Herramienta de apoyo en la detección de reutilización de código fuente

Raymundo Picazo-Alvarez, Esaú Villatoro-Tello,
Wulfrano A. Luna-Ramírez, Carlos R. Jaimez-González

Departamento de Tecnologías de la Información,
División de Ciencias de la Comunicación y Diseño,
Universidad Autónoma Metropolitana, Unidad Cuajimalpa, México D.F.

207363142@alumnos.cua.uam.mx,
{evillatoro, wluna, cjaimez}@correo.cua.uam.mx

Resumen. El acto de tomar parcial o totalmente contenidos generados por otras personas, y presentarlos como propios, sin dar el crédito correspondiente a los autores, es una forma indebida de reutilización de contenidos, considerada como plagio. Desafortunamente, en la actualidad, dada la amplia disponibilidad de contenidos a través de Internet, esta práctica se ha incrementado. La gran mayoría de los contenidos disponibles en la Web son materiales multimedia, aplicaciones y sobre todo textos, y todos ellos son susceptibles de plagio. En este artículo se hace énfasis en una clase de textos en particular: los programas escritos en algún lenguaje de programación, denominados código fuente. Dada la facilidad de acceso y las prácticas de reutilización de contenidos sin citar las fuentes (el abuso de la posibilidad de "*Copiar y Pegar*", derivado de deficiencias metodológicas o bien como acción deliberada), surge la necesidad de contar con herramientas para combatir el plagio, en especial, de código fuente. En el presente trabajo se propone una herramienta orientada a detectar la reutilización de código fuente en programas escritos en un mismo lenguaje de programación. Las técnicas aplicadas se basan en la detección de la similitud entre dos programas, a través del uso de su **Frecuencia de Términos** (TF) y su **Frecuencia Inversa** (TF-IDF), considerando como términos conjuntos de n-gramas de caracteres presentes en cada uno de ellos.

Palabras clave: n-gramas de caracteres, representación vectorial, similitud de documentos, reutilización de código fuente, procesamiento del lenguaje natural.

1. Introducción

La disponibilidad de grandes cantidades de información a través del acceso a Internet permite a millones de usuarios consultar información y materiales muy diversos. La cantidad de información accesible está en constante crecimiento, y se ha acelerado con la denominada Web 2.0, que permite a los usuarios la producción y publicación de materiales de distinta naturaleza. Esto ha sido posible, entre otras cosas, por la facilidad de reproducir y reutilizar contenidos ya existentes en formato digital. Sin embargo, muchas de estas reproducciones

son copias no autorizadas y la reutilización de una parte o su totalidad, frecuentemente conduce a prácticas de plagio y violación de las leyes de derechos de autor[1], pues no se citan las fuentes ni se toman en cuenta, pese a su existencia, las restricciones de autoría. Por esta razón, se han desarrollado distintas herramientas que intentan identificar la autoría de los materiales y, como consecuencia, el posible plagio de la información.

La tarea antes descrita enfrenta múltiples dificultades; en primer lugar, existen muchas definiciones de plagio, aunque es difícil nombrar a una como la más acertada [1]; sin embargo, coinciden en señalar que plagio es tomar ideas de otros y presentarlas como propias sin dar el crédito correspondiente al autor [1]. Un ejemplo de acciones de esta índole en el área de Tecnologías de la Información (TI), es el caso de la demanda entre empresas interpuesta por Oracle en contra de Google por plagio de código en el sistema operativo Android para su producto Smartphone[2].

En el ámbito universitario, en particular en las carreras de TI, merece especial atención, pues las prácticas de plagio son altamente lesivas para la actividad académica y debido a la abundancia de información disponible aunada a la facilidad de su reutilización, requiere el desarrollo de aplicaciones y procedimientos que las eviten. Lo anterior, dada su recurrencia y la afectación que provoca, motivó la realización de una herramienta académica auxiliar para la detección de la reutilización de código fuente monolingüe, es decir, dentro de un mismo lenguaje de programación (susceptible de ser evaluado como plagio a juicio del usuario experto, en este caso, un profesor de programación).

Diferentes autores han planteado variadas estrategias para detectar la reutilización de código fuente, como Faidhi y Robinson [2] en los que se basa parte del presente trabajo. Este último realizó una clasificación de seis niveles y/o tipos de plagio, según la dificultad que representa para el programador ocultar que está reutilizando código fuente. En la Figura 1 se muestran los diferentes niveles de plagio.

- **Nivel 0**. Es la copia exacta del código fuente.
- **Nivel 1**. Es la inserción, borrado, modificación de comentarios junto a la indentación, saltos de línea y espacios en blanco.
- **Nivel 2**. Se centra en los identificadores, ya sea cuando se cambian los nombres o se ponen en mayúsculas.
- **Nivel 3**. Consiste en cambiar de posición declaraciones, ya sean variables o funciones de una parte del código a otra; también añadir variables o funciones que no se usarán.
- **Nivel 4**. Resulta de la combinación y división de funciones.
- **Nivel 5**. Consiste en el cambio de estructuras de control del código fuente.
- **Nivel 6**. Realiza cambios en las expresiones contenidas en el programa.

La herramienta propuesta en este trabajo es capaz de identificar reutilización de código hasta el nivel 3. En este tenor, debe señalarse también que hay poca

[1] Disponible en: http://www.diputados.gob.mx/LeyesBiblio/pdf/122.pdf

[2] Disponible en: http://www.informationweek.com/software/operating-systems/oracle-appeals-android-lawsuit/d/d-id/1106694

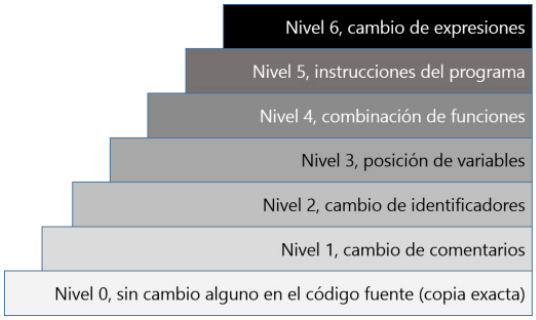

Fig. 1. Niveles de complejidad en la detección de plagio propuesta por Faidhi and Robinson [2].

información sobre desarrollos de herramientas en México para la detección de reutilización de textos en general y mucho menos de código fuente. De este modo, se propone contar con una herramienta desarrollada en nuestro país y contribuir así a su desarrollo tecnológico.

El resto del artículo está organizado de la siguiente manera. En la sección 2 se presentan algunos antecedentes y el estado del arte; la sección 3 describe el método propuesto en nuestra herramienta; la arquitectura de la herramienta se muestra en la sección 4. Finalmente, en la sección 5 se presentan las conclusiones, y el trabajo futuro.

2. Antecedentes y estado del arte

Los sistemas de detección automática de reutilización de código (que potencialmente es plagio) [1,3] nacieron debido a la imposibilidad de evitar el plagio por parte de organismos, tales como comisiones éticas en empresas y universidades. Por este motivo se desarrollaron varios modelos para la detección automática de reutilización de textos y de código fuente, los cuales pueden distinguirse en dos clases: la detección de plagio monolingüe, la cual detecta el plagio entre documentos pertenecientes al mismo lenguaje; y la detección de plagio translingüe, la cual es capaz de detectar el plagio entre varios lenguajes [4].

Estos modelos van más allá del análisis de códigos fuente completos para determinar si han sido escritos por un autor determinado, al analizar sus fragmentos, para intentar identificar que realmente fue escrito por el autor que lo presenta como propio [5].

En la literatura referente a la detección de reutilización existen dos enfoques que se han usado ampliamente para la detección de plagio: el **análisis intrínseco** y el **análisis extrínseco**. Los sistemas intrínsecos, tratan de identificar qué partes de un mismo código pertenecen a otro autor sin la necesidad de recurrir a fuentes externas. La idea intuitiva de estos sistemas es que cada autor/programador tiene estilos muy particulares, entonces, donde hay un cambio de estilo de programación se pre supone la existencia de un bloque que podría

pertenecer a otro autor. Por otro lado, los sistemas extrínsecos cuentan con una colección de códigos fuente confiables contra la cual se compara el código sospechoso. De esta manera, tratan de detectar si alguno de los códigos fuente confiables se han reutilizado o incluso si ha sido reutilizado el código completo de alguno o varios de ellos [6,7].

2.1. Técnicas para la detección de reutilización de código fuente

Existen dos técnicas principales en la comunidad científica para el análisis en la detección de reutilización de código fuente, las cuales se basan en ideas extraídas del área de Procesamiento de Lenguaje Natural, la cual es una sub disciplina de la Inteligencia Artificial. La primera de ellas, se basa en la comparación de características del propio código fuente, como son: el número de líneas, el número de palabras y caracteres, las líneas indentadas, los saltos de línea, la cantidad y tipos de *tokens*[3] de un código fuente, entre otras [6,7].

La segunda técnica, consiste en comparar la estructura del código fuente mediante un análisis sintáctico del documento. Algunos de los elementos del análisis son: las instrucciones, expresiones, asignaciones, identificadores, etc. Esta estructura se representa generalmente en forma de un árbol de ejecución; el cual se recorre en post orden, es decir, representando los nodos terminales (hojas) como 0 y los nodos internos (ramas) como 1. Este recorrido genera una cadena binaria que representa un perfil del árbol de ejecución; posteriormente con algoritmos de búsqueda de subcadenas comunes, se pueden identificar rápidamente partes en coincidentes entre dos árboles de ejecución representados de esta forma [6,7]. Esta técnica detecta la reutilización de código fuente a pesar de que se intente engañar al detector mediante técnicas de paráfrasis (la reformulación del fragmento reutilizado) o bien si se ha producido un resumen [4].

Técnica basada en Bolsa de Palabras.

Esta técnica utiliza vectores de características de código fuente, las cuales pueden ser palabras o n-gramas de palabras (tuplas de palabras o caracteres según su secuencia de aparición en el texto o en la palabra respectivamente). Una de las técnicas basadas en bolsa de palabras es el cociente de Jaccard, el cual consiste en dividir el tamaño de la intersección de dos vectores de características, entre el tamaño de su unión, como se muestra la Fórmula 1.

$$\mathcal{J}(A, B) = \frac{A \cap B}{A \cup B} \tag{1}$$

Donde $\mathcal{J}$ representa la probabilidad de reutilización del documento, A al documento original y B el documento sospechoso. $\mathcal{J}$ está acotado en [0, 1] siendo 1 cuando A y B son idénticos [4].

[3] Un token es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación, por ejemplo podría ser una palabra clave como: *while, print, if, for.*

Técnica basada en Huella Digital.
El concepto de huella digital, consiste en generar, a partir de fragmentos de un documento, una representación a modo de resumen de lo que éste contiene. Un ejemplo es el uso de una función *hash*, la cual a partir de una secuencia de caracteres, obtiene un número único y que al realizar la mínima modificación de la secuencia, la función devuelve un número distinto [6,7]. A continuación se describen los pasos para obtener la huella digital de acuerdo al trabajo propuesto en [9].

1. Se divide el documento en n-gramas.
2. A cada n-gama, se le aplica una función que permita obtener un valor único. Para ello, generalmente se aplica una función *hash*.
3. De todo el conjunto de valores *hash* obtenidos, se selecciona un pequeño grupo que representa a todo el documento en el proceso de similitud. Cada valor del pequeño grupo seleccionado se vuelve una huella digital del documento.
4. Se elabora un índice invertido con las huellas digitales obtenidas. Para poder encontrar todos los documentos que poseen similitud con un documento d_j, primero se leen todas las listas invertidas de la huella digital del documento d_j; posteriormente se mezclan estas listas, y finalmente se aplica una regla de verificación especificada.

Como se puede observar, el primer paso es muy importante pues consiste en seleccionar n, es decir el tamaño de los n-gramas, nótese que un número muy grande, como el de todo el documento, permitiría únicamente detectar copias exactas; mientras que un tamaño más reducido, por ejemplo de un solo n-grama, terminaría indicando que todos los documentos son copias entre sí. En el segundo paso, se selecciona el subconjunto de valores *hash* que sean lo suficiente representativos de todo el documento [9], lo cual tampoco es una tarea trivial.

La técnica de huella digital es eficiente en el sentido que permite almacenar una pequeña porción del documento para el proceso de comparación, a la vez que, al no guardar el documento en sí, evita que los sistemas que lo implementen puedan ser empleados como fuente de plagio en sí mismas. Sin embargo, posee la desventaja de ser susceptible a pequeños cambios en la estructura de los códigos, por ejemplo remplazar un *for* por un *while* [2].

2.2. Herramientas para la detección de reutilización de código fuente

Existen algunas herramientas para identificar la detección de reutilización de código fuente [10], desarrolladas en otros países, pero muchas de ellas son poco amigables con el usuario, además de que no cuentan con su respectivo manual de usuario y su instalación suele ser compleja. Algunas de estas herramientas se revisan a continuación.

JPlag[4]. Es una herramienta no comercial, capaz de detectar reutilización multilingüe entre códigos fuente. Se empezó a desarrollar en el año 1996 mediante un

[4] http://plagiostop.wordpress.com/gratuito/jplag/

proyecto de investigación de la universidad alemana Karisruhe. En el año 2005 surgió como servicio web; para poder usar esta aplicación es preciso completar un registro y justificar que será utilizada por un profesor o investigador de alguna universidad o institución educativa.

Sherlock[5]**.** Es una herramienta de código abierto capaz de detectar reutilización entre documentos o códigos fuente en los lenguajes de programación Java y C. Fue desarrollada en la Universidad de Sidney; está implementada en el lenguaje de programación C; y utiliza firmas digitales para encontrar los fragmentos similares en el código fuente. Una firma digital es un número que está formado por varias palabras. Algunas desventajas es que no cuenta con interfaz gráfica; y al no utilizar el documento en su totalidad, no se puede afirmar que los documentos sean iguales.

Simian[6]. Es una herramienta de uso comercial, capaz de detectar reutilización multilingüe entre códigos fuente, tales como: Java, C, C++, COBOL, Ruby, JSP, ASP, HTML, XML, Visual Basic y texto natural. Fue desarrollado por una consultora de Australia llamada REDHILL. La herramienta esta implementada en el lenguaje de programación Java. No cuenta con interfaz gráfica.

Tester SIM[7]**.** Es una herramienta de código abierto, capaz de detectar reutilización multilingüe, algunos de los lenguajes que soporta: C, Java, Lisp, Modula2, Pascal y texto natural. La herramienta detecta fragmentos del código fuente potencialmente duplicados en grandes proyectos de software, no cuenta con interfaz gráfica y no muestra las partes reutilizadas del código fuente. Fue desarrollado por la Universidad de Ámsterdam.

Moss[8]**.** Es una herramienta no comercial, capaz de detectar reutilización entre documentos de texto y código fuente de varios lenguajes de programación, tales como: C, C++, Java, C#, Python, Visual Basic, JavaScript, Fortran, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2. La herramienta utiliza la técnica de huella digital para encontrar la reutilización de código fuente; fue desarrollada en 1994; se puede utilizar a través de internet una vez estando registrado. Una desventaja es que no muestra las partes reutilizadas del código fuente.

[5] http://sydney.edu.au/engineering/it/ scilect/sherlock/
[6] http://www.harukizaemon.com/simian/index.html
[7] http://www.cs.vu.nl/pub/dick/similarity_tester/README.1ST
[8] http://theory.stanford.edu/ aiken/moss/

3. Método de detección propuesto

En esta sección se comenta el método de detección de reutilización de código fuente propuesto, iniciando con las fases de preprocesamiento y representación de documentos de código fuente, concluyendo con la descripción de la medida de similitud.

3.1. Preprocesamiento de los documentos de código fuente

El preprocesamiento consiste en eliminar del código fuente espacios en blanco, saltos de línea, la capitalización de las letras, convirtiendo todos los caracteres en minúsculas. La cadena resultante se divide en n-gramas de caracteres, ya que al dividirlo se pierde la localización espacial, y no importa si un programador sitúa una función que estaba al principio del código fuente original, al final o en cualquier parte del mismo; así, el conjunto de n-gramas de un mismo código o de un fragmento de él permanece igual pese a estos cambios de posición. El uso de n-gramas de caracteres ha mostrado ser una forma de representación útil para identificar el estilo de cada autor [8]. En este sentido, nuestro trabajo se basó entonces en las ideas propuestas por [6,7] con la finalidad de tener una representación que sea capaz de identificar estilos de programación.

3.2. Representación de los documentos de código fuente

La representación más comúnmente utilizada para representar cada documento es un vector con términos ponderados como entradas, concepto tomado del modelo de espacio vectorial usado en Recuperación de Información [11]. Es decir, un texto d_j es representado como el vector $\overrightarrow{d}_j = \langle w_{kj}, \ldots, w_{|\tau|j} \rangle$, donde τ es el *diccionario*, *i.e.*, el conjunto de términos que ocurren al menos una vez en algún documento de Tr, mientras que w_{kj} representa la importancia del término t_k dentro del contenido del documento d_j. En ocasiones τ es el resultado de filtrar las palabras del vocabulario, *i.e.,* resultado de un preprocesamiento (sección 3.1). Una vez que hemos hecho los filtrados necesarios, el diccionario τ puede definirse de acuerdo a diferentes criterios. El criterio que se empleó en esta propuesta corresponde a Bolsa de Palabras (conocido como BOW del inglés Bag of Words), que es la forma tradicionalmente utilizada para representar documentos [12]. Este método de representación utiliza a las palabras simples como los elementos del vector de términos.

Con respecto al peso (*i.e.,* la importancia) w_{kj}, se tienen diferentes formas de calcularlo, entre las más usadas en la comunidad científica se tienen el ponderado booleano, ponderado por **frecuencia de término** y el ponderado por **frecuencia relativa de términos**. Una breve descripción se presenta a continuación:

- *Ponderado Booleano:* Consiste en asignar el peso de 1 si la palabra ocurre en el documento y 0 en otro caso.

$$w_{kj} = \begin{cases} 1, \text{ si } t_k \in d_j \\ 0, \text{ en otro caso} \end{cases} \tag{2}$$

- *Ponderado por frecuencia de termino (TF):* En este caso el valor asignado es el número de veces que el término t_k ocurre en el documento d_j.

$$w_{kj} = f_{kj} \tag{3}$$

- *Ponderado por frecuencia relativa (TF-IDF):* Este tipo de ponderado es una variación del tipo anterior y se calcula de la siguiente forma:

$$w_{kj} = TF(t_k) \times IDF(t_k) \tag{4}$$

donde $TF(t_k) = f_{kj}$, es decir, la frecuencia del termino t_k en el documento d_j. IDF es conocido como la "frecuencia inversa" del termino t_k dentro del documento d_j. El valor de IDF es una manera de medir la "rareza" del termino t_k. Para calcular el valor de IDF se utiliza la siguiente formula:

$$IDF(t_k) = \log \frac{|D|}{\{d_j \in D : t_k \in d_j\}} \tag{5}$$

donde D es la colección de documentos que está siendo representada en su forma vectorial.

3.3. Medida de similitud

Para el calculo de similitud entre dos documentos ($\overrightarrow{d_i}$ y $\overrightarrow{d_j}$) se han propuesto varias métricas que permiten determinar el parecido de éstos [13]. El objetivo de estas métricas es contar con un valor numérico al cual llamaremos *coeficiente de similitud SC*, el cual nos dirá cuán parecidos son los documentos $\overrightarrow{d_i}$ y $\overrightarrow{d_j}$. Una de las medidas ampliamente utilizadas en el campo de recuperación de información que permiten determinar la similitud entre documentos es la medida *cosenoidal*, la cual se describe a continuación.

Medida Cosenoidal. La idea básica de ésta es medir el ángulo entre el vector de $\overrightarrow{d_i}$ y de $\overrightarrow{d_j}$, para hacerlo, calculamos:

$$SC(\overrightarrow{d_i}, \overrightarrow{d_j}) = \frac{\sum_{k=1}^{t} w_{ik} w_{jk}}{\sqrt{\sum_{k=1}^{t} (w_{jk})^2 \sum_{k=1}^{t} (w_{ik})^2}} \tag{6}$$

Note que k va de 1 a el número total de términos del vocabulario τ, w_{ik} indica la importancia del término k en el documento $\overrightarrow{d_i}$ mientras que w_{jk} la importancia del término k en el documento $\overrightarrow{d_j}$.

4. Arquitectura de la herramienta

En esta sección se describe la arquitectura de la herramienta propuesta, la cual se muestra en la Figura 2. Durante el desarrollo de la herramienta, se utilizaron las siguientes tecnologías y lenguajes de programación: HTML, MySql, JSP, Java, CSS, Ajax y JavaScript. La herramienta desarrollada tiene un alcance de detección de reutilización de código hasta el nivel 3 (Sección 1) de complejidad. En las siguientes subsecciones de describen los tres módulos que componen la herramienta propuesta

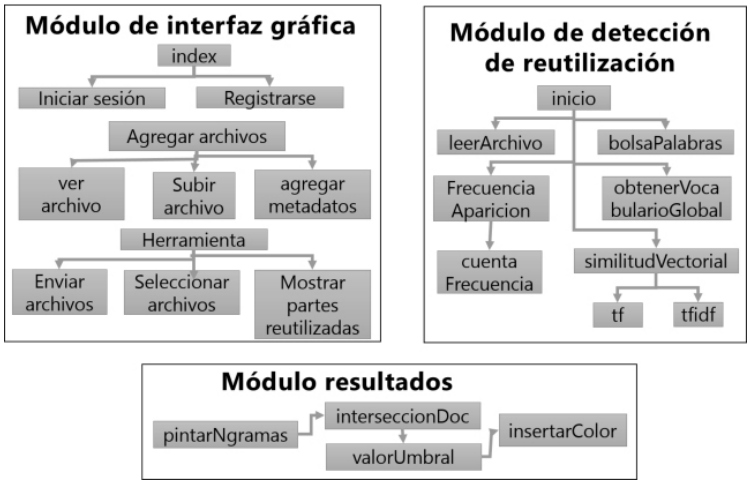

Fig. 2. Arquitectura del sistema propuesto. Note el diseño altamente modular, lo que permitirá en un futuro hacer modificaciones y/o extensiones de forma sencilla al sistema desarrollado hasta el momento.

4.1. Módulo de interfaz gráfica

Este módulo permite la interacción entre el usuario de la herramienta. Mediante éste, el usuario puede registrarse para realizar análisis de detección de reutilización de código fuente. También permite agregar los archivos que se quieran comparar y guardarlos en una base de datos para una futura utilización. La Figura 3 muestra un ejemplo de cómo seleccionar los archivos que se desean comparar.

4.2. Módulo de detección de reutilización de código fuente

Este módulo realiza la detección de reutilización de código fuente, tomando como entrada un archivo base y un archivo sospechoso que el usuario le proporciona al sistema mediante la interfaz gráfica (Sección 4.1). Una vez que el módulo recibe los archivos, realiza los pasos descritos en la Sección 3, y envía el resultado al Módulo de Interfaz Gráfica, para que sea mostrado al usuario, tal como se aprecia en la Figura 4.

4.3. Módulo de visualización de porciones reutilizadas

Este módulo se implementó para visualizar las partes reutilizadas entre un documento de código fuente base y un documento de código fuente sospechoso. Las partes del código fuente reutilizadas se verán resaltadas en tres colores diferentes que indican la frecuencia de los n-gramas encontrados, como se muestra en la Figura 5.

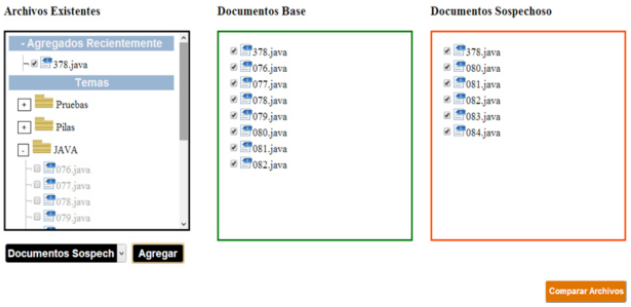

Fig. 3. Interfaz gráfica principal de la herramienta propuesta. En el recuadro más a la izquierda el usuario puede seleccionar de uno o varios directorios los archivos que desea analizar. Al momento de hacer esta selección el sistema pregunta al usuario cuáles son los archivos sospechosos de plagio y cuáles son los archivos originales, colocando los sospechosos en la ventana de la extrema derecha y los originales en la ventana de en medio.

Archivos Sospechoso	Archivos Base							
378.java	378.java 100.0%	076.java 36.0%	077.java 51.0%	078.java 29.0%	079.java 43.0%	080.java 49.0%	081.java 21.0%	082.java 21.0%
080.java	378.java 49.0%	076.java 44.0%	077.java 63.0%	078.java 42.0%	079.java 57.0%	080.java 100.0%	081.java 19.0%	082.java 38.0%
081.java	378.java 21.0%	076.java 12.0%	077.java 18.0%	078.java 13.0%	079.java 14.0%	080.java 19.0%	081.java 100.0%	082.java 14.0%
082.java	378.java 21.0%	076.java 25.0%	077.java 31.0%	078.java 26.0%	079.java 28.0%	080.java 38.0%	081.java 14.0%	082.java 100.0%
083.java	378.java 45.0%	076.java 44.0%	077.java 65.0%	078.java 39.0%	079.java 62.0%	080.java 61.0%	081.java 17.0%	082.java 45.0%
084.java	378.java 39.0%	076.java 29.0%	077.java 46.0%	078.java 30.0%	079.java 33.0%	080.java 51.0%	081.java 24.0%	082.java 27.0%

Fig. 4. Resultados del módulo de detección de reutilización de código fuente. Con la intensión de facilitar al usuario la tarea de identificar códigos reutilizados, los resultados se muestran en forma de una matriz de similitudes, siendo las filas los archivos sospechosos y las columnas los archivos marcados como originales en la interfaz principal (Figura 3). Note que en las celdas de la matriz aparece el nombre del archivo junto con su porcentaje de similitud, así entonces porcentajes altos significa que hay una alta similitud entre los archivos correspondientes.

Fig. 5. Código de colores empleado para resaltar las partes reutilizadas. El color más intenso (rojo) significa una coincidencia alta, mientras que el color menos intenso (amarillo) representa una reutilización baja.

Una vez que el usuario selecciona cualquiera de los archivos de la matriz de resultados (Figura 4), la herramienta le muestra el código base y el código sospechoso resaltados de acuerdo al código de colores explicado en la Figura 5. La Figura 6 ilustra un ejemplo del resaltado de secciones reutilizadas entre un par de códigos.

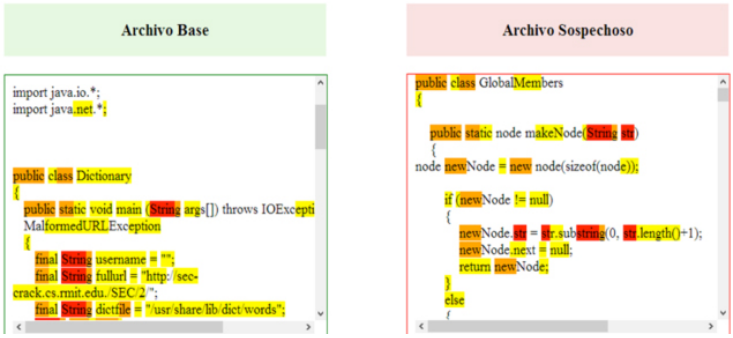

Fig. 6. Ejemplo de la visualización de secciones reutilizadas empleando el código de colores mostrado en la Figura 5. Note que las palabras reservadas tienden a ser las secciones del código que suelen tener altas coincidencias. Sin embargo, este tipo de visualización ayuda en gran medida al experto a identificar de manera rápida las posibles secciones reutilizadas, proporcionándole más elementos al momento de emitir un juicio de plagio o no-plagio entre uno más códigos fuente.

Es importante mencionar que para definir cuando un n-grama es de frecuencia alta, intermedia o baja no se utilizó un umbral fijo de frecuencia. En su lugar, el sistema propuesto calcula al vuelo cuando una frecuencia es alta, intermedia o baja tomando en cuenta el peso TF-IDF de los n-gramas que conforman el vocabulario del par de códigos que se están comparando en determinado momento.

5. Conclusiones y trabajo futuro

En este trabajo se presenta una herramienta para la detección de reutilización de código fuente, sin precedente en nuestro país. La herramienta aún necesita probarse con un mayor número de usuarios y códigos fuente.

Hasta el momento se ha observado que los comentarios, palabras reservadas, caracteres repetitivos (como puntos, comas, y los signos propios de los lenguajes de programación) interfieren en la detección al elevar el porcentaje de similitud con la medida adoptada, sin embargo, al hacer pruebas cualitativas contra una de las herramientas más populares (*i.e.,* JPLAG) notamos que casos de reutilización alta no son detectados por Jplag y si por el sistema propuesto. Con la intención de validar el sistema desarrollado se planea como parte del trabajo futuro conseguir un conjunto de colecciones estándar con las cuales sea posible determinar el grado de efectividad de la herramienta desarrollada.

Agregado a lo anterior, como parte del trabajo a futuro también se prevee extender el enfoque que realice un análisis de código con base en n-gramas de palabras, además de probar otras técnicas de comparación de código y emplear diversas medidas de similitud. Finalmente, es conveniente mencionar que la herramienta desarrollada en este trabajo estará disponible en el sitio del Grupo de Lenguaje y Razonamiento de la UAM-C cuya página es: http://lyr.cua.uam.mx

Agradecimientos. Agradecemos el apoyo otorgado por la Universidad Autónoma Metropolitana Unidad Cuajimalpa y el SNI-CONACyT.

Referencias

1. Sánchez Vega, J.F.: Detección automática de plagio basada en la distinción de fragmentación del texto reutilizado. Tesis de Maestría. Instituto Nacional de Astrofísica Óptica y Electrónica (2011)
2. Faidhi,J.A.W., Robinson, S.: An empirical approach for detecting program similarity and plagiarism within a university programming environment. Computers and Education, 11(1), pp.11–19 (1987)
3. Sánchez-Vega, J.F., Villatoro-Tello, E., Montes-y-Gómez, M., Villaseñor-Pineda, L., Rosso, P.: Determining and characterizing the reused Text for Plagiarism Detection. Expert Systems with Applications, 40(5), pp. 1804–1813 (2013)
4. Franco Salvador, M.: Detección de plagio translingüe utilizando una red semántica multilingüe. Tesis de Maestría, México D.F., México (2013)
5. Barrón-Cedeño, A.: Detección automática de Plagio en Texto. Tesis de Doctorado. Valencia, España (2012)
6. Flores, E., Barrón-Cedeño, A., Rosso, P., Moreno, L.: Towards the detection of cross-language source code reuse. In: Proceedings of the 16th International conference on Applications of Natural Language to Information Systems (NLDB), pp. 250–253 (2011)
7. Flores, E., Barrón-Cedeño, A., Rosso, P., Moreno, L.: DeSoCoRe: Detecting source code re-use across programming languages. In: Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstration Session, NAACL, pp. 1–4 (2012)

8. Frantzeskou, G., Stamatatos, E., Gritzails, S., Katsikas, S.K.: Source Code Author Identification Based On N-gram Author Profiles. pp. 508–515 (2006)
9. Alva Manchego, F.E.: Sistema de información de detección de plagio en documentos digitales usando el método document fingerprinting. Tesis de Maestría. Peru (2010)
10. Herramienta. Sitio web describiendo varias herramientas para la detección de plagio en Código Fuente. (Noviembre 2013). http://www.linti.unlp.edu.ar/uploads/docs/herramientas_para_la_deteccion_de_plagio_de_software_un_caso_de_estudio_en_trabajos_de_catedra. %20Un %20caso %20de %20estudio %20Anhi.pdf
11. Baeza-Yates, R., y Ribeiro-Neto, B. Modern Information Retrival. Addison Wesley (1999)
12. Sebastiani, F.: Machine Learning in Automated Text Categorization. In: ACM Computing Surveys, 34(1), pp. 1–47 (2002)
13. Grossman, D.A., Frieder, O.: Information Retrieval, Algorithms and Heuristics. Springer (2004)