2_Estudiante_1037.R</a> (60.1%)	<a href="#">Entrega_2_Estudiante_1000.R</a> (59.1%)		
Entrega_2_Estudiante_1037.R ->	<a href="#">Entrega_2_Estudiante_1031.R</a> (65.2%)	<a href="#">Entrega_2_Estudiante_1032.R</a> (63.8%)	<a href="#">Entrega_2_Estudiante_1017.R</a> (63.7%)	<a href="#">Entrega_2_Estudiante_1021.R</a> (60.5%)	<a href="#">Entrega_2_Estudiante_1000.R</a> (59.4%)	
Entrega_2_Estudiante_1017.R ->	<a href="#">Entrega_2_Estudiante_1032.R</a> (64.1%)	<a href="#">Entrega_2_Estudiante_1003.R</a> (61.8%)	<a href="#">Entrega_2_Estudiante_1009.R</a> (61.3%)	<a href="#">Entrega_2_Estudiante_1025.R</a> (61.2%)		
Entrega_2_Estudiante_1032.R ->	<a href="#">Entrega_2_Estudiante_1031.R</a> (61.0%)	<a href="#">Entrega_2_Estudiante_1025.R</a> (59.4%)				
Entrega_2_Estudiante_1006.R ->	<a href="#">Entrega_2_Estudiante_1000.R</a> (59.4%)					

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

[download csv](#)

Entrega_2_Estudiante_1028.R ->	<a href="#">Entrega_2_Estudiante_1004.R</a> (100.0%)	<a href="#">Entrega_2_Estudiante_1027.R</a> (79.2%)	<a href="#">Entrega_2_Estudiante_1024.R</a> (74.6%)	<a href="#">Entrega_2_Estudiante_1010.R</a> (74.2%)		
Entrega_2_Estudiante_1002.r ->	<a href="#">Entrega_2_Estudiante_1021.R</a> (96.7%)	<a href="#">Entrega_2_Estudiante_1027.R</a> (79.2%)	<a href="#">Entrega_2_Estudiante_1024.R</a> (73.7%)			
Entrega_2_Estudiante_1027.R ->	<a href="#">Entrega_2_Estudiante_1024.R</a> (95.5%)	<a href="#">Entrega_2_Estudiante_1000.R</a> (83.7%)	<a href="#">Entrega_2_Estudiante_1031.R</a> (79.7%)	<a href="#">Entrega_2_Estudiante_1025.R</a> (79.2%)	<a href="#">Entrega_2_Estudiante_1004.R</a> (79.2%)	<a href="#">Entrega_2_Estudiante_1015.R</a> (78.0%)
Entrega_2_Estudiante_1010.R ->	<a href="#">Entrega_2_Estudiante_1012.R</a> (87.1%)	<a href="#">Entrega_2_Estudiante_1014.R</a> (77.1%)	<a href="#">Entrega_2_Estudiante_1018.R</a> (75.7%)	<a href="#">Entrega_2_Estudiante_1021.R</a> (74.2%)	<a href="#">Entrega_2_Estudiante_1004.R</a> (74.2%)	
Entrega_2_Estudiante_1024.R ->	<a href="#">Entrega_2_Estudiante_1000.R</a> (82.9%)	<a href="#">Entrega_2_Estudiante_1025.R</a> (80.8%)	<a href="#">Entrega_2_Estudiante_1011.R</a> (80.8%)	<a href="#">Entrega_2_Estudiante_1037.R</a> (79.2%)	<a href="#">Entrega_2_Estudiante_1032.R</a> (75.6%)	<a href="#">Entrega_2_Estudiante_1015.R</a> (74.6%)
Entrega_2_Estudiante_1021.R ->	<a href="#">Entrega_2_Estudiante_1015.R</a> (80.6%)					

Figura 6.5: Página principal de los resultados de la Entrega 2 (JPLAG) parte 2

a continuación en la parte de abajo de la página en Figura 6.5. Se muestran ordenados en base a dos criterios, basándose en la cobertura entre programas (es decir, cuánto de un programa se ha usado en otro):

- Similaridad Media: media entre las coberturas de ambos programas entre si. Emparejamientos con una alta cobertura indican pares muy similares.
- Similaridad Máxima: el máximo entre las dos coberturas. Esta medida es útil para encontrar los casos en los que ha habido plagio y se ha añadido código extra para "rellenar" y que los archivos tengan tamaños muy diferentes.

Haciendo click en el nombre de alguno de estos archivos iremos a la página comparativa de código entre estos.

En la parte superior de la página (Figura 6.6) tenemos a la izquierda el nombre de los archivos que se comparan, su porcentaje de similaridad y dos hiperlinks uno para volver a la página principal y otro de ayuda. A la derecha de esta página se muestra una tabla similar a la que teníamos en MOSS, en la que también se listan los trozos de código donde se han encontrado cadenas de tokens idénticas. Por cada par de trozos de código se especifica



el número de tokens en común que se han encontrado seguidos y el color que se le ha asignado para que su visualización sea más facil en la parte de abajo de esta misma página (Figura 6.7).

En esta parte cada pasaje encontrado tiene una especie de flecha al principio, al pinchar sobre esta se alinea la otra parte del código del otro programa para que se puedan ver lado a lado los trozos de código con el mismo color. Si una parte del código no coincide con la del otro programa esta se mostrará en color negro.

Matches for Entrega_2_Estudiante_1021.R & Entrega_2_Estudiante_1002.r		
87.5%		
<a href="#">INDEX</a>	<a href="#">HELP</a>	
Entrega_2_Estudiante_1021.R (80.0%)	Entrega_2_Estudiante_1002.r (96.77419%)	Tokens
Entrega_2_Estudiante_1021.R(2-2)	Entrega_2_Estudiante_1002.r(3-2)	24
Entrega_2_Estudiante_1021.R(7-14)	Entrega_2_Estudiante_1002.r(7-12)	17
Entrega_2_Estudiante_1021.R(15-26)	Entrega_2_Estudiante_1002.r(12-23)	34
Entrega_2_Estudiante_1021.R(26-56)	Entrega_2_Estudiante_1002.r(22-48)	65
Entrega_2_Estudiante_1021.R(58-100)	Entrega_2_Estudiante_1002.r(50-93)	110
Entrega_2_Estudiante_1021.R(102-116)	Entrega_2_Estudiante_1002.r(101-115)	45
Entrega_2_Estudiante_1021.R(118-128)	Entrega_2_Estudiante_1002.r(116-125)	31
Entrega_2_Estudiante_1021.R(144-220)	Entrega_2_Estudiante_1002.r(131-202)	134

Figura 6.6: Página de comparación de código entre dos archivos en JPLAG (parte de arriba)

<pre>matrix(1:6,2,3,byrow=TRUE)  #vector z con 30 primeros numeros y crear matriz n 3x10 con esos numeros z&lt;-c(1:30) m&lt;-matrix(z,2,10)  #la columna en vector m[,3]  #comprobar efectos x&lt;-matrix(c(3,-1,21,1),2,2) y&lt;-matrix(c(1,4,6,8,1,-1),2,3,byrow=TRUE) x[1,] x[2,] x[,2] y[1,2] y[,2:3]  #transforma la matriz n en array multidim array(data=m,dim=dim(m))  #crea array 5x5 con valores 1 a 25 x&lt;-array(data=c(1:25),dim=c(5,5)) x  #escribe array x en vector y y&lt;-as.vector(x)  #usa rbind y cbind para crear matrices m1&lt;-matrix(1,nr=2,nc=2) m2&lt;-matrix(2,nr=2,nc=2) M1&lt;-rbind(m1,m2) M2&lt;-cbind(m1,m2)  #multiplicar M1 M2 M3&lt;-M2  #traspuesta de M1 t(M1)  #comandos con diag diag(m1) diag(rbind(m1,m2))%cbind(m1,m2)) diag(m1)&lt;-10 diag(3) vc&lt;-c(10,20,30) diag(v) diag(2,1,nr=3,nc=5)  #ordena la matriz x&lt;-matrix(1:100,ncol=10) #en orden descendente por su segunda columna x1&lt;-x[order(x[,2]),decreasing=TRUE,] #en orden descendente por su segunda fila x1[order(x[2,]),decreasing=TRUE] #ordena solo la primera columna de x descendente</pre>	<pre>matrix(1:6, 2, 3) matrix(1:6, 2, 3, byrow=TRUE) #Crea un vector z con los 30 primeros números y crea con el una matriz n con 3 filas y 10 columnas. z&lt;-1:30 m&lt;-matrix(z,3,10) #Escribe la tercera columna en un vector m[,3] #Crea en 8 las matrices x&lt;-matrix(c(3,21,-1,4),2,2,byrow = TRUE) y&lt;-matrix(c(1,4,6,8,1,-1),2,3,byrow=TRUE) #Calcula los efectos de los siguientes comandos x[1,] x[2,] x[,2] y[1,2] y[,2:3] #Transforma la matriz m que creaste en el ejercicio anterior en un array multidimensional. (Pista: averigua lo que puedes de la función dim()) array(data=m,dim=dim(m)) #Crea un array de 5 x 5 y rellénalo con valores del 1 al 25. Investiga la función array(). Llama al array x x&lt;-array(1:25, dim=c(5,5)) #Escribe el array x en un vector y y&lt;-as.vector(x) #Dadas las matrices m1 y m2 usa rbind() y cbind() para crear matrices nuevas utilizando estas funciones, llámalas M1 y M2. ¿En que se diferencian? m1&lt;- matrix(1, nr = 2, nc = 2) m2&lt;- matrix(2, nr = 2, nc = 2) M1&lt;-rbind(m1,m2) M2&lt;-cbind(m1,m2) #rbind ha juntado por filas y cbind por columnas #El operador para el producto de dos matrices es '%*%'. Por ejemplo, considerando las dos matrices creadas en el ejercicio anterior utilízalo. M3&lt;-M2 #Usa la matriz M1 del ejercicio anterior y aplica la función t(). ¿Qué hace esa función? t(M1) #Ejecuta los siguientes comandos basados en la función diag() sobre las matrices creadas anteriormente m1 y m2. ¿Qué tipo de acciones puedes ejecutar? diag(m1) diag(rbind(m1,m2))%cbind(m1,m2)) diag(m1)&lt;-10 diag(3) vc&lt;-c(10,20,30) diag(v) diag(2,1,nr=3,nc=5) #Se pueden obtener las diagonales de las matrices que deseemos incluso introducir las #ordena la matriz: x&lt;- matrix(1:100, ncol=10) #en orden descendente por su segunda columna y asigna el resultado a una nueva matriz x1. Pista: función order() x1&lt;-x[order(x[,2]),decreasing=TRUE,] #en orden descendente por su segunda fila y asigna el resultado a una nueva matriz x2 x2&lt;-x[order(x[2,]),decreasing=TRUE] #ordena solo la primera columna de x de forma descendente sort(x[,1],decreasing=TRUE) #accede al dataset "women". #Primero confirma que los datos están ordenados de forma creciente según la altura (height) y el peso (weight) sin mirar los datos is.unsorted(order(women["weight"],women["height"])) #Crea una nueva variable "bmi". Este valor responde a la siguiente fórmula: BMI = ( Height in Pounds / (Height in inches) x (Height in inches) ) bmi&lt;-(women["weight"]/(women["height"]^2))*703 #ordena el dataframe por el valor de bmi y luego ordena alfabético de la variable name women[order(bmi),]</pre>
--	--

Figura 6.7: Página de comparación de código entre dos archivos en JPLAG (parte de abajo)

## 6.3. Pruebas

Para comprobar que el frontend que hemos creado facilita la detección de plagio entre archivos en R, se ha hecho un estudio de las diferencias entre los resultados obtenidos con nuestra versión de JPLAG usando el frontend de R y los resultados que se podían obtener usando otros medios ya existentes. Los archivos en R usados para realizar este estudio han sido las entregas de los alumnos de la asignatura "Introducción a la programación para ciencia de datos" del Master de ciencia de datos de la Universidad de Granada. Cada entrega consiste en ejercicios sobre diferentes aspectos de R, empezando por una introducción con ejercicios simples y añadiendo nuevos conceptos y estructuras de R con cada nueva entrega.

Son un total de 7 entregas:

- **Entrega 1.** Introducción a R y ejercicios de vectores. 36 programas enviados(cada programa es un archivo de un único alumno).
- **Entrega 2.** Ejercicios de matrices, arrays y factores. 34 programas enviados.
- **Entrega 3.** Ejercicios de manipulación de strings. 36 programas enviados.
- **Entrega 4.** Ejercicios de dataframes y listas. 33 programas enviados.
- **Entrega 5.** Ejercicios de entrada y salida. 36 programas enviados.
- **Entrega 6.** Ejercicios de Funciones. 36 programas enviados.
- **Entrega 7.** Ejercicios de estructuras de programación en R. 36 programas enviados.

Los ejercicios de los alumnos se han convertido a archivos .R (ya que en algunos archivos estaban en .pdf, .docx, .rmd,...) para que JPLAG los pueda procesar y han sido además anonimizados asignándosele a cada alumno un ID. Hay 38 alumnos en total.

Cada una de estas entregas ha sido procesada en JPLAG con nuestro frontend, con el frontend de texto plano y en MOSS en modo Matlab y texto plano.

Se han analizado los resultados en base a las entregas:

### 6.3.1. Análisis de resultados

Se han analizado los resultados en base a las peculiaridades de cada entrega, ya que estas se dividen en diferentes aspectos del lenguaje, lo cual es idóneo para observar la precisión de la detección de plagio en ejercicios de áreas totalmente distintas.

La primera entrega consiste en una serie de ejercicios de introducción a R, para que el alumno aprenda cómo generar secuencias de números, usar vectores y a usar las llamadas más comunes que se usan en R.

Por norma general los alumnos han necesitado aproximadamente cien líneas de código para completar estos ejercicios, aunque los programas tienen más de doscientas en total debido a comentarios con aclaraciones sobre el código y con los enunciados de los ejercicios.

Los resultados obtenidos en esta entrega con JPLAG se pueden ver plasmados en dos histogramas en Figura 6.8 donde se muestran los valores de similaridad media para todos los pares posibles de programas (en esta entrega al ser 36 programas habrá 630 emparejamientos diferentes).

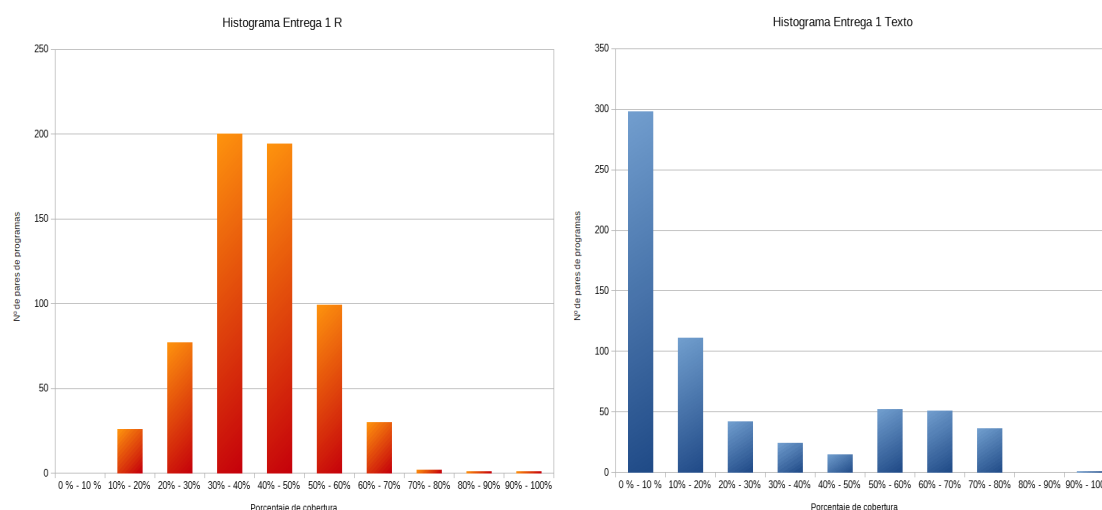


Figura 6.8: Histogramas de valores de similaridad entre todos los pares de programas de la primera entrega.

Para evitar mostrar este caso repetidas veces con cada gráfica que se muestre, se ha de mencionar que en todas las entregas hechas menos en la séptima, los estudiantes con id 1004 y 1028 se han copiado al 100 %, es decir, han entregado programas iguales.

Como era de esperar, el frontend de JPLAG de texto plano, correspondiente al histograma de la derecha, sólo detecta plagio en caso de que se haya hecho una copia literal del código, es decir, que sea exáctamente el mismo con las mismas palabras. Por esta razón la mayor parte de los pares de programas tienen un índice de cobertura (o similaridad) muy bajo. Existen algunos casos con un índice alto, pero esto se debe a similaridad en

los comentarios, ya que algunos alumnos han escrito en los comentarios del programa el enunciado de los ejercicios o explicaciones similares.

En el caso de nuestro frontend de R (histograma a la izquierda), se detectan una gran cantidad de pares con un índice de cobertura superior al treinta por ciento, esto se debe a que los ejercicios de esta entrega son muy concretos y no hay muchas formas de hacerlos por lo que es normal que los alumnos hayan usado estructuras muy similares que al fin y al cabo están formadas por los mismos tokens.

Los pares con un índice superior al 70 por ciento son casos casi seguros de plagio ya que es altamente improbable que dos alumnos tengan más de un setenta por ciento de su código estructuralmente igual en 100 líneas de código sin haberse copiado.

En las gráficas de Figura 6.9 podemos ver una representación de los resultados obtenidos en MOSS.

En estas gráficas figuran los índices de similaridad de los 10 pares de programas más "parecidos" de la primera entrega.

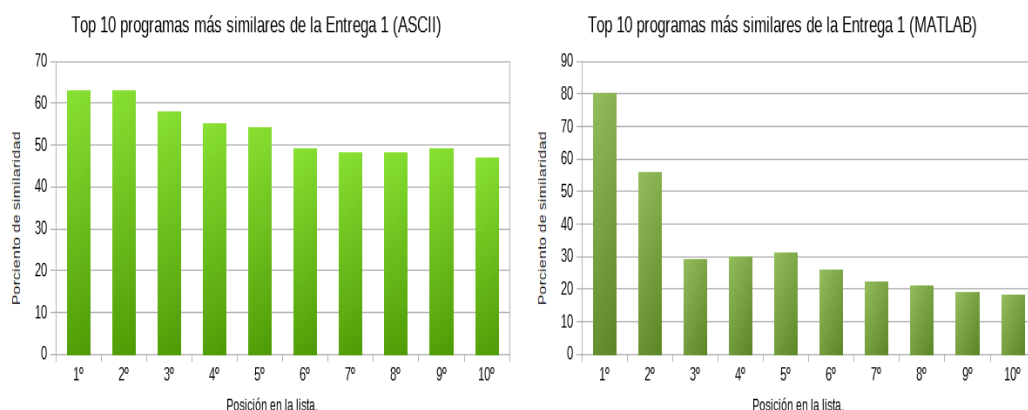


Figura 6.9: 10 primeros pares de programas con mayor cantidad de código en común de la entrega 1

Se puede apreciar que estos índices son más bajos que los obtenidos en JPLAG, esto en parte es debido a que MOSS ignora fragmentos de código que se repiten en al menos diez programas, lo que permite lidiar en cierta forma con los comentarios con el enunciado de los ejercicios (si es que al menos diez alumnos han decidido ponerlos).

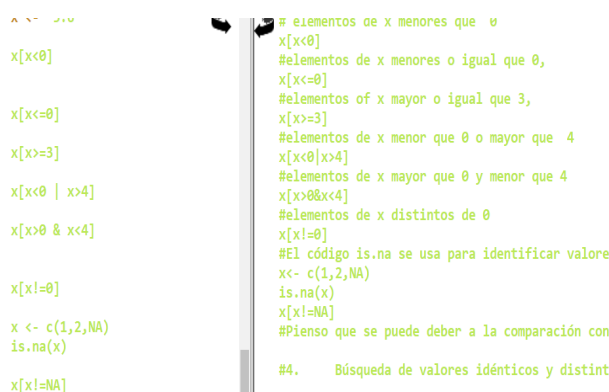
Aun así, los resultados en modo texto plano (ASCII) sólo nos ayudan a encontrar programas con comentarios iguales o con código idéntico.

MOSS no encuentra tantas ocurrencias de plagio como nuestra versión de JPLAG, ya que se basa en un algoritmo que no tiene en cuenta los tokens

si no un vector de características.

Lo que hace MOSS es tener en cuenta las palabras comunes del lenguaje elegido y en caso de aparecer no las considera plagio y no las tiene en cuenta en los índices de similaridad.

Los resultados obtenidos eligiendo como opción Matlab (ya que es el lenguaje mas parecido a R de los compatibles) son mejores que los de texto plano ya que nos han permitido encontrar algunas partes con sintaxis idéntica entre programas. De todas formas nuestro frontend de R ha encontrado estas mismas similitudes sintácticas además de una gran cantidad de plagio estructural y plagio en el que se cambia de orden el código o se añade código muerto, como en el caso de la Figura 6.10, donde se ha encontrado un fragmento muy similar entre el código de los alumnos 1002 y 1036 que el resto de herramientas no ha detectado.



```

# elementos de x menores que 0
x[x<0]
#elementos de x menores o igual que 0,
x[x<=0]
#elementos of x mayor o igual que 3,
x[x>=3]
#elementos de x menor que 0 o mayor que 4
x[x<0|x>4]
#elementos de x mayor que 0 y menor que 4
x[x>0&& x<4]
#elementos de x distintos de 0
x[x!=0]
#El código is.na se usa para identificar valore
x<- c(1,2,NA)
is.na(x)
x[x!=NA]

#4. Búsqueda de valores idénticos y distint

```

Figura 6.10: Ejemplo de plagio que sólo nuestro frontend ha encontrado en la primera entrega

Las entregas 2 y 4 son muy similares ya que contienen ejercicios de uso de matrices, manipular y mostrar los datos que queramos de un dataset y manejo de factores(sólo en la 2), listas (sólo en la 4)y dataframes (sólo en la 4).

Ambas entregas son más del doble de largas que la primera entrega.

A continuación mostramos las gráficas de los resultados para estas entregas (Figuras 6.11, 6.12,6.13, 6.14):

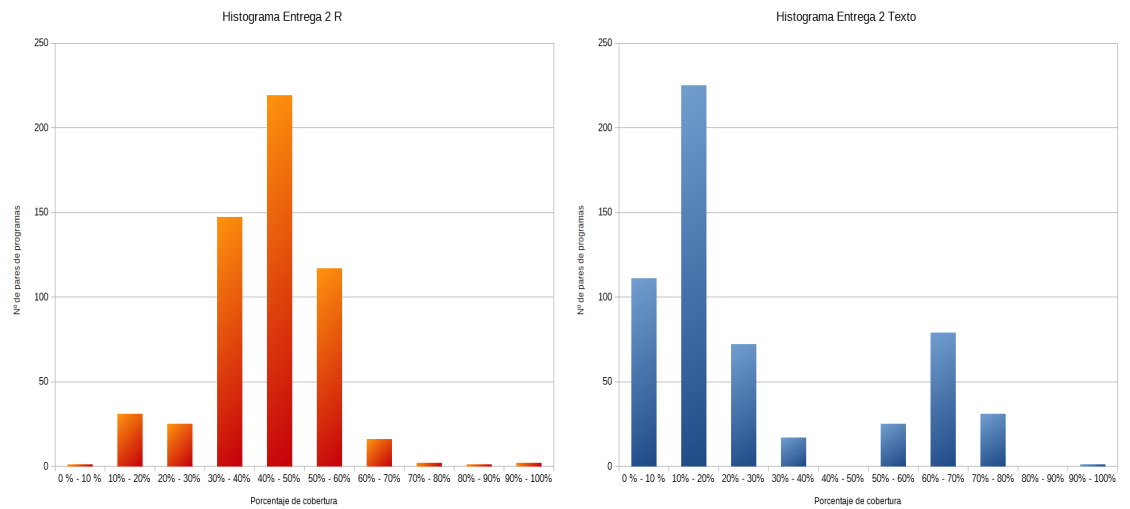


Figura 6.11: Histogramas de valores de similaridad entre todos los pares de programas de la segunda entrega.

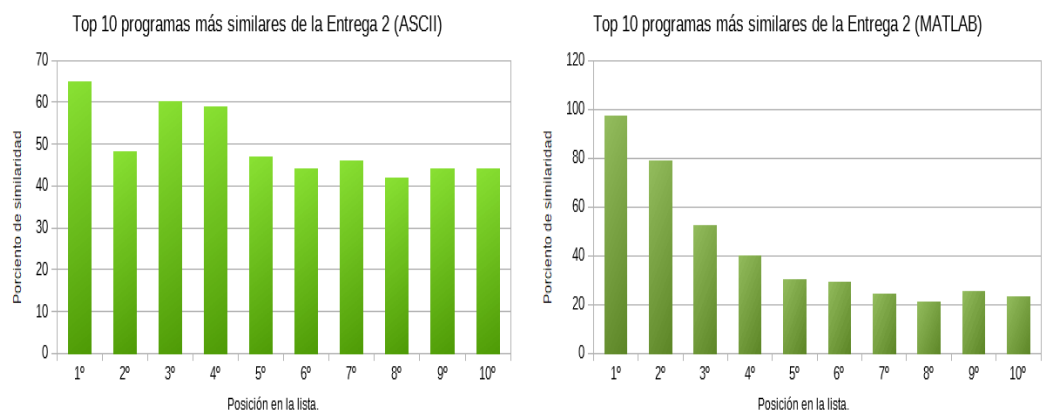


Figura 6.12: 10 primeros pares de programas con mayor cantidad de código en común de la entrega 2

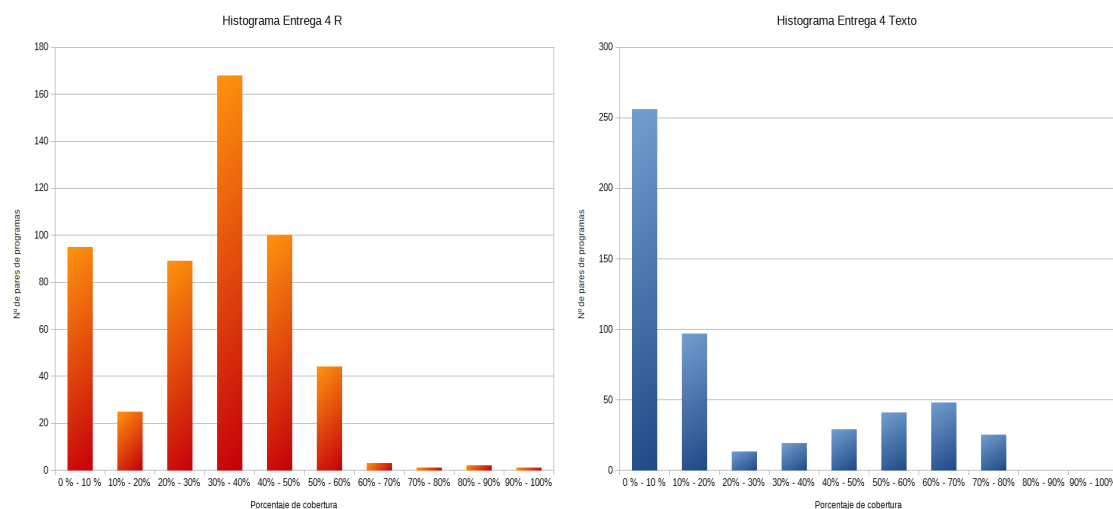


Figura 6.13: Histogramas de valores de similaridad entre todos los pares de programas de la cuarta entrega.

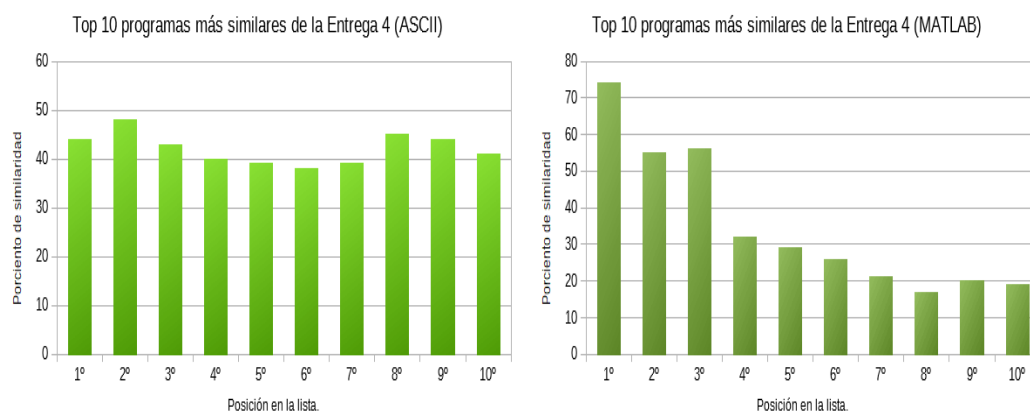


Figura 6.14: 10 primeros pares de programas con mayor cantidad de código en común de la entrega 4

Al consistir estas en programas más largos es más complicado que tengan una gran cantidad de código en común, es por esto que los histogramas de JPLAG y las gráficas de MOSS de estas entregas tienen índices de similaridad menores que los de la entrega 1.

Aún así JPLAG con nuestro frontend sigue encontrando bastantes pares con similaridad superior al 30 %.

El resto de entregas son cortas pero obtenemos gráficas distintas debido a los ejercicios de cada una de estas:

La tercera y quinta entregas son algo distintas de lo visto hasta el momento, son entregas muy cortas con ejercicios similares; con 5 ejercicios de manipulación de strings en la tercera entrega y 5 ejercicios de E/S en la quinta. Las gráficas de estas entregas se muestran a continuación (Figuras 6.15, 6.16, 6.17, 6.18):

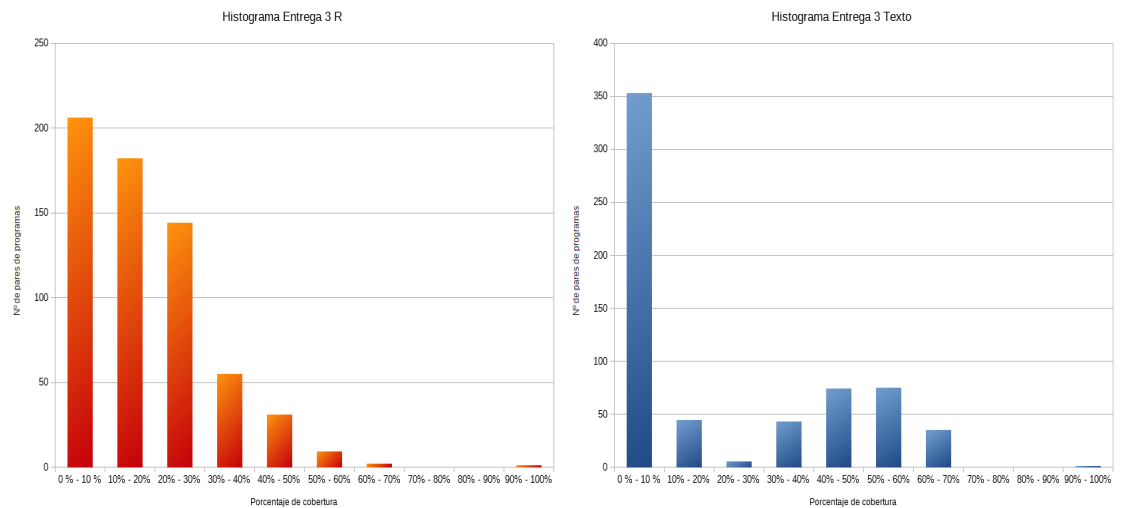


Figura 6.15: Histogramas de valores de similaridad entre todos los pares de programas de la tercera entrega.



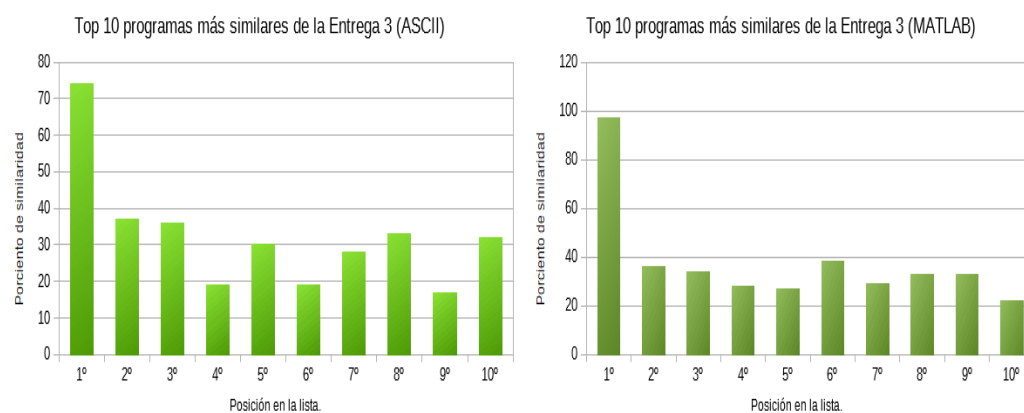


Figura 6.16: 10 primeros pares de programas con mayor cantidad de código en común de la entrega 3

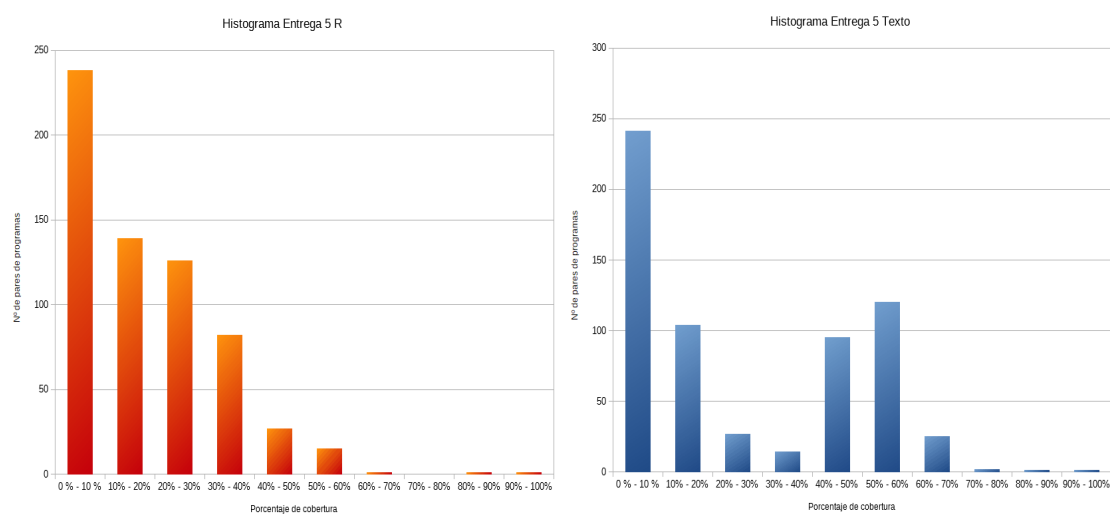


Figura 6.17: Histogramas de valores de similaridad entre todos los pares de programas de la quinta entrega.

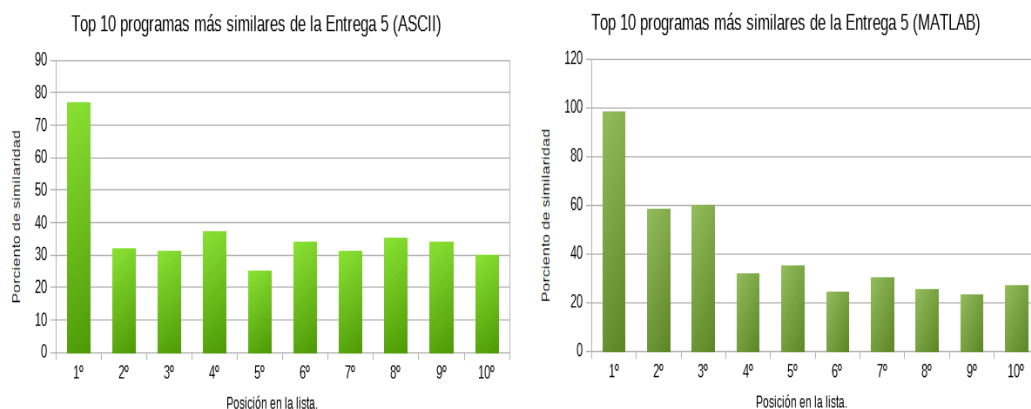


Figura 6.18: 10 primeros pares de programas con mayor cantidad de código en común de la entrega 5

Aún siendo ejercicios muy cortos los que se piden en estas entregas, estos se pueden abordar de numerosas formas, es por esto que los índices de similitud detectados con JPLAG y MOSS entre estos pares de archivos son los más bajos hasta el momento.

Los pares detectados en MOSS y en JPLAG con texto con un índice superior al 40 % se deben a programas de alumnos que han escrito el enunciado de los ejercicios en su código, y al ser tan pocas las líneas de código que se requieren para hacerlo, los comentarios suponen más de un 50 % del documento en la mayoría de los casos.

Por otra parte, los contados casos en los que nuestro frontend ha detectado un alto porcentaje de cobertura entre emparejamientos se deben en su mayoría a alumnos que se han copiado pero han intentado ocultarlo cambiando el nombre de las variables, el de las cadenas de strings usadas y el de los documentos usados en los ejercicios de E/S. Podemos ver un ejemplo de estos en Figura 6.19:

```
write.table(matrix(seq(from=0, to=18, by=2)), "dos.txt", row.names=FALSE, col.names=FALSE)
write.table(matrix(seq(from=0, to=27, by=3)), "tres.txt", row.names=FALSE, col.names=FALSE)
write.table(matrix(seq(from=0, to=45, by=5)), "cinco.txt", row.names=FALSE, col.names=FALSE)

d<-scan("dos.txt")
t<-scan("tres.txt")
c<-scan("cinco.txt")
nuevaMatriz<-matrix(c(d,t,c), ncol = 3, nrow = 18)
nuevaMatriz

write.table(nuevaMatriz[1:5,], "prime.txt", sep = ",", row.names=FALSE, col.names=FALSE)
write.table(nuevaMatriz[6:10,], "fin.txt", sep = ",", row.names=FALSE, col.names=FALSE)
```

```
write.table(matrix(meros*2, nrow=11, ncol=1), "/datasets/dos.txt", row.names=FALSE, col.names=FALSE)
write.table(matrix(meros*5, nrow=11, ncol=1), "/datasets/tres.txt", row.names=FALSE, col.names=FALSE)
write.table(matrix(meros*3, nrow=11, ncol=1), "/datasets/cinco.txt", row.names=FALSE, col.names=FALSE)

#####
# Carga los tres ficheros creados en el punto anterior y construye una matriz que, en cada
# columna, tengo el contenido de cada fichero.
#####

dos<-scan("/datasets/dos.txt")
tres<-scan("/datasets/tres.txt")
cinco<-scan("/datasets/cinco.txt")
```

Figura 6.19: Ejemplo de plagio entre alumnos en el que se ha cambiado el nombre de variables y archivos(Entrega 5)

Por último, en las entregas 6 y 7 se obtienen resultados similares a las entregas 3 y 5, ya que son igual de cortas, pero a diferencia de estas, tratan sobre ejercicios que requieren la definición de funciones.

Para declarar una función como las que se piden en los ejercicios de estas entregas el alumno usará sus propias estructuras de control, variables y cálculos matemáticos.

Debido a esto se obtienen las siguientes gráficas de resultados (Figuras 6.20, 6.21, 6.22, 6.23):

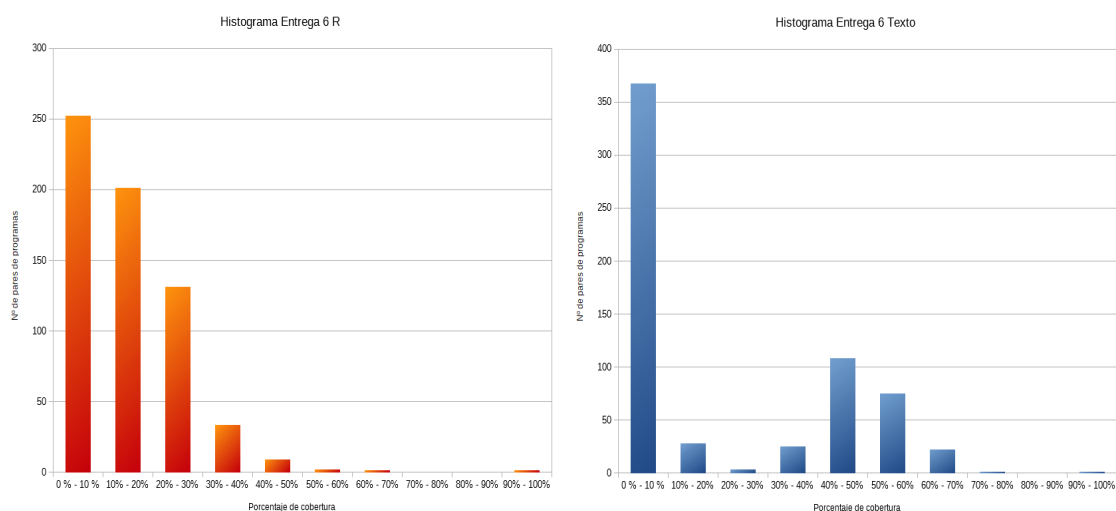


Figura 6.20: Histogramas de valores de similaridad entre todos los pares de programas de la sexta entrega.

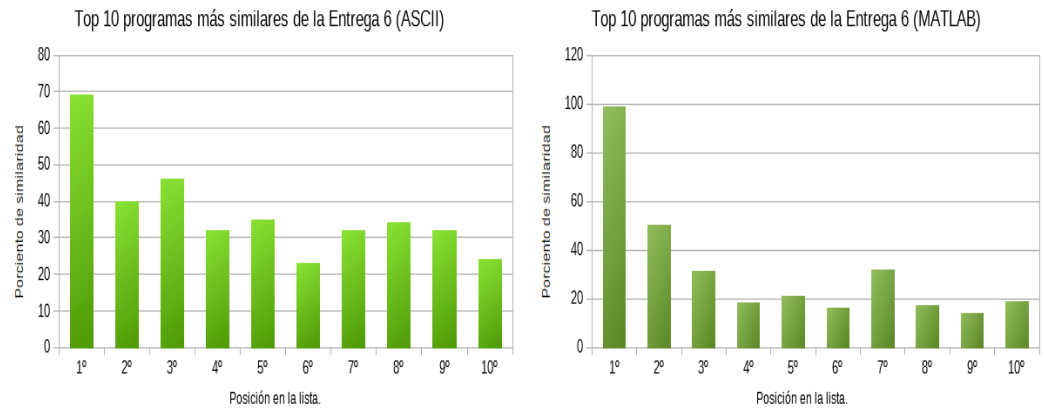


Figura 6.21: 10 primeros pares de programas con mayor cantidad de código en común de la entrega 6

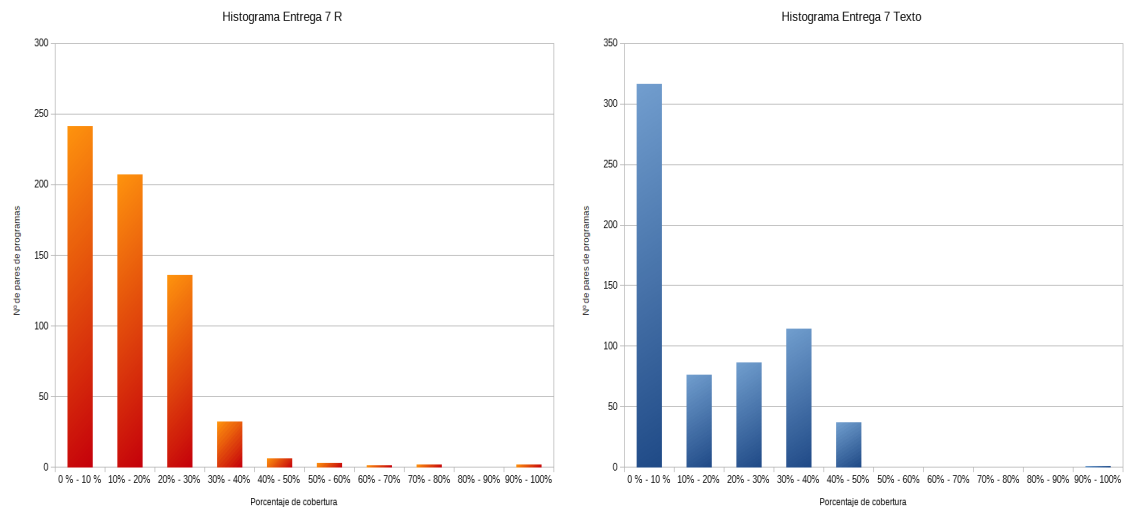


Figura 6.22: Histogramas de valores de similaridad entre todos los pares de programas de la séptima entrega.

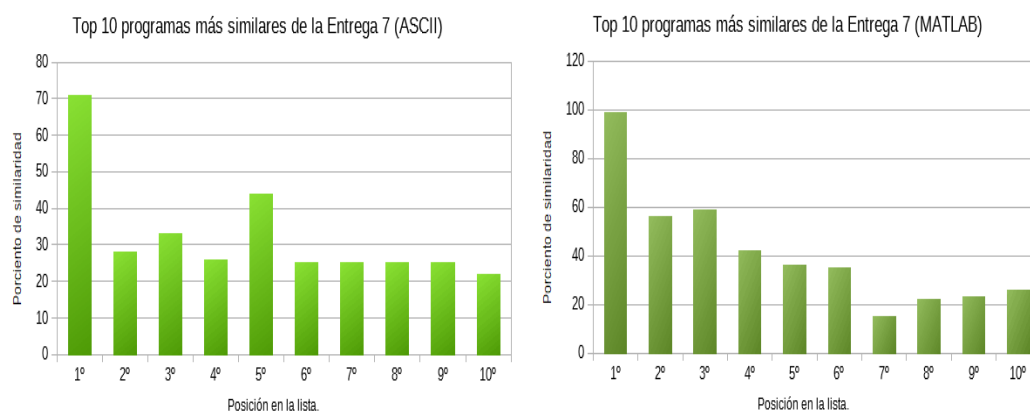


Figura 6.23: 10 primeros pares de programas con mayor cantidad de código en común de la entrega 7

Los índices de similaridad son aún más bajos, los altos se deben a comentarios iguales como en los casos anteriores y nuestro frontend sigue detectando los contados plagios debidos a estructuras de control casi iguales.

Hemos podido comprobar en los resultados obtenidos en las siete entregas que el frontend que hemos creado para JPLAG permite detectar:

1. Plagios en los que el código se ha copiado entero.
2. Plagios en los que sólo se ha cogido parte del código.
3. Plagios en los que se ha modificado parcialmente el código para ocultar la copia, ya sea cambiando el nombre de variables o usando estructuras de control equivalentes.

Pueden ocurrir falsos positivos con índice de similaridad alto, es por esto que los casos intermedios (entre 30 % y 60 % de similaridad) deben investigarse en mayor profundidad para comprobar si verdaderamente hay ocurrido plagio.

Para ello tan sólo tendremos que ir a las páginas de comparación de código donde se muestran los fragmentos donde se han detectado estructuras similares o iguales.

Los casos en los que ha ocurrido plagio y JPLAG con nuestro frontend no lo detecta requieren que el alumno haya modificado hasta tal punto la estructura del archivo que ya no se consideraría plagio, ya que esto significaría que el alumno entiende como funciona perfectamente el código que esta plagiando y sabe por tanto resolver el ejercicio.



## Capítulo 7

# Conclusiones y trabajo futuro

### 7.1. Conclusión

En este trabajo hemos buscado y estudiado los orígenes del plagio y los recursos disponibles para su detección motivados por resolver el problema de la copia entre documentos en R.

Dado que en nuestra búsqueda no encontramos ninguna forma verdaderamente efectiva de detectar el plagio en este lenguaje, nos decidimos a resolver el problema modificando una de las mejores herramientas de detección de plagio llamada JPLAG.

Para ello, como ya explicamos en los capítulos de Elección de Herramientas e Implementación, ha sido necesario crear un frontend, adaptar un analizador sintáctico, elegir los tokens más importantes de R y comunicar todo con el programa principal.

Hemos comparado entonces la nueva herramienta creada con las mejores opciones existentes para detectar la copia entre alumnos.

Los resultados obtenidos de esta comparación han sido satisfactorios ya que nuestra herramienta, al contrario que el resto, consigue identificar prácticamente todos los casos de plagio entre alumnos, cosa imposible hasta el momento a menos que el profesor o evaluador hiciese todas las comparaciones manualmente.

Se han cumplido por tanto todos los objetivos que especificamos en un principio en el capítulo de introducción (Sección 1.3).

### 7.2. Trabajo Futuro

Aunque el trabajo hecho permite que sea posible detectar la mayor parte de los plagios en R entre tareas de alumnos, hay diversos añadidos que se

pueden hacer para mejorar la herramienta:

- Basándonos en los detectores de plagio en documentos de texto como Turnitin y Viper, se podría añadir una base de datos de documentos para poder comparar los archivos enviados con archivos similares encontrados por internet o con las mismas entregas de alumnos de años anteriores. Para hacer esto se tendría que lidiar con la inmensa carga extra de trabajo que le supondría al algoritmo de comparación. En caso de poder solucionar ese problema, esto sería una gran mejora ya que permitiría saber si el alumno se ha copiado de un documento externo diferente al del resto de sus compañeros.
- Aunque JPLAG ya tenga soporte para los lenguajes más usados, estaría bien añadir soporte para más lenguajes.
- Entrenar una red neuronal con casos que son y no son plagio para que JPLAG pueda decir con un alto porcentaje de certeza si los casos que hasta el momento teníamos que inspeccionar con mayor profundidad (los casos intermedios) son plagio o no. Esto ahorraría un gran volumen de trabajo al evaluador.



# Bibliografía

- [1] Definición de plagio. <http://dle.rae.es/?id=TIZy4Xb>.
- [2] Plagiarism etymology. <https://www.etymonline.com/word/plagiarism>.
- [3] STUART P. GREEN. Plagiarism, norms, and the limits of theft law: Some observations on the use of criminal sanctions in enforcing intellectual property rights. [https://www.researchgate.net/publication/228244566\\_Plagiarism\\_Norms\\_and\\_the\\_Limits\\_of\\_Theft\\_Law\\_Some\\_Observations\\_on\\_the\\_Use\\_of\\_Criminal\\_Sanctions\\_in\\_Enforcing\\_Intellectual\\_Property\\_Rights](https://www.researchgate.net/publication/228244566_Plagiarism_Norms_and_the_Limits_of_Theft_Law_Some_Observations_on_the_Use_of_Criminal_Sanctions_in_Enforcing_Intellectual_Property_Rights), 2002.
- [4] Vinod K.R.\*, Sandhya.S, SathishKumar D, Harani A, Kumar D, David Banji, and Otilia JF Banji. Plagiarism: history, detection and prevention. <http://www.hygeiajournal.com/downloads/Editorial/1597787464plagiarism.pdf>, 2011.
- [5] Normativa de evaluación y de calificación de los estudiantes de la universidad de granada. <https://www.ugr.es/~minpet/pages/enpdf/normativaevaluacioncalificacion.pdf>.
- [6] Moss main website. <https://theory.stanford.edu/~aiken/moss/>.
- [7] Jplag official website, main page. <https://jplag.ipd.kit.edu/>.
- [8] The software and text similarity tester sim. [https://dickgrune.com/Programs/similarity\\_tester/](https://dickgrune.com/Programs/similarity_tester/).
- [9] Plaggie detection engine official website. <https://www.cs.hut.fi/Software/Plaggie/>.
- [10] R language definition. <https://www.r-project.org/>.
- [11] Agile methodology. <https://www.qasymphony.com/blog/agile-methodology-guide-agile-testing/>.
- [12] What is scrum. <https://www.scrum.org/resources/what-is-scrum>.

- [13] Online plagiarism checking with viper. <https://www.scanmyessay.com/plagiarism-check.php>.
- [14] Grammarly plagiarism checker. <https://www.grammarly.com/plagiarism>.
- [15] Ithenticate plagiarism detection software. <http://www.ithenticate.com/about>.
- [16] Plagscan plagiarism checker. <https://www.plagscan.com/en/>.
- [17] Turnitin use. <https://www.turnitin.com/blog/does-turnitin-detect-plagiarism>.
- [18] Maeve Paris. Source code and text plagiarism detection strategies. [https://www.researchgate.net/publication/228379807\\_Source\\_Code\\_and\\_Text\\_Plagiarism\\_Detection\\_Strategies](https://www.researchgate.net/publication/228379807_Source_Code_and_Text_Plagiarism_Detection_Strategies), 2014.
- [19] Sherlock - plagiarism detection software. <https://warwick.ac.uk/fac/sci/dcs/research/ias/software/sherlock/>.
- [20] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. Gplag: Detection of software plagiarism by program dependence graph analysis. 2006. [http://www1.se.cuhk.edu.hk/~hcheng/seg5010/slides/kdd06\\_gplag.pdf](http://www1.se.cuhk.edu.hk/~hcheng/seg5010/slides/kdd06_gplag.pdf).
- [21] Daniël Heres. Source code plagiarism detection using machine learning. pages 1–9, 2017.
- [22] Michael J. Wise. Yap3: Improved detection of similarities in computer program and other texts. 1996. [https://www.researchgate.net/publication/2249233\\_YAP3\\_Improved\\_Detection\\_Of\\_Similarities\\_In\\_Computer\\_Program\\_And\\_Other\\_Texts](https://www.researchgate.net/publication/2249233_YAP3_Improved_Detection_Of_Similarities_In_Computer_Program_And_Other_Texts).
- [23] Kevin W. Bowyer and Lawrence O. Hall. Experience using "moss" to detect cheating on programming assignments. 2017. <https://www3.nd.edu/~kwb/nsf-ufe/1110.pdf>.
- [24] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: Local algorithms for document fingerprinting. <http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf>, 2003.
- [25] Jplag github repository. <https://github.com/jplag/jplag>.
- [26] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. Jplag: Finding plagiarisms among a set of programs, 2000.
- [27] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. <https://pdfs.semanticscholar.org/c47d/151f09c567013761632c89e237431c6291a2.pdf>, 1987.

- 
- [28] Moss credits page. <http://moss.stanford.edu/general/credits.html>.
  - [29] Apache maven official website. <https://maven.apache.org/>.
  - [30] Terence Parr. *The Definitive ANTLR 4 Reference*. The Pragmatic Programmers, 1 edition, 2012.
  - [31] R base grammar github repository. <https://github.com/antlr/grammars-v4>.
  - [32] R language definition. <https://cran.r-project.org/doc/manuals/r-release/R-lang.html>.
  - [33] Getting started with moss. <https://gist.github.com/danielmai/9162349>.
  - [34] Installing apache maven. <https://maven.apache.org/install.html>.



# Apéndice A

## Código

### A.1. RTokenConstants.java

```
1 package jplag.R;
2
3 /*
4 Tokens que consideramos importantes de R a la hora de analizar si hay
5 plagio entre entregas de estudiantes.
6 */
7
8 public interface RTokenConstants extends jplag.TokenConstants {
9     final static int FILE_END = 0;
10    final static int SEPARATOR_TOKEN = 1;
11
12    final static int BEGIN_FUNCTION = 2;
13    final static int END_FUNCTION = 3;
14    final static int FUNCTION_CALL = 4;
15    final static int NUMBER = 5;
16    final static int STRING = 6;
17    final static int BOOL = 7;
18    final static int ASSIGN = 8;
19    final static int ASSIGN_FUNC = 9;
20    final static int ASSIGN_LIST = 10;
21    final static int HELP = 11;
22    final static int INDEX = 12;
23    final static int PACKAGE = 13;
24    final static int IF_BEGIN = 14;
25    final static int IF_END = 15;
26    final static int FOR_BEGIN = 16;
27    final static int FOR_END = 17;
28    final static int WHILE_BEGIN = 18;
29    final static int WHILE_END = 19;
30    final static int REPEAT_BEGIN = 20;
31    final static int REPEAT_END = 21;
32    final static int NEXT = 22;
33    final static int BREAK = 23;
34    final static int COMPOUND_BEGIN = 24;
35    final static int COMPOUND_END = 25;
36
37    final static int NUM_DIFF_TOKENS = 26;
```

39 }

## A.2. RToken.java

```

1 package jplag.R;
2
3
4 public class RToken extends jplag.Token implements RTokenConstants {
5     private static final long serialVersionUID = 1L;
6     private int line, column, length;
7
8     public RToken(int type, String file, int line, int column, int
9         length) {
10         super(type, file, line, column, length);
11     }
12
13     // Gets y sets
14
15     public int getLine() {
16         return line;
17     }
18
19     public int getColumn() {
20         return column;
21     }
22
23     public int getLength() {
24         return length;
25     }
26
27     public void setLine(int line) {
28         this.line = line;
29     }
30
31     public void setColumn(int column) {
32         this.column = column;
33     }
34
35     public void setLength(int length) {
36         this.length = length;
37     }
38
39     // Token en String.
40     public static String type2string(int type) {
41         switch (type) {
42             case RTokenConstants.FILE_END:
43                 return "*****";
44             case RTokenConstants.SEPARATOR_TOKEN:
45                 return "METHOD_SEPARATOR";
46             case BEGIN_FUNCTION:
47                 return "FUNCTION{ ";
48             case END_FUNCTION:
49                 return "}FUNCTION ";
50             case FUNCTION_CALL:
51                 return "FUNCTION() ";
52             case NUMBER:
53                 return "NUMBER ";

```

```
54         case STRING:
55             return "STRING    ";
56     case BOOL:
57         return "BOOL        ";
58         case ASSIGN:
59             return "ASSIGN     ";
60     case ASSIGN_FUNC:
61         return "ASSIGN_FUNC";
62     case ASSIGN_LIST:
63         return "ASSIGN_LIST";
64         case HELP:
65             return "HELP       ";
66     case INDEX:
67         return "INDEX        ";
68         case PACKAGE:
69             return "PACKAGE    ";
70     case IF_BEGIN:
71         return "IF{         ";
72     case IF_END:
73         return "}IF        ";
74         case FOR_BEGIN:
75             return "FOR{       ";
76     case FOR_END:
77         return "}FOR        ";
78         case WHILE_BEGIN:
79             return "WHILE{     ";
80     case WHILE_END:
81         return "}WHILE     ";
82     case REPEAT_BEGIN:
83         return "REPEAT{    ";
84         case REPEAT_END:
85             return "}REPEAT   ";
86     case NEXT:
87         return "NEXT        ";
88         case BREAK:
89             return "BREAK      ";
90     case COMPOUND_BEGIN:
91         return "COMPOUND{   ";
92     case COMPOUND_END:
93         return "}COMPOUND  ";
94     default:
95         System.err.println("*UNKNOWN: " + type);
96         return "*UNKNOWN" + type;
97     }
98 }
99
100 public static int numberOfTokens() {
101     return NUM_DIFF_TOKENS;
102 }
103 }
```

### A.3. Language.java

```
1 package jplag.R;
2
3 import java.io.File;
4
5 import jplag.ProgramI;
```

```
6
7
8  /*
9  En este archivo definimos lo que JPLAG tiene que saber del lenguaje
   cuyo frontend estamos haciendo: los sufijos de los archivos de
   este lenguaje, nombre que le ponemos al parser,
10 el numero minimo de tokens iguales que se considere un emparejamiento
   ,...
11 */
12
13 public class Language implements jplag.Language {
14     private Parser parser;
15
16     public Language(ProgramI program) {
17         this.parser = new Parser();
18         this.parser.setProgram(program);
19     }
20
21     public String[] suffixes() {
22         String[] res = { ".r", ".R" };
23         return res;
24     }
25
26     public int errorsCount() {
27         // TODO Auto-generated method stub
28         return this.parser.errorsCount();
29     }
30
31     public String name() {
32         return "R Parser";
33     }
34
35     public String getShortName() {
36         return "R";
37     }
38
39     public int min_token_match() {
40         return 8;
41     }
42
43     public jplag.Structure parse(File dir, String[] files) {
44         return this.parser.parse(dir, files);
45     }
46
47     public boolean errors() {
48         return parser.getErrors();
49     }
50
51     public boolean supportsColumns() {
52         return true;
53     }
54
55     public boolean isPreformatted() {
56         return true;
57     }
58
59     public boolean usesIndex() {
60         return false;
61     }
62
63     public int noOfTokens() {
```



```
65         return jplag.R.RToken.numberOfTokens();
66     }
67
68     public String type2string(int type) {
69         return jplag.R.RToken.type2string(type);
70     }
71 }
```

## A.4. JplagRListener.java

```
1 package jplag.R;
2
3 import jplag.R.grammar.*;
4 import org.antlr.v4.runtime.ParserRuleContext;
5 import org.antlr.v4.runtime.misc.NotNull;
6 import org.antlr.v4.runtime.tree.ErrorNode;
7 import org.antlr.v4.runtime.tree.TerminalNode;
8
9 /*
10 Esta clase se encarga de implementar los metodos clase RListener que
11 se genera por ANTLR4 de forma que cada vez que se reconozca un
12 Token del lenguaje
13 se entrara en su "enter<NombredelToken>" y "exit<NombredelToken>" y en
14 caso de ser un token importante (es decir, que sea relevante
15 considerarlo a la hora de identificar plagio)
16 tan solo tendremos que irnos a su metodo he implementarlo haciendo un
17 jplagParser.add o un jplagParser.addEnd (en caso de que sea un
18 exit) con el nombre del token que le hayamos
19 asociado en RToken.java y RTokenConstants.java .
20 */
21
22 public class JplagRListener implements RListener, RFilterListener,
23     RTokenConstants {
24
25     private jplag.R.Parser jplagParser;
26
27     public JplagRListener(jplag.R.Parser jplag) {
28         jplagParser = jplag;
29     }
30
31     @Override
32     public void enterProg(RParser.ProgContext ctx){
33
34     }
35
36     @Override
37     public void exitProg(RParser.ProgContext ctx){
38
39     }
40
41     @Override
42     public void enterExpr(RParser.ExprContext ctx){
43
44     }
45
46     @Override
47     public void exitExpr(RParser.ExprContext ctx){
48
49     }
50 }
```

```

42     }
43
44     @Override
45     public void enterIndex_statement(RParser.
46         Index_statementContext ctx){
47         jplagParser.add(INDEX, ctx.getStart());
48     }
49
50     @Override
51     public void exitIndex_statement(RParser.Index_statementContext
52         ctx){
53     }
54
55     @Override
56     public void enterAccess_package(RParser.Access_packageContext
57         ctx){
58         jplagParser.add(PACKAGE, ctx.getStart());
59     }
60
61     @Override
62     public void exitAccess_package(RParser.Access_packageContext
63         ctx){
64     }
65
66     @Override
67     public void enterFunction_definition(RParser.
68         Function_definitionContext ctx){
69         jplagParser.add(BEGIN_FUNCTION, ctx.getStart());
70     }
71
72     @Override
73     public void exitFunction_definition(RParser.
74         Function_definitionContext ctx){
75         jplagParser.addEnd(END_FUNCTION, ctx.getStart());
76     }
77
78     @Override
79     public void enterFunction_call(RParser.Function_callContext
80         ctx){
81         jplagParser.add(FUNCTION_CALL, ctx.getStart());
82     }
83
84     @Override
85     public void exitFunction_call(RParser.Function_callContext ctx
86         ){
87     }
88
89     @Override
90     public void enterConstant(RParser.ConstantContext ctx){
91     }
92
93     @Override
94     public void exitConstant(RParser.ConstantContext ctx){

```

```
95     public void enterConstant_number(RParser.  
96         Constant_numberContext ctx){  
97         jplagParser.add(NUMBER, ctx.getStart());  
98     }  
99     @Override  
100     public void exitConstant_number(RParser.Constant_numberContext  
101         ctx){  
102     }  
103     @Override  
104     public void enterConstant_string(RParser.  
105         Constant_stringContext ctx){  
106         jplagParser.add(String, ctx.getStart());  
107     }  
108     @Override  
109     public void exitConstant_string(RParser.Constant_stringContext  
110         ctx){  
111     }  
112     @Override  
113     public void enterConstant_bool(RParser.Constant_boolContext  
114         ctx){  
115         jplagParser.add(BOOL, ctx.getStart());  
116     }  
117     @Override  
118     public void exitConstant_bool(RParser.Constant_boolContext ctx  
119         ){  
120     }  
121     @Override  
122     public void enterHelp(RParser.HelpContext ctx){  
123         jplagParser.add(HELP, ctx.getStart());  
124     }  
125     @Override  
126     public void exitHelp(RParser.HelpContext ctx){  
127     }  
128     @Override  
129     public void enterIf_statement(RParser.If_statementContext ctx)  
130     {  
131         jplagParser.add(IF_BEGIN, ctx.getStart());  
132     }  
133     @Override  
134     public void exitIf_statement(RParser.If_statementContext ctx){  
135         jplagParser.addEnd(IF_END, ctx.getStart());  
136     }  
137     @Override  
138     public void enterFor_statement(RParser.For_statementContext  
139         ctx){  
140         jplagParser.add(FOR_BEGIN, ctx.getStart());  
141     }  
142     @Override  
143     public void exitFor_statement(RParser.For_statementContext  
144         ctx){  
145         jplagParser.addEnd(FOR_END, ctx.getStart());  
146     }  
147     @Override  
148     public void exitFor_statement(RParser.For_statementContext ctx){
```

```
149     @Override
150     public void exitFor_statement(RParser.For_statementContext ctx
151     ){
152         jplagParser.addEnd(FOR_END, ctx.getStart());
153     }
154     @Override
155     public void enterWhile_statement(RParser.
156         While_statementContext ctx){
157         jplagParser.add(WHILE_BEGIN, ctx.getStart());
158     }
159     @Override
160     public void exitWhile_statement(RParser.While_statementContext
161         ctx){
162         jplagParser.addEnd(WHILE_END, ctx.getStart());
163     }
164     @Override
165     public void enterRepeat_statement(RParser.
166         Repeat_statementContext ctx){
167         jplagParser.add(REPEAT_BEGIN, ctx.getStart());
168     }
169     @Override
170     public void exitRepeat_statement(RParser.
171         Repeat_statementContext ctx){
172         jplagParser.addEnd(REPEAT_END, ctx.getStart());
173     }
174     @Override
175     public void enterNext_statement(RParser.Next_statementContext
176         ctx){
177         jplagParser.add(NEXT, ctx.getStart());
178     }
179     @Override
180     public void exitNext_statement(RParser.Next_statementContext
181         ctx){
182     }
183     @Override
184     public void enterBreak_statement(RParser.
185         Break_statementContext ctx){
186         jplagParser.add(BREAK, ctx.getStart());
187     }
188     @Override
189     public void exitBreak_statement(RParser.Break_statementContext
190         ctx){
191     }
192     @Override
193     public void enterCompound_statement(RParser.
194         Compound_statementContext ctx){
195         jplagParser.add(COMPOUND_BEGIN, ctx.getStart());
196     }
197     @Override
198     public void enterCompound_statement(RParser.
199         Compound_statementContext ctx){
```

```
200     public void exitCompound_statement(RParser.  
201         Compound_statementContext ctx){  
202         jplagParser.addEnd(COMPOUND_END, ctx.getStart());  
203     }  
204     @Override  
205     public void enterExprlist(RParser.ExprlistContext ctx){  
206     }  
207     @Override  
208     public void exitExprlist(RParser.ExprlistContext ctx){  
209     }  
210     @Override  
211     public void enterFormlist(RParser.FormlistContext ctx){  
212     }  
213     @Override  
214     public void exitFormlist(RParser.FormlistContext ctx){  
215     }  
216     @Override  
217     public void enterForm(RParser.FormContext ctx){  
218     }  
219     @Override  
220     public void exitForm(RParser.FormContext ctx){  
221     }  
222     @Override  
223     public void enterSublist(RParser.SublistContext ctx){  
224     }  
225     @Override  
226     public void exitSublist(RParser.SublistContext ctx){  
227     }  
228     @Override  
229     public void enterSub(RParser.SubContext ctx){  
230     }  
231     @Override  
232     public void exitSub(RParser.SubContext ctx){  
233     }  
234     @Override  
235     public void enterAssign_value(RParser.Assign_valueContext ctx)  
236     {  
237         jplagParser.add(ASSIGN, ctx.getStart());  
238     }  
239     @Override
```

```
260     public void exitAssign_value(RParser.Assign_valueContext ctx){
261     }
262
263     @Override
264     public void enterAssign_func_declaration(RParser.
265         Assign_func_declarationContext ctx){
266         jplagParser.add(ASSIGN_FUNC, ctx.getStart());
267     }
268
269     @Override
270     public void exitAssign_func_declaration(RParser.
271         Assign_func_declarationContext ctx){
272     }
273
274     @Override
275     public void enterAssign_value_list(RParser.
276         Assign_value_listContext ctx){
277         jplagParser.add(ASSIGN_LIST, ctx.getStart());
278     }
279
280     @Override
281     public void exitAssign_value_list(RParser.
282         Assign_value_listContext ctx){
283     }
284
285     @Override
286     public void enterStream(RFilter.StreamContext ctx){
287     }
288
289     @Override
290     public void exitStream(RFilter.StreamContext ctx){
291     }
292
293     @Override
294     public void enterEat(RFilter.EatContext ctx){
295     }
296
297     @Override
298     public void exitEat(RFilter.EatContext ctx){
299     }
300
301     @Override
302     public void enterElem(RFilter.ElemContext ctx){
303     }
304
305     @Override
306     public void exitElem(RFilter.ElemContext ctx){
307     }
308
309     @Override
310     public void enterAtom(RFilter.AtomContext ctx){
311     }
312
313     @Override
314     public void enterAtom(RFilter.AtomContext ctx){
315     }
316
317     @Override
318     public void enterAtom(RFilter.AtomContext ctx){
319     }
```

```
318     }
319
320     @Override
321     public void exitAtom(RFilter.AtomContext ctx){
322
323     }
324
325     @Override
326     public void enterOp(RFilter.OpContext ctx){
327
328     }
329
330     @Override
331     public void exitOp(RFilter.OpContext ctx){
332
333     }
334
335
336     @Override
337     public void enterEveryRule(ParserRuleContext ctx){
338
339     }
340
341     @Override
342     public void exitEveryRule(ParserRuleContext ctx){
343
344     }
345
346     @Override
347     public void visitTerminal(TerminalNode node){
348         if (node.getText().equals("=")) {
349             jplagParser.add(ASSIGN, node.getSymbol());
350         }
351     }
352
353     @Override
354     public void visitErrorNode(ErrorNode node){
355
356     }
357 }
```

## A.5. Parser.java

```
1 package jplag.R;
2
3 import java.io.BufferedInputStream;
4 import java.io.BufferedReader;
5 import java.io.File;
6 import java.io.FileInputStream;
7 import java.io.FileReader;
8 import java.io.IOException;
9
10 import jplag.Structure;
11 import jplag.R.grammar.RLexer;
12 import jplag.R.grammar.RParser;
13 import jplag.R.grammar.RParser.ProgContext;
14
15
```

```

16 import org.antlr.v4.runtime.ANTLRInputStream;
17 import org.antlr.v4.runtime.CommonTokenStream;
18 import org.antlr.v4.runtime.tree.ParseTree;
19 import org.antlr.v4.runtime.tree.ParseTreeWalker;
20
21 /*
22 Esta clase se encarga de pasar el analizador lexico y sintactico
    creado en ANTLR4 y mandar los tokens (que decidimos en
    JplagRListener) al programa principal
23 para que aplique el algoritmo.
24 */
25
26
27 public class Parser extends jplag.Parser implements RTokenConstants {
28     private Structure struct;
29     private String currentFile;
30
31     public static void main(String args[]) {
32         if (args.length < 1) {
33             System.out.println("Only one or more files as
                parameter allowed.");
34             System.exit(-1);
35         }
36         Parser parser = new Parser();
37         parser.setProgram(new jplag.StrippedProgram());
38         jplag.Structure struct = parser.parse(null, args);
39         try {
40             BufferedReader reader = new BufferedReader(new FileReader(
                new File(args[0])));
41             int lineNr = 1;
42             int token = 0;
43             String line;
44             while ((line = reader.readLine()) != null) {
45                 if (token < struct.size()) {
46                     boolean first = true;
47                     while (struct.tokens[token] != null
                        && struct.tokens[token].getLine() ==
                            lineNr) {
48                         if (!first) {
49                             System.out.println();
50                         }
51                         RToken tok = (RToken) struct.tokens[token];
52                         System.out.print(RToken.type2string(tok.type)
53                             + " ("
54                                 + tok.getLine() + ", "
55                                 + tok.getColumn() + ", "
56                                 + tok.getLength() + ")\t");
57                         first = false;
58                         token++;
59                     }
60                     if (first) {
61                         System.out.print("
62                             \t");
63                     }
64                     System.out.println(line);
65                     lineNr++;
66                 }
67                 reader.close();
68             } catch (IOException e) {
69                 System.out.println(e);
70             }
71         }

```



```
72
73     public jplag.Structure parse(File dir, String files[]) {
74         struct = new Structure();
75         errors = 0;
76         for (int i = 0; i < files.length; i++) {
77             getProgram().print(null, "Parsing file " + files[i] + "\n
78             ");
79             if (!parseFile(dir, files[i])) {
80                 errors++;
81             }
82             System.gc(); // Emeric
83             struct.addToken(new RToken(FILE_END, files[i], -1, -1, -1)
84             );
85         }
86         this.parseEnd();
87         return struct;
88     }
89
90     public boolean parseFile(File dir, String file) {
91         BufferedInputStream fis;
92
93         ANTLRInputStream input;
94         try {
95             fis = new BufferedInputStream(new FileInputStream(new File
96             (dir, file)));
97             currentFile = file;
98             input = new ANTLRInputStream(fis);
99
100             // create a lexer that feeds off of input CharStream
101             RLexer lexer = new RLexer(input);
102
103             // create a buffer of tokens pulled from the lexer
104             CommonTokenStream tokens = new CommonTokenStream(lexer);
105
106             // create a parser that feeds off the tokens buffer
107             RParser parser = new RParser(tokens);
108             ProgContext in = parser.prog();
109
110             ParseTreeWalker ptw = new ParseTreeWalker();
111             for (int i = 0; i < in.getChildCount(); i++) {
112                 ParseTree pt = in.getChild(i);
113                 ptw.walk(new JplagRListener(this), pt);
114             }
115
116             } catch (IOException e) {
117                 getProgram().addError("Parsing Error in '" + file + "':\n"
118                 + e.getMessage() + "\n");
119                 return false;
120             }
121
122             return true;
123         }
124
125     public void add(int type, org.antlr.v4.runtime.Token tok) {
126         struct.addToken(new RToken(type, (currentFile == null ? "null"
127         : currentFile), tok.getLine(), tok.getCharPositionInLine
128         () + 1,
129         tok.getText().length()));
130     }
131
132     public void addEnd(int type, org.antlr.v4.runtime.Token tok) {
```

```
127         struct.addToken(new RToken(type, (currentFile == null ? "null"  
128             : currentFile), tok.getLine(), struct.tokens[struct.size  
129             ()-1].getColumn() + 1,0));  
    }  
}
```