Máster en "Software Libre"
2015-2016

Estudio e Implementación de una herramienta para la detección de plagio entre circuitos digitales

Artur Dinaret Dorca
Universitat Oberta de Catalunya
Correo-e: adinaret@gmail.com

Abstract. *En el entorno educativo, a menudo es necesario la utilización de herramientas para automatizar el proceso de detección de plagio entre los ejercicios presentados por los alumnos para descartar posibles copias totales o parciales de los trabajos. La asignatura de Fundamentos de Computadores del Grado de Ingeniería Informática y del Grado en Tecnologías de Telecomunicación de la UOC se encuentra con el reto de automatizar el proceso de detección de plagio en la creación de los diversos diseños de circuitos digitales que son realizados por parte del alumnado.*

En este contexto es dónde nace la herramienta para la detección de plagio que se presenta a continuación, implementando un algoritmo para realizar esta detección a nivel de similitud entre los diseños de los circuitos y de los elementos que lo forman, se intenta automatizar este proceso para la detección de copias de tal manera que el profesorado disponga de una herramienta fiable con la que realizar este proceso de detección de plagio.

1 Introducción

En el entorno educativo, a menudo es necesario la utilización de herramientas para automatizar el proceso de detección de plagio entre los ejercicios presentados por los alumnos para descartar posibles copias totales o parciales de los trabajos y ejercicios presentados.

Partiendo de este contexto, la investigación está claramente orientada al mundo educativo. Busca facilitar la evaluación de los ejercicios realizados por los alumnos comprobando el plagio entre ellos de una manera automatizada. El proceso se realiza mediante la búsqueda de la similitud entre los ficheros de las dos propuestas y la detección de plagio entre ejercicios mediante otras técnicas que permitan analizar el contenido de cada una de las propuestas y aplicar los diferentes algoritmos para detectar intentos de copias más sutiles, como la modificación de elementos concretos en algunos de sus atributos. Concretamente, este trabajo de investigación tiene una aplicación directa en el ámbito de la asignatura de Fundamentos de Computadores de la "Universitat Oberta de Catalunya". Pero si evaluamos la propuesta de una manera más global, puede ser claramente utilizada en cualquier otra universidad o centro educativo, instituto o centro de formación profesional, que tenga la necesidad de detectar el plagio en los ejercicio de diseño de circuitos entregados por sus alumnos utilizando métodos mas avanzados que la simple comparación de archivos. Estos métodos tienen como objetivo profundizar más en la comprensión de los elementos del circuito y detectar similitudes en los elementos que conforman el circuito, aunque tengan atributos que puedan ser distintos fruto de la manipulación intencionada por parte del alumno para disimular el plagio.

La aplicación propuesta ha sido escrita íntegramente en Java, lo cual nos facilita, entre otras muchas cosas, la representación de entidades propias de las funcionalidades que desarrollamos y proporciona librerías ya implementadas por terceros para realizar muchas acciones que no aportan valor a la investigación, como por ejemplo, el análisis de documentos XML. Estos son precisamente el tipo de documentos que la aplicación espera recibir como formato de entrada y que representan los diseños de circuitos generados con la herramienta VeriUOC (Logisim). El archivo de salida con los resultados de la comparación se muestra en formato HTML facilitando de este modo la visualización y navegación entre los diferentes resultados de la aplicación del algoritmo.

En este artículo se presenta, estructurado por secciones, el trabajo realizado para el desarrollo de la herramienta. En la sección 2 se puede ver el estudio previo realizado en trabajos similares para la detección de plagio en los campos de la similitud entre textos, circuitos o código en ámbitos de programación. A continuación, en la sección 3, se describe cómo se realiza la detección de copias entre diseños de circuitos, el método utilizado. Las fases de las que se compone el algoritmo implementado en la herramienta y el diseño que se ha seguido para conseguir el objetivo se describe en la sección 4. Los resultados obtenidos durante los experimentos y las pruebas realizadas se presentan en la sección 5 del presente artículo. Por último, en la sección 6, se relatan las conclusiones a las que se ha llegado durante la realización del trabajo

2 Trabajos Relacionados

Dentro del ámbito educativo son diversas las opciones en forma de software que los profesores tienen a su alcance para automatizar los procesos de detección de plagio entre trabajos y ejercicios entregados por sus alumnos.

En el caso de ejercicios basados en texto, las opciones son múltiples si hablamos de herramientas que se centran en la detección y la comparación de documentos de texto siguiendo diferentes técnicas, como por ejemplo, la que nos encontramos en el artículo [1], donde se describe la evolución del clásico algoritmo de recuento de atributos de los elementos que forman el texto hacia un algoritmo basado en la comparación de estructuras y en la similitud de las mismas. Fusionando los dos conceptos se crea un herramienta mucho más precisa a la hora de afrontar las comparaciones puesto que se basa en dos técnicas que aportan fiabilidad, hasta el punto que algunas herramientas que implementan este tipo de algorítmica pueden realizar comparaciones, no sólo entre documentos entregados por los alumnos, si no que van más allá y realiza comparaciones con documentos que se pueden encontrar en Internet, en diferentes formatos (Url, .docx, odt, pdf...etc) y lo hacen de manera rápida y eficiente. El diseño de los circuitos generados por VeriUOC (Logisim) tiene un formato XML, lo que permite que se traten cómo un documento de texto tal y como se describe en el artículo.

En el caso de comparación de documentos encontramos bastantes ejemplos de casos prácticos que funcionan muy bien, pero si centramos la atención en herramientas para detectar el plagio en el caso de ejercicios que pertenecen a asignaturas típicas de carreras científico-técnicas vemos que el abanico de posibilidades es un poco menor, siendo la detección de plagio en software la más común de todas y aconteciendo una fuente de inspiración importante para la implementación de la presente herramienta, puesto que los patrones y técnicas para la comparación son fácilmente exportables a la detección del plagio en circuitos lógicos.

Una aproximación a la detección de plagio en código adecuada al presente objetivo es la metodología basada en la comparación de estructuras (*structure-based method*), la cual se basa en un algoritmo de división de cadenas de caracteres y su posterior comparación para medir las posibles similitudes. Algunos de los sistemas de detección de plagio actuales utilizan este método, como por ejemplo el software Jplag.

Tal como podemos extraer del artículo [2], Jplag es un sistema utilizado para la detección de código escrito en C, C++ y Java que se ofrece vía web de manera gratuita. Dado un conjunto de programas objetivos de nuestro análisis, Jplag analiza su contenido y lo divide en muestras. Estas muestras serán comparadas entre sí siguiendo el algoritmo RKR-GST [2] (*Running Karp-Rabin Greedy String Tiling algorithm*). Una vez realizado, los resultados serán mostrados al usuario vía web donde verá estadísticas de detección, distribuciones similares y parejas de programas sospechosas de ser plagio, que podrán ser seleccionadas por el usuario y estudiadas en detalle, de una manera sencilla, para ver los fragmentos plagiados.

En esta misma línea, el artículo [3] nos presenta un escenario similar. Partimos de un escenario previo centrado en el entorno académico, más concretamente en todas las asignaturas que impliquen desarrollo de software de algún modo u otro y la dificultad para detectar el plagio de cualquier otra manera que no sea un proceso automatizado.

Aquí se nos presenta un prototipo de herramienta, llamada Deimos, que tiene como objetivo la detección de plagio en diversos fragmentos de código escritos con diferentes paradigmas, por ejemplo, con Pascal o LISP. Lo hace siguiendo un algoritmo muy similar al caso anterior, RKR-GST, que podemos dividir en dos partes diferenciadas. Por un lado, el programa analiza el código objeto y lo divide en partes relevantes, una vez hecho esto intenta buscar patrones similares y aplicar una lógica en la que, además similitud y tamaño de los fragmentos, más posibilidad de plagio y por lo tanto se puede categorizar el porcentaje de plagio por ejercicio.

En esta misma línea, otro ejemplo de herramienta centrada en el plagio de ejercicios basados en desarrollo de código, es la que aparece en el artículo [4], que se acerca a la resolución de este problema mediante una herramienta llamada PK2, que ha sido utilizada en el Departamento de Arquitectura Computacional de la Universidad Politécnica de Madrid durante diez años para intentar detectar el plagio entre un gran número de muestras. La filosofía en la que se basa el algoritmo para determinar el plagio es que la comparación no puede ser reducida a

una sola medida para determinar la similitud, dado que un programa no es un objeto que se pueda reducir a una sola dimensión. Por ello, el mecanismo de detección se basa en cuatro criterios diferentes con sus correspondientes medidas.

Los criterios para escoger estas cuatro medidas pasaron porque debían ser criterios heurísticamente sencillos de entender, eficientes a nivel computacional e independiente entre sí.

- El primer criterio se basa en la longitud de la secuencia más larga de palabras reservadas entre dos archivos, se trata de una medida absoluta dado que no está normalizada. Obviamente cuanto más grande sea la cadena más grande será la posibilidad de plagio, pero una de las primeras cosas que se hacen al plagiar documentos es mover los bloques y por tanto el *hash* de las palabras claves no coincidirá en muchos de los casos, es por eso que se necesitan más criterios.

- El segundo criterio pasa por la medida del valor acumulativo de las palabras claves, aquí no se tienen en cuentas conceptos como la posición que si teníamos en el primer criterio y se trata también en una medida absoluta.

- El tercer criterio es la normalización del segundo valor. Aprovechando que el código plagiado, muchas veces, intenta introducir fragmentos de código que no son necesarios por lo tanto de ocultar el material copiado, este criterio intenta identificar palabras o cadenas de palabras dentro de otros. Dado que esta medida depende de la longitud relativa de las cadenas de caracteres su valor se normalizará en base al 100%

- El cuarto criterio se basa en la localización de palabras y el cálculo del número existentes con el objetivo de no depender de la posición. Es un criterio muy similar a *attribute counting*. Este criterio es el más flojo de todos ya que puede dar valores muy altos en ejercicios diferentes y originales, no obstante es la métrica más difícil de engañar para los tramposos, ya que requiere un número elevado de cambios para reducir el valor de este atributo.

Los ámbitos de aplicación de PK2 son sistemas programados en C, en Ensamblador, microprogramación, sistemas de entrada / salida. Algunas de las deficiencias que presenta PK2 es un programa compuesto por varios fragmentos de código muy pequeño, dado que en esta casuística la información de cada fragmento es demasiado limitada, el programa PK2 no puede distinguir el plagio dado que todas las soluciones de los estudiantes son demasiado similares entre ellas.

Otra aproximación interesante es la que encontramos en el artículo [5]. Aquí tenemos un trabajo que compara y analiza dos formas diferentes de detectar el plagio, *attribute counting* (visto en el artículo anterior) y system-counting metric, con una implementación de referencia en este sistema como es YAP. Por un lado, en el caso de algoritmos basados en attribute counting, el autor plantea dos sistemas que se basan en este principio y que son afinados de tal manera que sean capaces de detectar el plagio en programas escritos en Pascal:

Grier's Accuse system [4]: Verifica los parámetros en diferentes combinaciones para determinar la combinación más efectiva, el sistema resultante utiliza siete parámetros
1. Número operadores únicos
2. Número operandos únicos
3. Operadores Totales
4. Operandos Totales
5. Líneas de Código
6. Variables declaradas y utilizadas
7. Declaraciones de control Totales

Faidhi-Robinson System [4]: sistema más sensibles que versiones anteriores ya que proporciona menos falsos positivos argumentando que las métricas que implementa tienen un grado mayor de desacoplamiento entre ellas y son métricas que deberían ser difícil de engañar por un usuario novato, en este caso las diez métricas utilizadas son
1. Media del número de caracteres por línea
2. Número de líneas comentadas
3. Número de líneas con sangría
4. Número de líneas en blanco
5. Media de las longitudes de las funciones
6. Número de palabras reservadas
7. Media de la longitud de los identificadores
8. Media de porcentaje de espacio por línea
9. Número de etiquetas y *GoTo* (Ensamblador, Pascal...)
10. Número de identificadores únicos

Y a continuación, catorce criterios más que intentan medir características menos obvias e intrínsecas de la estructura del programa

11. Número de intervalos del programa
12. Número de vértices con colores básicos (1 y 2)
13. Número de vértices con colores 3
14. Número de vértices con colores 4
15. Porcentaje de expresiones en la estructura del programa
16. Porcentaje de expresiones simples en la estructura del programa
17. Porcentaje *Program Term Structure*
18. Porcentaje *Program Factor Structure*
19. Número de impurezas en el programa
20. Número de líneas dentro de funciones o procedures

21. Número de módulos
22. Porcentaje de expresiones condicionales
23. Porcentaje de expresiones repetitivas (bucles)
24. Número de declaraciones

Una vez definidas las bases de los algoritmos *attribute counting*, realiza la comparación entre las dos añadiendo YAP3, sistema basado en *system-counting* sobre una muestra de diferentes programas escritos en Pascal, realizado por alumnos.

Una vez obtenidos los datos y estudiadas podemos concluir que un algoritmo *attribute counting* es útil y tiene mejor rendimiento en los casos de detección de plagio de copias muy similares entre sí, pero tiene problemas a la hora de detectar plagios parciales, entonces parece más adecuada su utilización en el caso de un grupo de estudiantes más inexpertos. *Attribute counting* también tiene problemas para detectar la reescritura del programa siguiendo la misma estructura que su copia, por tanto el artículo desaconseja el uso de Grier s Accuse system y restringe el uso de Faidhi-Robinson System en casos muy concretos.

Por lo que podemos deducir herramientas como YAP3, basados en sistemas structure-metrics mejoran en muchos aspectos las basadas en attribute counting, si nos centramos en el artículo [6], vemos más en profundidad la herramienta y destacamos que está basada en el algoritmo RKS-GST que actúa en dos fases. Simplificando podemos describir que en una primera fase el código fuente se utiliza para generar secuencias de referencia, esta fase se denomina scan patter (búsqueda de patrones), en la segunda fase, llamada markstrings intenta encontrar las concordancias más grandes posibles en la estructura, para de esta manera determinar el grado de plagio.

En esta línea de mejora, el artículo [7] plantea una mejora destacando que el sistema system-counting metric presenta debilidad en la segunda parte de su fase, concretamente en cómo medir la similitud en una pareja de secuencias y se corre el riesgo de perder información si no se utiliza una métrica apropiada. El artículo realiza una nueva aproximación a la detección de la similitud de dos secuencias basada en la Complejidad de Kolmogorov y nos la presenta en forma de herramienta que denomina SID (Software Integrity Diagnosis system), escrita en Java, y que implementa el algoritmo prpuesto en dos fases la primera llamada “Token Compress” y la segunda que llama “System Integration”.

Un análisis general de los algoritmos existentes utilizados en este ámbito lo podemos encontrar en el artículo [8], presentado a modo de resumen, destaca las ideas más relevantes comentadas hasta el momento intentando describir aquellas características comunes entre ellos y representa de una manera muy visual, la lógica de este algoritmos en forma de diagrama de flujo, información muy útil para sintetizar todas las ideas descritas hasta el momento.

Los artículos vistos hasta ahora centran su campo de aplicación en la detección de plagio en software, estos casos son extensibles al plagio en circuitos digitales, no obstante, existen una serie de artículos que pueden ejemplificar el trabajo realizado hasta el momento en el campo de la detección de plagio entre circuitos lógicos. Una muestra de ello la encontramos en el artículo [9] donde vemos varios criterios utilizados para detectar coincidencias de los diferentes elementos que conforman los circuitos en sus disposiciones y el lugar que ocupan en el diseño. Los criterios son:

TDLC - *Transistor distance to the layout center*
TTD - *Transistor-to-transistor distance*
Type-aware transistor-to-transistor distance

Todas estas técnicas suponen una gran ventaja en el sentido de que al ser medidas invariantes podemos extraer las diferentes firmas que se vayan produciendo de tal modo que el circuito original no debe ser analizado de nuevo, conservando su información en una base de datos para el posterior análisis de siguientes iteraciones.

Finalmente el artículo presenta un desarrollo en Python basado en los tres criterios que nos presentan y vemos los resultados que se producen a la hora de detectar el plagio en un conjunto de sesenta y dos diseños de circuitos hechos por estudiantes. El resultado es una herramienta capaz de identificar las similitudes existentes, y por tanto, poder detectar el plagio. El rendimiento de la aplicación es suficiente para la tarea que se quiere llevar a cabo, pero existe un problema con la aplicación de falsos positivos, lo que requiere una inspección visual más exhaustiva para determinar la aparición o no de copia.

Fuera del ámbito académico, encontramos dos artículos interesantes relacionados con el tema tratado.

Por un lado [10] nos propone una técnica interesante en forma de patrón de búsqueda para analizar los diferentes elementos que forman el circuito, de tal manera que el algoritmo es capaz de detectar patrones en grandes circuitos con un rendimiento aceptable. Al ser una técnica aplicable independiente de la tecnología de los circuitos y del estilo de diseño parece claramente adaptable a un algoritmo de búsqueda de patrones para comparar bloques que puedan ser plagios dentro de un entorno académico.

Por otra parte, también se puede destacar [11] que propone varias ideas que pueden resultar interesantes. Son tres los procesos que se realizan: reducción de redes, comparación basada en isomorfismo de grafos y una regla basada en la búsqueda de inconsistencias.

3 Método Propuesto

De todos los algoritmos antes analizados, la solución propuesta se basa en *attribute counting* porque es la que ha dado mejores resultados en su aplicación en distintos ámbitos similares al problema que nos ocupa. Al igual que en el artículo [4], el existo de la aplicación de la técnica, reside en la selección correcta de diversas métricas relevantes en el caso de los diseños de circuitos lógicos.

La metodología utilizada para la detección del plagio en el diseño de los ejercicios realizados por los alumnos se basa en dos fases diferenciadas.

En la primera fase se realiza una comparación directa entre los elementos que forman el circuito, buscando aquellos que son exactamente iguales entre sí. Esta primera funcionalidad podría considerarse como una detección de documentos XML idénticos pero introduce una diferencia importante. Los elementos analizados son aquellos que se consideran relevantes y propios del diseño del circuito. De esta manera se tienen en cuenta elementos como cables o componentes, por ejemplo puertas lógicas y se descartan todos aquellos elementos que no aportan nada en el diseño y que aparecen en los documentos XML resultado del diseño con la herramienta VeriUOC, como importación de librerías o configuración propia del programa de diseño.

Esta fase inicial tiene como objetivo detectar aquellos ejercicios que han sido plagiados de forma literal, sin introducir variación alguna, por lo que podemos asumir que lo que estamos realizando es una comparación entre los dos documentos, aunque solo nos fijemos en elementos del propio diseño. Todos aquellos ejercicios que coincidan en todos los elementos analizados y comparados serán un plagio idéntico y, por lo tanto, serán considerados como copia. En este caso no se aplicará la siguiente fase del algoritmo ya que carece de sentido al tratarse de un claro caso de plagio.

La segunda fase del algoritmo, tiene como objetivo realizar un análisis para la detección de casos de plagio más sutil donde se realizan modificaciones de los elementos con el objetivo de alterar lo suficiente el diseño para que no se aprecie una copia directa. Este análisis se basa en realizar una comparación, siguiendo distintos criterios, de los elementos relevantes que forman el circuito de tal manera que se analicen, por ejemplo, la posición que ocupan dentro del diseño, la longitud de los cables o el número de elementos comunes que comparten los dos diseños.

Los criterios han sido elegidos de tal manera que representen situaciones relevantes en el diseño de los circuitos y puedan ser valores representativos del diseño propio de este tipo de ejercicios.

A continuación se detallan los criterios aplicados en este apartado:

- Componentes utilizados: Cuenta el número de componentes de un tipo concreto utilizados en el diseño y lo compara con el número de componentes utilizado en el otro circuito. De tal manera que, por ejemplo en un circuito se pueden haber utilizado 5 puertas lógicas AND y en el otro el mismo número. Esto no siempre será indicativo de un plagio en los diseños, ya que un ejercicio en concreto puede ser que solo se resuelva utilizando un número determinado de elementos. Por lo tanto, se trata, quizás, de la métrica menos indicativa de todas, pero que puede ser concluyente si todas las demás indican plagio

- Posición idéntica de componentes: Se analiza el tipo de componente que estamos tratando y se extraen los atributos del mismo que indican la posición en la que está situado, se busca de manera iterativa el mismo elemento en el otro objeto de nuestra comparación en busca de una posición coincidente y realizar un recuento de esta coincidencia en este tipo de atributo.

- Posición de cables: Esta métrica tiene como objetivo centrarse en los cables que forman el diseño y busca posiciones de inicio o final coincidentes. Tal como sucedía en el caso anterior, se realiza un recuento para todos aquellos elementos que cumplan los criterios anteriores.

- Dimensión de cables: De la misma manera que el caso anterior, se centra en los cables del diseño y busca dimensiones iguales de los cables que forman el circuito para aquellos en los que coincidan en posiciones iniciales o finales. En caso de encontrar una coincidencia realiza un recuento de los elementos que cumplen el criterio

- Bloques conectados del mismo modo: Se identifican aquellos conjuntos de cables y componentes conectados de la misma manera con posiciones similares con el objetivo de detectar estructuras parecidas en el diseño del circuito. Se entiende por posiciones similares todas aquellas en las que los valores son exactamente iguales o se mueven en un margen de +- 5 puntos.

Una vez calculados todos los elementos anteriores, consideraremos que todos aquellos que tengan valores elevados en una o diversas métricas tendrán más probabilidad de tratarse de una copia.

Finalmente, se ofrece una métrica resultante que pretende interpretar cada uno de los resultados de las fases anteriores en su conjunto y ofrecer un porcentaje de probabilidad que determina en qué

grado existe similitud entre los dos diseños, y por tanto plagio. De esta manera, una comparación donde uno o varios de los criterios anteriores tengan unos valores elevados, aumentarán las posibilidades de ser un plagio. Cuanto más alto sea su valor y más métricas contengan un valor diferente de cero, más probabilidad existirá de que se trate de una modificación de ciertos elementos en un diseño original. Hay que tener en cuenta que el porcentaje de plagio se calculo en base al número de elementos totales presentes en el diseño, es decir, un circuito que tenga un elemento coincidente de diez existentes tendrá un porcentaje de plagio mayor que un diseño que contenga un elemento coincidente sobre cien

Evidentemente, no todas las métricas antes propuestas tienen el mismo peso dentro de la determinación de plagio. De esta manera se tiene en cuenta que en algunos diseños el número de componentes utilizados puede ser coincidente y puede no tratarse de un caso de plagio. Así mismo otros valores como por ejemplo, posiciones coincidentes y longitudes coincidentes en los cables del diseño pueden determinar en un grado más elevado una posibilidad de plagio. De esta manera, realizando una media ponderada según el peso de cada uno de los valores, se calcula la posibilidad de copia en el sumario resultante

Los cálculos, según los distintos criterios, son los siguientes, estos valores no pueden ser modificados por el usuario ya que componen parte del algoritmo original y por lo tanto están integrados dentro del código

Si no se detecta ningún elemento exactamente igual en los dos circuitos.

- Componentes utilizados – 5%
- Componentes posición idéntica – 20%
- Cables posición idéntica (inicio o final) – 20%
- Cables longitud idéntica (con posición inicio o final coincidente) - 30 %
- Conexiones realizadas del mismo modo – 25%

Si se detecta algún elemento exactamente igual en los dos circuitos.

- Componentes exactamente igual o cables exactamente igual - 50%
- Componentes utilizados – 2,5%
- Componentes posición idéntica – 10%
- Cables posición idéntica (inicio o final) – 10%
- Cables longitud idéntica (con posición inicio o final coincidente) - 15 %
- Conexiones realizadas del mismo modo – 12,5%

Si se detecta que todos los elemento son exactamente igual en los dos circuitos.

- Se anula el cálculo del resto de métricas y se proporciona un valor de 100% en la posibilidad de plagio

Cada uno de los valores de estos porcentajes han sido elegidos después de diversas pruebas con el objetivo de representar el peso relevante de cada una de las métricas que representan el algoritmo.

4 Diseño e Implementación

La herramienta se presenta en forma de aplicación que implementa toda la metodología explicada en el punto anterior. Este hecho busca como objetivo ejemplificar y poner en práctica la metodología propuesta y proporciona una solución funcional para la detección del plagio en el entorno previamente descrito.

La aplicación está escrita en Java y sigue todos los parámetros canónicos de la programación orientada a objetos. Crea entidades que representan objetos reales dentro de nuestro diseño del circuito y extiende funcionalidades utilizando herencias múltiples, de tal manera, que la implementación siga el paradigma de la POO y proporcione un código que pueda ser reutilizado o ampliado en futuras revisiones. Cada clase pretende proporcionar un elevado grado de abstracción de cara a ser utilizada desde otra parte del código de tal manera que encapsule les funciones necesarias en cada caso.

El diseño UML de clases que forman la aplicación es el mostrado en la Figura1. Se ha omitido todas aquellas clases relativas a la visualización de la aplicación y a la formación de los HTML resultantes, ya que se considera que no aportan valor en este diseño al no aportar nada nuevo a la investigación.

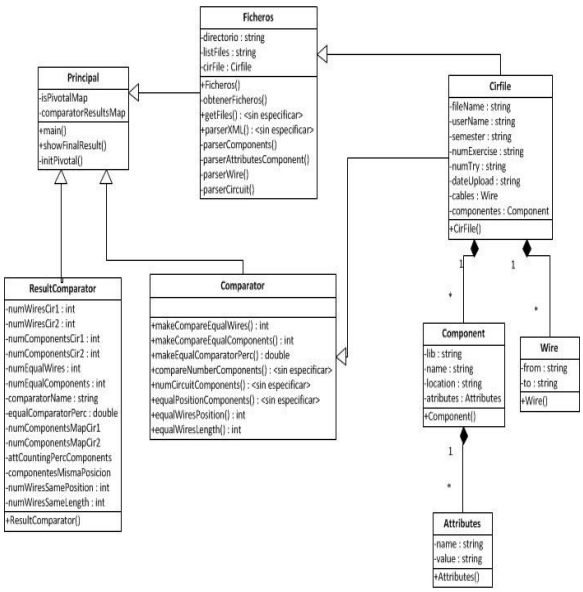

Figura 1. Diseño UML de la aplicación

Como entrada, al programa se le facilita la carpeta donde se encuentran los ejercicios, en formato XML, de los diseños hechos con VeriUOC. El algoritmo es capaz, de manera automatizada, de realizar comparaciones entre aquellos diseños que tengan un número de ejercicio coincidente. Así mismo, busca realizar comparaciones entre el número de ejercicio más alto ya que se considera el último y definitivo intento por parte del alumno y jamás tendrá en cuenta comparaciones entre ejercicios del mismo alumno.

Una vez determinados los objetos de la comparación, se analiza todo el fichero XML y se extraen aquellos datos necesarios para montar estructuras que representen los elementos, instanciando clases que serán los componentes de la entidad circuito.

A partir de aquí, se empieza a realiza de manera iterativa una comparación entre los diseños de la carpeta objeto, donde nunca se repetirá una comparación ya realizada. El resultado se irá almacenando en un objeto llamado ResultComparator que encapsulará todos los resultados de la comparación para los distintos criterios definidos en el apartado metodología. Toda la lógica de la metodología de comparación esta implementada en la clase Comparator donde se ofrece de manera pública todos aquellos métodos que implementan, de manera aislada, cada uno de los criterios que compone la metodología.

Al terminar este proceso de comparación, se recorre la colección que alberga todos los objetos ResultComparator para extraer los datos y plasmarlos sobre diversos documentos de salida en formato HTML lo que facilitará la lectura por parte del usuario y permite obtener detalles de cada una de las comparaciones realizadas navegando entre los distintos elementos interactivos. Estos archivos de salida se almacenarán de manera local en el ordenador donde se haya ejecutado el programa.

El diagrama de flujo general que sigue la aplicación se muestra en la Figura 2.

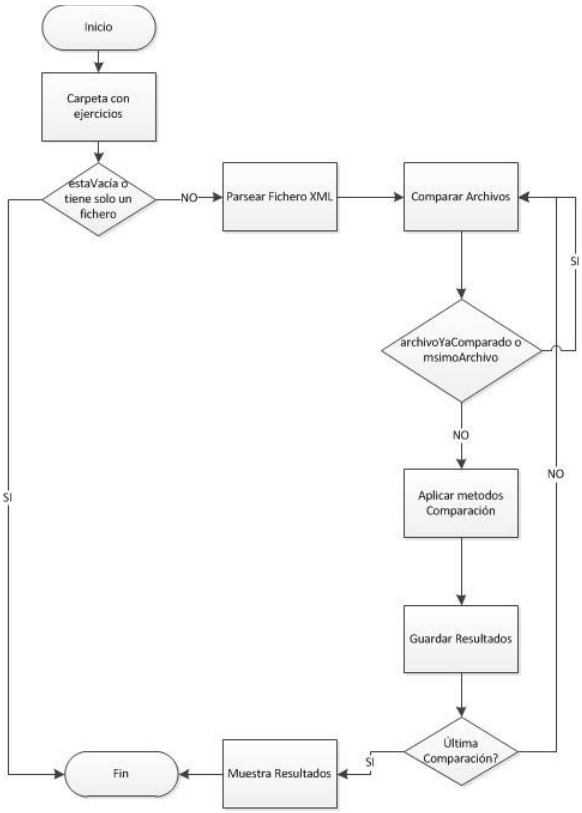

Figura 2. Diagrama de flujo

Los resultados son mostrados al usuario en formato HTML, cosa que facilita su lectura. La salida la componen un HTML general donde se puede ver el resumen del resultado de la comparación entre todos los archivos y un conjunto de ficheros, también en formato HTML, que contienen los detalles de cada una de las comparaciones. Estos están linkados de manera automática al HTML sumario.

5 Resultados Experimentales

Para realizar las pruebas de la aplicación, se ha tenido a disposición ejercicios reales realizados por alumnos en diferentes cursos donde se ha utilizado la herramienta VeriUOC para el diseño de los mismos. Por lo tanto todas las pruebas y los resultados obtenidos han sido en un entorno real donde la aplicación del método propuesto tiene como objetivo ser utilizada.

Una vez analizados y comparados todos los ejercicios entre si, se ha determinado que un % superior a 25 en el grado de similitud establece una alerta amarilla siendo necesaria una revisión por parte del profesorado, ya que muchas de las coincidencias encontradas pueden indicar que se está produciendo una copia. Para una coincidencia superior o igual al 50% se establece una alerta roja, indica que la comparación tiene un alto grado de igualdad entre los

dos diseños y con mucha probabilidad puede tratarse de un plagio. Se ha reservado el valor de 100% de igualdad para aquellos casos donde todos los elementos del circuito son exactamente los mismos, indicando por lo tanto, que se trata del mismo diseño y la copia es evidente. Estos valores pueden ser personalizados por el profesorado que utiliza la herramienta según su propio criterio y dependiendo del tipo de diseño que se este analizando, Figura 3

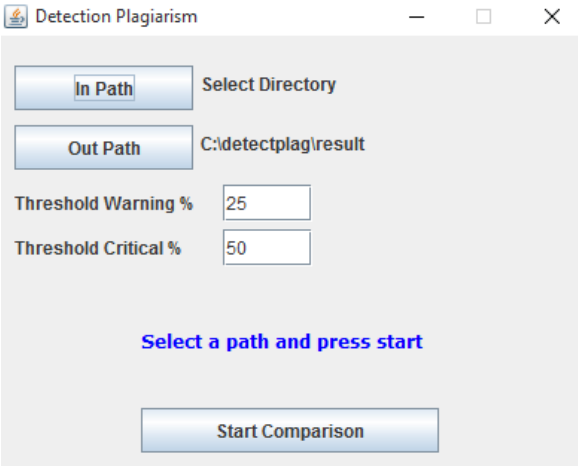

Figura 3. Interfície visual de la herramienta

Los resultados obtenidos han sido satisfactorios, siendo una de las prioridades durante la implementación la reducción al mínimo posible de falsos positivos, como objetivo principal para que la herramienta puedo ser lo mas fiable posible.

A continuación se muestran algunos resultados de las pruebas funcionales realizadas sobre ejercicios reales.

El volumen analizado de ejercicios soportados por la herramienta es elevado, a continuación se muestran resultados correspondientes a 10111 comparaciones que se han realizado en menos de 10 segundos, el tiempo en realizar las comparaciones es variable según la potencia de la máquina donde se ejecute la aplicación, en las pruebas realizadas las especificaciones han sido las siguientes :

- Procesador: Intel core i5-3230M 2.60 Ghz
- Memoria : 8Gb
- Disco : SSD 240 GB
- Sistema Operativo Windows 10, 64bits
- Otros : Java JRE 1.7

Estos resultados se han realizado sobre el ejercicio número 111 correspondiente a diversos ejercicios realizados entre el año 2011 y 2012 en sus dos semestres. Los resultados son los siguientes.

En primer lugar se muestra una tabla resumen con todos los resultados de las comparaciones, Figura 4

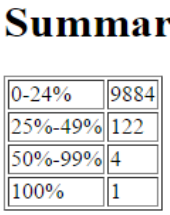

Figura 4. Resumen resultados comparación ejercicio 111

Se puede ver que la mayoría de comparaciones analizadas, 9884, no representan un valores a destacar (0%-24%). 122 comparaciones necesitan ser revisadas ya que algunos elementos podrían indicar que se esta produciendo copia parcial en algunos de sus elementos (25%-49%), pero en general, la mayoría de del diseño no comparte elementos comunes. En cambio,4 de las comparaciones realizadas superan el porcentaje de similitud del 50%, cosa que indica que muchos de sus elementos son coincidentes en alguna de las métricas descritas anteriormente y para finalizar, una de las comparaciones realizadas da un resultado de similitud del 100%, valor que indica un plagio total. Todos los resultados se han comprobado manualmente para verificar que efectivamente, el grado de similitud es elevado. Por motivos de privacidad se han eliminado el nombre de los alumnos de las capturas de pantalla que se indica en la columna de la izquierda.

En este mismo Html se puede ver la tabla resumen de los valores, Figura 5

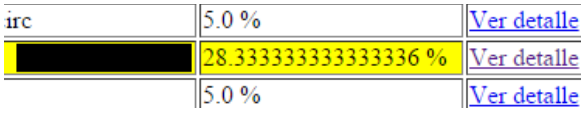

Figura 5. Tabla general comparación ejercicio 111

Si pinchamos en el enlace de la comparación marcada como "*warning*", se puede acceder al detalle del valor proporcionado por las métricas aplicadas en la comparación, para su análisis

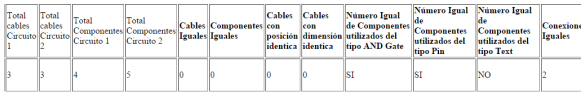

Figura 6. Detalle de la comparación de circuito *warning*

Claramente se pueden visualizar los valores que indican plagio después de la aplicación de las diversas técnicas.

Ahora se analiza el caso de una comparación que haya mostrado porcentajes de plagio mas elevados, en concreto un valor que ha dado como resultado un porcentaje de plagio superior al 50% :

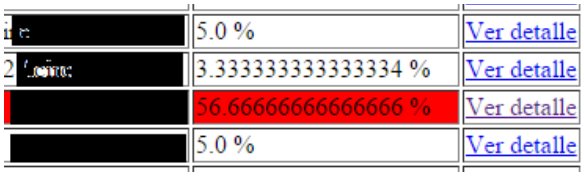

Figura 7. Resultado superior al 50%

Total cables Circuito 1	Total cables Circuito 2	Total Componentes Circuito 1	Total Componentes Circuito 2	Cables Iguales	Componentes Iguales	Cables con posición identica	Cables con dimensión identica	Número Igual de Componentes utilizados del tipo AND Gate	Número Igual de Componentes utilizados del tipo Pin	Conexiones Iguales	Componente Pin misma posición
3	3	4	4	0	0	2	0	SI	SI	2	2

Figura 8. Detalles de la comparación superior al 50%

Se puede destacar como el número de cables utilizados en los dos diseños han sido 3 y coinciden en posiciones en dos de ellos, se ha utilizado el mismo número de componentes de tipo “AND gate” y “Pin”. Las conexiones realizadas son de la misma manera en posiciones muy similares y además los componentes de tipo Pin tiene la misma localización. Todos estos atributos indican que el grado de similitud en los dos diseños es elevado.

A continuación se analiza un caso detectado correspondiente al 100% de similitud.

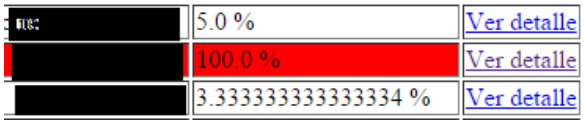

Figura 9. Resultado 100%

Total cables Circuito 1	Total cables Circuito 2	Total Componentes Circuito 1	Total Componentes Circuito 2	Cables Iguales	Componentes Iguales	Cables con posición identica	Cables con dimensión identica	Número Igual de Componentes utilizados del tipo AND Gate	Número Igual de Componentes utilizados del tipo Pin	Conexiones Iguales	Componente Pin misma posición	Compo AND misma posici
7	7	4	4	7	4	7	7	SI	SI	4	3	1

Figura 10. Detalles de la comparación de 100%

Se puede apreciar como el número de cables y componentes utilizados para el diseño son exactamente los mismos y todos sus atributos coinciden de manera exacta tanto en posiciones como en longitudes en el caso de los cables, como la forma como han sido conectados, de tal manera que se concluye que se trata exactamente del mismo diseño y por lo tanto se puede afirmar con rotundidad que se ha producido plagio.

6 Conclusión

El proceso de detección tiene como voluntad automatizar una serie de funcionalidades, de tal manera, que se realice una primera criba para detectar todos aquellos aspectos que son considerados como elementos que pueden indicar que se ha producido una copia entre los diseños de los distintos ejercicios.

Todos los criterios aplicados han sido seleccionados por considerarse relevantes para el estudio de los mismos según parámetros como la localización, la longitud o la forma como están conectados. Los resultados se han considerado como satisfactorios después de múltiples pruebas sobre ejercicios reales.

Si bien el resultado final puede ser considerado correcto en alto grado de casos, no deja de ser una generalización, y por lo tanto, es necesario la revisión posterior por parte del profesorado para corroborar que todos aquellos resultados que han sido detectados por la herramienta como susceptibles de ser plagio, efectivamente lo son. Para ello, es necesario que el profesor aplique los conocimientos y la experiencia acumulada para determinar de manera concluyente si efectivamente se trata de un caso de copia. La aplicación tiene como objetivo facilitar esta tarea pero en ningún caso puede ser considerado un elemento suficiente para determinar una copia.

Se puede considerar como excepción todos aquellos casos detectados como plagio exacto (100%), donde si se puede determinar una copia exacta.

Como mejoras y líneas futuras en la evolución del trabajo se podrían diferenciar claramente dos puntos.

Por un lado, mejoras en la usabilidad del programa, que pueden pasar por la posibilidad de configuración de peso en cada una de las métricas a la hora de calcular el porcentaje de plagio. Esto daría flexibilidad al usuario de cara a proporcionar mas o menos peso a una métrica determinada según la naturaleza del diseño analizado.

Por otro lado, una linea de mejora reside en la posibilidad de ampliar el algoritmo con otros criterios que determinen plagio en el circuito, añadiéndolos a los ya existentes o ampliar algunos de los ya existentes, como por ejemplo, la métrica que calcula las conexiones de los elementos, actualmente solo lo hace para parejas de componente – cable, esto podría ser ampliado con mas elementos a la serie y poco a poco ir ampliando el rango de selección de elementos.

Referencias

[1] Saul Schleimer, Daniel S. Wilkerson, Alex Aiken. *“Winnowing: Local Algorithms for Document Fingerprinting”*. Proceedings of the (2003) ACM SIGMOD *international conference on management of data,* pp 1 – 10

[2] Lutz Prechelt, Guido Malpohl, Michael Philippsen. *“Finding plagiarisms among a set of programs with Jplag”*. *Journal of Universal Computer Science*, vol. 8, no. 11. (2001), pp 1 - 23

[3] Cynthia Kustanto, Inggriani Liem. *“Automatic Source Code Plagiarism Detection”*. Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed

Computing, (2009). SNPD '09. 10th ACIS International Conference. pp 481 - 486

[4] Francisco Rosales, Antonio García, Santiago Rodríguez, José L. Pedraza, Rafael Méndez, and Manuel M. Nieto. *"Detection of Plagiarism in Programming Assignments". IEEE Transactions on Education*, (2008), Volum 51, Número 2, pp 174 – 183

[5] Kristina L. Verco and Michael J. Wise. *"Software for Detecting Suspected Plagiarism: Comparing Structure and Attribute-Counting Systems". Department of Computer Science University of Sydney, Australia (*2006), ACSE '96 Proceedings of the 1st Australasian conference on Computer science education (1996), ACM New York, NY, USA pp 81 - 88

[6] Wise, Michel J. *"YAP3: Improved detection of similarities in computer program and other texts". Department of Computer Science University of Sydney, Australia,* SIGCSE '96 Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education (1996) ACM New York, NY, USA, pp 130 - 134

[7] Xin Chen, Brent Francia, Ming Li, Brian McKinnon, and Amit Seker. *"Shared Information and Program Plagiarism Detection"*. Information Theory, IEEE Transactions Volume: 50, Issue: 7. (July 2004), pp. 1545 – 1551

[8] Alan Parker, James O. Hamblen. *"Computer Algorithms for Plagiarism Detection". Education, IEEE Transactions* (1989) Volume: 32, Issue: 2, pp 94 - 99

[9] Dominik Kasprowicz, HilekaanWada. *"Methods for automated detection of plagiarism in integrated-circuit layouts". Microelectronics Journal,* (05/2014), Volum 45, Número 9, pp 1212 - 1219

[10] Georg Pelz, Uli Roettcher. *"Circuit comparison by hierarchical pattern matching"*. Computer-Aided Design, (1991). ICCAD-91. *Digest of Technical Papers*. pp 290 - 293

[11] Makoto Takashima, Atsuhiko Ikeuchi, Shoichi Kojima, Toshikazu Tanaka, Tamaki Saitou and Jun-ichi Sakata. *"A circuit comparison system with rule-based functional isomorphism checking". 25th ACM/IEEE, Design Automation Conference.Proceedings* (1988), pp 512 – 516