Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

Algoritmos Externos de Detección de Plagio

por
Joshua Briceño Quiel
y
Andrea Solano González

Proyecto de graduación sometido a la consideración del
Comité Asesor para optar al grado de Licenciatura en
Computación e Informática

Ciudad Universitaria Rodrigo Facio
San Pedro, San José, Costa Rica

Noviembre de 2016

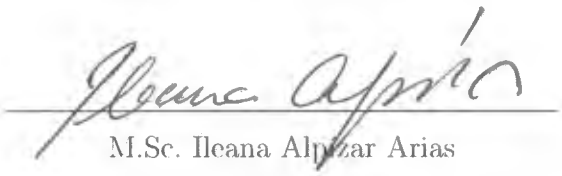
Este trabajo final de graduación ha sido aprobado por los miembros Tribunal Examinador como requisito para optar al grado académico de Licenciatura en Computación e Informática.

Miembros del Tribunal Examinador:

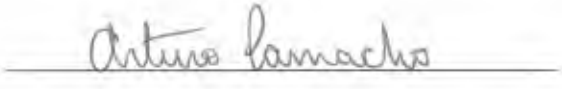
Dr. Carlos Vargas Castillo
Presidente del Tribunal

Dra. Gabriela Barrantes Sliesarieva
Profesor Lector del Tribunal

Dr. Juan José Vargas Morales
Director del Trabajo Final de Graduación



M.Sc. Ileana Alpízar Arias
Miembro del Comité asesor



Dr. Arturo Camacho Lozano
Miembro del Comité Asesor

Índice general

Índice de figuras

Índice de cuadros

Lista de abreviaturas

En este trabajo se utilizan las siguientes abreviaturas o acrónimos.

AOT:	Análisis de ocurrencia de términos.
MEV:	Modelos de espacio vectorial.
PAN:	Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection.
EHD:	Extracción de huellas digitales.

Resumen

Las tecnologías de información y comunicación juegan actualmente un papel trascendental en el acceso a la información. Una de sus ventajas es que permiten que datos de múltiples disciplinas estén al alcance de cualquier persona que tenga un dispositivo con acceso a internet. Sin embargo, un punto negativo de esta facilidad de acceso a la información es el incremento en la susceptibilidad de los datos ante el plagio.

Dada esta situación, se hace inminente la necesidad de crear mecanismos de detección de plagio automáticos que logren detectar documentos sospechosos de manera eficiente y en volúmenes de información que para un ser humano sería impráctico revisar.

En el presente proyecto se diseñaron y programaron dos algoritmos: extracción de huellas digitales (EHD) y análisis de ocurrencia de términos (AOT) por modelos de espacio vectorial. Los algoritmos se diseñaron con el fin de ayudar a detectar plagio literal, disfrazado y de ideas comparando los documentos bajo tres granularidades: un párrafo, tres párrafos y texto completo.

Se creó una tabla para comparar los algoritmos de EHD y AOT basados en su precisión y exhaustividad en las tres granularidades. Esto con el fin de averiguar si alguno de los algoritmos se desempeña mejor en una granularidad en particular. Para hacer un mejor análisis, adicionalmente se comparó la eficiencia (tiempo de ejecución) de los algoritmos. Con base en el análisis de los resultados, exponemos las conclusiones sobre cuál algoritmo es más eficaz o eficiente bajo una configuración de parámetros específica y contra los diferentes tipos de plagio.

Palabras clave

Algoritmo de detección, plagio, modelos de espacio vectorial, ocurrencia de términos, huella digital, fuentes de texto.

Capítulo 1

Introducción

Gracias al internet, la información sobre un amplia gama de temas es mucho más accesible en comparación con épocas pasadas. Lamentablemente, esto ha facilitado el plagio, y por el volumen de información, se ha hecho más ardua la tarea de detectarlo [Born, 2003]. De ahí ha surgido la necesidad de crear herramientas automáticas que ayuden a detectar el plagio, comparando un texto sospechoso contra colecciones de documentos, que a los seres humanos les resultaría impráctico revisar [Clough, 2000]. En este proyecto nos enfocaremos en la detección del plagio académico, que ocurre cuando se muestran como propias ideas, lenguaje u otro material original proveniente de fuentes a las cuales no se les da el reconocimiento debido [Meuschke and Gipp, 2013].

Existen distintos tipos de plagio; los más comunes son el plagio literal y el plagio disfrazado. El plagio literal se refiere a la copia, con poca o ninguna modificación, de partes de textos de fuentes no reconocidas [Maurer et al., 2006]. El plagio disfrazado es la práctica de disfrazar el texto original de otras fuentes, cambiando el orden de las palabras o parafraseando ideas sin dar referencia de su origen [Lancaster, 2003].

En los últimos años se han creado múltiples algoritmos y métodos de detección de plagio [Clough, 2000]. Hay dos tipos de ellos: los que realizan análisis interno y los que realizan análisis externo. Los del primer tipo solo requieren el texto en el que se buscará plagio, mientras que los del segundo tipo hacen comparaciones del texto sospechoso de plagio con otros textos [Potthast et al., 2014].

Un ejemplo de algoritmo que usa un análisis interno es el de detección por estilometría, el cual busca irregularidades o cambios abruptos en el estilo de escritura de un documento con el fin de identificar fragmentos sospechosos de plagio [Juola, 2006].

Ejemplos de algoritmos que usan un análisis externo son extracción de huellas digitales (EHD), análisis de ocurrencia de términos y la detección por citas bibliográficas. Este proyecto se enfoca en los primeros dos.

La técnica de EHD construye representaciones compactas (usualmente vectores de enteros) de los textos en una colección, las cuales utiliza para comparar y medir su similitud. Para construir la representación compacta o huella, se divide el texto en segmentos y estos se codifican. Luego un subconjunto de ellos se usa para medir la similitud con las huellas de otros documentos [Hoad and Zobel, 2003].

El análisis de ocurrencia de términos compara los textos en una colección con base en modelos de espacios vectoriales. Para este tipo de análisis, se cuenta la cantidad de veces que aparecen los distintos términos en una colección de documentos, luego se construye un vector por cada documento y un vector total para la colección. Posteriormente, se comparan estos vectores para medir la similitud entre los documentos [Wong et al., 1985, Salton et al., 1975].

En este proyecto se propuso identificar cuál de estos algoritmos tiene un mejor desempeño al detectar los diferentes tipos de plagio bajo las granularidades de un párrafo, de tres párrafos y de texto completo.

Para ayudar a identificar los resultados correctos y erróneos de los algoritmos, utilizamos los términos *verdaderos positivos* para referirnos a documentos correctamente detectados como plagio, *verdaderos negativos* para documentos en los que los algoritmos no encontraron plagio y verdaderamente no tienen plagio, *falsos positivos* para documentos que los algoritmos identificaron como plagio equivocadamente, por último, *falsos negativos* para documentos con plagio que los algoritmos no pudieron detectar.

Para evaluar la eficacia de los algoritmos, utilizamos las medidas de precisión y exhaustividad [Salton and McGill, 1986]. Los resultados arrojados por estos indican cuales casos o parejas de documentos se clasificaron como positivos. La precisión es el porcentaje de casos correctamente clasificados como plagio, entre todos los casos positivos y mide la certeza de que los resultados arrojados sean correctos. La exhaustividad indica qué porcentaje de todos los casos con plagio, fueron detectados por los algoritmos y mide la capacidad de los algoritmos para detectar los diferentes tipos de plagio. En la Figura 1.1 se observa una representación gráfica de ambas medidas, en la cual el semicírculo superior incluye todos los casos en los que no hay plagio, el semicírculo inferior, los casos en los que sí lo hay, y el trapecio completo incluye todos los casos que

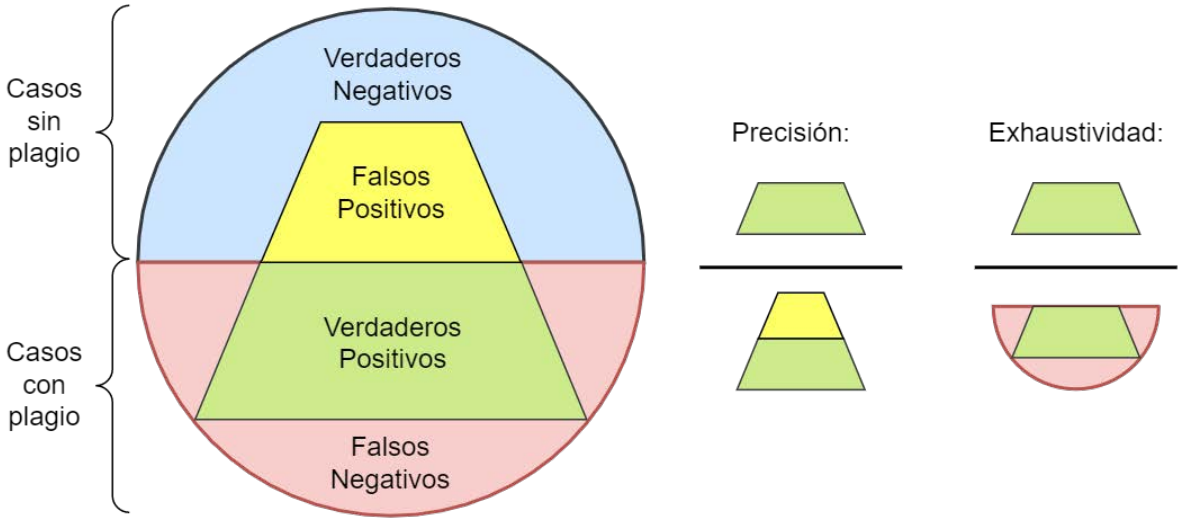


Figura 1.1: Representación gráfica de las medidas de precisión y exhaustividad.

fueron clasificados como positivos por los algoritmos. La precisión se calcula como

$$\text{Precision} = \frac{\text{VP}}{\text{VP+FP}}$$

y la exhaustividad como

$$\text{Exhaustividad} = \frac{\text{VP}}{\text{VP+FN}}.$$

donde VP es la cantidad de verdaderos positivos, FP la cantidad de falsos positivos y FN la cantidad de falsos negativos.

Las pruebas de los algoritmos se hicieron con un *corpus* de ejemplo de la página `http://pan.webis.de/`, que es una colección de miles de parejas de textos previamente catalogados como plagio o no plagio por expertos humanos. La organización *Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection*(PAN) a cargo de la página, promociona competencias anuales, en las cuales se juzgan y estudian diferentes algoritmos de detección de plagio [Stamatatos et al., 2015].

1.1. Objetivos

Este proyecto, en modalidad de proyecto de graduación, tiene como objetivo general lo siguiente:

Comparar la eficacia y eficiencia, para detectar plagio literal, plagio disfrazado y plagio de ideas, de los algoritmos de extracción de huella digital (EHD) y análisis de ocurrencia de términos (AOT) por modelos de espacio vectorial, en tres granularidades: párrafo, trío de párrafos y texto completo.

1.1.1. Objetivos específicos

Los objetivos específicos son los siguientes:

1. Programar una versión funcional de los algoritmos de EHD y AOT para la detección de plagio literal, disfrazado y de ideas.

2. Crear un estudio comparativo de la precisión, exhaustividad y eficiencia de las diferentes configuraciones de los algoritmos, con el fin de determinar cuál algoritmo y qué parámetros obtienen un mejor desempeño al detectar los tipos de plagio propuestos. Para esto fue necesario crear una base de datos que almacena los resultados obtenidos al aplicar los algoritmos *corpus* de ejemplo de PAN en diferentes granularidades con diferentes combinaciones de parámetros. Luego se analizaron estos resultados y se creo un estudio comparativo para encontrar los valores de precisión y exhaustividad más altos. Esto con el fin de poder dar recomendaciones que indiquen cuál algoritmo y que parámetros emplear según lo requiera el usuario.

1.1.2. Alcances y limitaciones

Para probar los algoritmos se utilizó el corpus de entrenamiento de la página `http://pan.webis.de/`, del cual no se incluyó la detección de plagio para parejas de textos en diferentes idiomas.

Capítulo 2

Marco teórico

2.1. Plagio

La Real Academia Española define plagio como "copiar en lo sustancial obras ajenas, dándolas como propias" [RAE, 2014]. Meuschke y Gippel definen plagio académico como el uso de ideas, lenguaje u otro material original de fuentes a las cuales no se les da el reconocimiento requerido por principios académicos [Meuschke and Gipp, 2013].

Dependiendo del método que se utilizó para copiar otras fuentes, el plagio se puede categorizar o dividir en diferentes tipos:

1. La copia literal de texto o *verbatim*, comúnmente llamado *copy & paste*. Es el tipo de plagio más fácil de detectar [Maurer et al., 2006].

2. El plagio disfrazado, que pretende evitar la detección e incluye:

   a) El plagio por parafraseo, que consiste en utilizar sinónimos de las palabras o cambiar la gramática u orden del texto original [Clough, 2000].

   b) El plagio de *batir y pegar*, que se refiere a la práctica de tomar párrafos o frases de distintas fuentes y copiarlas cambiando lo necesario para que unidas tengan sentido [Weber-Wulff, 2010].

3. El plagio por traducción, que consiste en traducir y apropiarse párrafos o frases tomadas de textos en otros lenguajes [Weber-Wulff, 2010].

4. El plagio de ideas. Este es el tipo de plagio más difícil de detectar con máquinas, ya que no necesariamente copia frases o párrafos de la fuente original. Consiste en apropiarse las ideas, estructura argumentativa o fuentes de otro trabajo [Maurer et al., 2006, Meuschke and Gipp, 2013].

2.2. Algoritmos de detección de plagio

Para hacer frente al incremento sustancial [Ercegovac and Richardson, 2004] de plagio académico, se han creado diversos métodos automáticos para ayudar a detectarlo de manera automática. Existen métodos de análisis interno y externo. Los métodos de análisis interno solo necesitan el documento en el que se buscará plagio para brindar un resultado. Los métodos externos dividen el proceso en dos partes: recolectar las fuentes contra las cuales se quiere comparar un documento y comparar el documento con las fuentes para identificar los pasajes que tienen plagio [Potthast et al., 2014].

La recolección de las fuentes se puede dar localmente en el sistema de archivos o se puede dar por medio de internet buscando en páginas y repositorios utilizando *bots* o servicios web. Los algoritmos generalmente ignoran las palabras muy utilizadas, como "el", "la", "los", "las" y "porque". Estas palabras aportan tan poco al contenido que se les denomina palabras vacías o *stopwords* [Rajaraman and Ullman, 2011].

2.2.1. Análisis de ocurrencia de términos

Los modelos de espacio vectorial (MEV) son modelos algebraicos utilizados para representar documentos como vectores de términos. Los MEV toman en cuenta los N términos en una colección de documentos y codifica cada documento como un vector disperso, registrando la cantidad de veces que aparece cada término en el documento [Wong et al., 1985, Salton et al., 1975]. Los vectores son dispersos porque cada término que aparezca en la colección, pero no en un documento específico, se representa con cero o nulo. Un ejemplo de estos vectores se puede apreciar en la Figura 2.1.

Los MEV utilizan una función (generalmente el coseno del ángulo entre los vectores) para determinar la similitud entre dos documentos. Un valor cercano a cero indica ortogonalidad o que no están relacionados y un valor cercano a uno indica que son casi idénticos. La mayoría de los algoritmos de MEV utilizan un solo modelo para codifi-

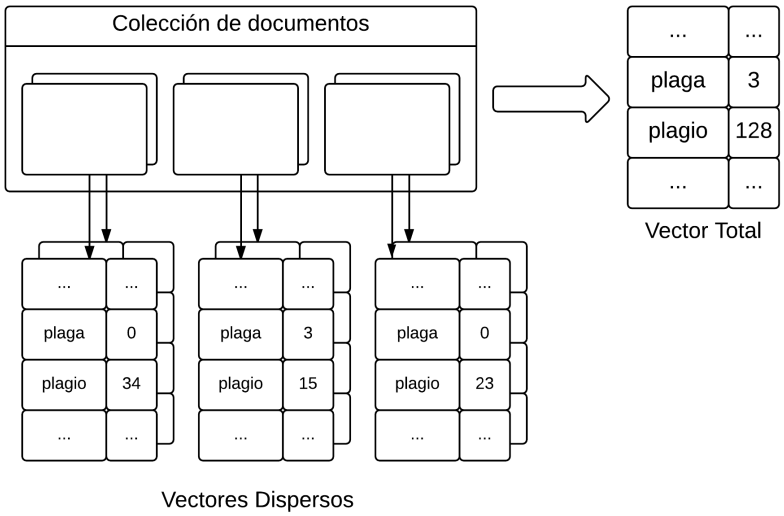


Figura 2.1: Ejemplo de un modelo de ocurrencia de términos.

car un documento entero, mientras que otros utilizan múltiples modelos que codifican oraciones o párrafos para hacer comparaciones más locales [Meuschke and Gipp, 2013].

Generalmente los MEV tienen una estrategia para dar más peso a los términos más relevantes. La más utilizada es la de *term frecuency inverse document frecuency* (tf-idf). Esta estrategia le da más peso a los términos que son poco comunes en la colección, luego el peso de cada palabra se multiplica en los vectores de los documentos, por la cantidad de veces que aparece dicha palabra. De esta manera se ayuda a controlar el hecho de que generalmente hay palabras más comunes que otras [Manning et al., 2008, Meuschke and Gipp, 2013]. El peso p_i se obtiene al calcular el logaritmo de la cantidad total de documentos (N) dividida entre la cantidad de documentos en los que la palabra en particular aparece (n_i).

$$p_i = \ln \frac{N}{n_i}.$$

2.2.2. Extracción de huellas digitales

El algoritmo de extracción de huellas digitales (EHD) construye una representación compacta o huella de los documentos en una colección, de forma tal que al comparar

las huellas de distintos documentos se pueda determinar el grado de similitud entre estos. Las huellas son colecciones de enteros que se construyen al aplicar fórmulas a las subcadenas de texto de un documento. Cada uno de estos enteros se conoce como minutiae (minutia en plural) [Hoad and Zobel, 2003].

Existen cuatro aspectos importantes en el proceso de EHD que necesitan consideración: la función que será utilizada para producir los minutia, la granularidad de la huella, la resolución de la huella y la selección de los segmentos de texto a ser incluidos en la huella [Hoad and Zobel, 2003].

El proceso que convierte las subcadenas en minutia tiene un efecto significativo en el algoritmo, por lo que debe escogerse para que cumpla ciertas propiedades que aseguren la eficacia y eficiencia del algoritmo [Hoad and Zobel, 2003]. Cada vez que se procesa una cierta subcadena, el entero resultante debe ser el mismo (reproductibilidad) [Manber, 1993]. Para evitar que dos subcadenas diferentes den como resultado el mismo minutiae, la función debe producir una distribución uniforme de enteros [Heintze, 1996]. Por último es importante que sea un proceso rápido, ya que será aplicado a una gran cantidad de subcadenas de texto. La Figura 2.2 muestra un ejemplo de EHD aplicado con *hash* criptográfico.

Una alternativa al método estándar de EHD, es utilizar una función *hash* perceptual en lugar de los *hash* comunes o criptográficos. Un *hash* perceptual de dos subcadenas similares produce dos minutia similares, a diferencia de los *hash* criptográficos, los cuales con una mínima variación en la entrada producen resultados muy diferentes (efecto avalancha [Feistel, 1973]). Se les llama *hash* perceptuales, porque se les suele usar para comparar videos, imágenes y audio, con el fin de medir su similitud, basándose en la percepción humana y no en la secuencia de bits que los conforman [Zauner, 2010].

La granularidad de la huella se refiere al tamaño de las subcadenas usadas para generar los minutia. La granularidad es un factor importante en la precisión y la exhaustividad al comparar documentos [Shivakumar and Garcia-Molina, 1999]. La granularidad se puede definir como el número de caracteres [Manber, 1993], palabras [Shivakumar and Garcia-Molina, 1999] u oraciones [Heintze, 1996] que conforman las subcadenas. Si la granularidad es muy alta se puede perder precisión al ocurrir más falsos positivos, ya que se estarían comparando subcadenas muy cortas que probablemente sean muy comunes. Por otro lado, si la granularidad es muy baja, se puede reducir la exhaustividad al ocurrir más falsos negativos, ya que se estarían comparando subcadenas muy

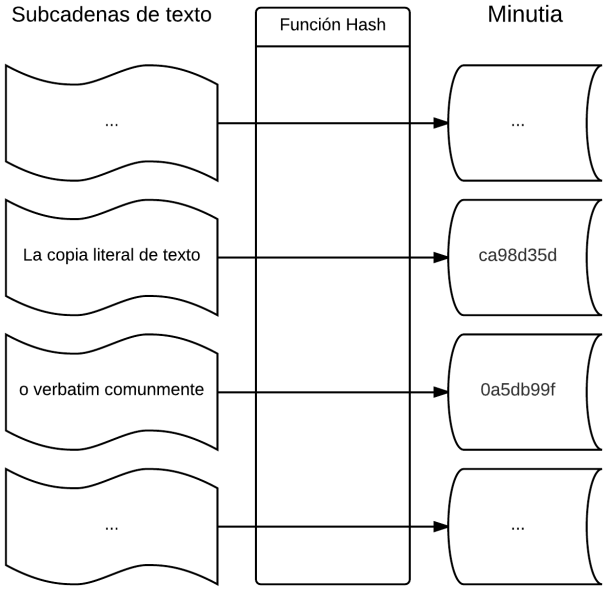


Figura 2.2: Ejemplo de generación de minutia a partir de subcadenas de texto con una función hash.

largas que difícilmente se repetirían entre los documentos [Heintze, 1996].

La resolución o tamaño de la huella es el número de minutia usado para representar un documento. La resolución esta limitada por la granularidad de la huella y el tamaño del documento. Es fácil notar que una mayor resolución puede brindar una mejor eficacia, pero el espacio y procesamiento necesario para hacer las comparaciones incrementa conforme aumente la cantidad de minutia [Hoad and Zobel, 2003].

Una vez fijadas la granularidad y resolución, se decide cuáles subcadenas son tomadas en cuenta para generar la huella [Heintze, 1996], esto tiene un efecto dramático en la eficiencia y eficacia del algoritmo [Shivakumar and Garcia-Molina, 1999]. La estrategia más simple es escoger cada subcadena en el documento. Otra estrategia toma cada subcadena no superpuesta y otras toman un número específico de subcadenas (pueden estar o no superpuestas). En todos los casos, es común descartar los espacios en blanco y los signos de puntuación.

Un método común para comparar documentos consiste en fijar un umbral máximo para el porcentaje de minutia que pueden compartir los documentos. Otro es fijar el

umbral máximo para la cantidad absoluta de minutia compartidos. Otro factor que se puede tomar en cuenta para medir la similitud de los documentos es la frecuencia con que un mismo minutiae aparece en los documentos [Meuschke and Gipp, 2013]. Estos métodos toman en cuenta los minutia presentes en ambos documentos, pero no la similitud de los demás minutia. Si se utiliza un *hash* perceptual, sería posible desarrollar otra medida de comparación que tome en cuenta la similitud de todos los minutia que conforman los documentos.

2.2.3. Detección de plagio basada en citas bibliográficas

La detección de plagio basada en citas bibliográficas es un algoritmo externo que compara la semántica de los documentos midiendo su similitud estructural por medio de citas bibliográficas. Desde hace mucho tiempo se sabe que las citas brindan valiosa información para la comparación semántica de documentos [Small, 1973], pero recientemente se ha descubierto que las citas son útiles para detectar plagio porque son independientes del idioma y son más difíciles de alterar que el resto del documento de una forma que no levante sospechas [Meuschke and Gipp, 2013]. Otra ventaja que tiene este método de detección es que requiere menos recursos computacionales. Esto se debe a que la cantidad de texto que se compara es mucho menor y además el número de documentos que comparten citas bibliográficas suele ser una fracción mucho más baja, por lo que se reduce drásticamente la cantidad de comparaciones [Meuschke and Gipp, 2013].

Un patrón de citas es una secuencia común de citas en dos documentos. Dentro de un patrón se pueden incluir citas intermedias que no aparecen en alguno de los documentos, pero se sospecha que fueron removidas para disfrazar el plagio [Gipp, 2014]. Para identificar y evaluar estos patrones, se toman en cuenta aspectos como el orden, la proximidad, la cantidad de citas compartidas, la fracción de citas y la probabilidad de que las citas ocurran en ambos documentos. Es más significativo para medir la similitud si dos citas poco comunes son compartidas, que si dos citas de fuentes muy comunes y reconocidas están en ambos documentos [Gipp, 2014]. Al ser este un método que compara la semántica de los documentos, tiene un mejor desempeño para encontrar el plagio disfrazado que otros algoritmos. Sin embargo, no es un sustituto de otros métodos, sino un complemento que puede ayudar a detectar plagio en caso de que

los documentos tengan citas y no sea suficiente la comparación por métodos basados en caracteres (como EHD y AOT por MEV). [Meuschke and Gipp, 2013]

2.3. Desempeño de los algoritmos de detección de plagio

Es difícil medir y comparar el desempeño real de los algoritmos de detección de plagio, porque generalmente sus autores no usan una forma estándar para evaluarlos o no prueban contra documentos con formatos similares. Un proyecto que combate esta falta de homogeneidad es la competencia anual internacional de detección de plagio *PAN-PC Uncovering Plagiarism, Authorship and Social Software Misuse* iniciada en 2009 y cuya próxima competencia evaluará programas para la detección de plagio desde enero hasta junio de 2017. En esta competencia se evalúan los programas de detección de plagio en un ambiente controlado y haciendo las mismas pruebas contra un mismo repertorio o *corpus* de documentos para todos los programas [Meuschke and Gipp, 2013].

Estas competencias anuales han mejorado a lo largo del tiempo sus *corpus* de prueba y técnicas para calificar los algoritmos. Para poder comparar el desempeño de los algoritmos modernos con los evaluados previamente, se aplican los algoritmos anteriores al nuevo *corpus* de prueba (cuando es posible) y los algoritmos nuevos al *corpus* anterior. Gracias a esta comparación se puede notar la mejora en el desempeño que han tenido los algoritmos de detección de plagio en los últimos años y el creciente interés en esta área de aplicación [Potthast et al., 2014]. La organización encargada de la competencia facilita un *corpus* de ejemplo que se utilizó para probar los algoritmos.

Capítulo 3

Metodología

En este capítulo se especifica el proceso utilizado en la implementación de los algoritmos y el desarrollo de de las pruebas para cumplir con lo propuesto en sección 1.1.1.

3.1. Primer objetivo específico

Para cumplir con el primer objetivo específico, se puso en funcionamiento una versión completa de los algoritmos de EHD y análisis de ocurrencia de términos (AOT).

1. Para el algoritmo de EHD se definieron los parámetros de entrada, entre ellos: utilización o eliminación de palabras vacías del documento, resolución, granularidad de la huella y terminación de las subcadenas de texto por carácter, palabra u oración. Una vez definidas estas características se procedió a probar el algoritmo asignándole a dichos parámetros diferentes valores, con el fin de documentar con cuáles de estos valores el algoritmo obtiene mejores resultados.

2. Para el algoritmo de AOT se utilizó el modelo de espacios vectoriales. Este modelo consiste en crear un vector que incluya cada término que aparezca en una colección de documentos. El valor de cada entrada corresponde al número de documentos en los que aparece el término asociado. Además, se crean los vectores de los documentos a comparar.

3.2. Segundo objetivo específico

Para completar el segundo objetivo específico se realizó un estudio comparativo de la precisión, exhaustividad y tiempo de ejecución de los algoritmos, con el fin de averiguar cuál de ellos es mejor contra un tipo particular de plagio bajo una granularidad específica.

1. Como primer paso se creó una base de datos que contiene los documentos preprocesados, con el fin de aligerar la comparación posterior. Se entiende como documentos preprocesados, los resultantes de aplicar los algoritmos de detección de plagio a los documentos, es decir, los vectores de términos en el algoritmo de AOT y las huellas en el algoritmo de EHD.

   Por ejemplo, a los documentos se les aplica el algoritmo de EHD en granularidad de párrafo inicialmente, luego en granularidad de tríos de párrafos y posteriormente en granularidad de texto completo. Como resultado se obtienen las huellas digitales de los documentos, las cuales se almacenan en la base de datos. Este proceso se repite, pero ahora con los vectores de términos del algoritmo de AOT.

2. Luego de completar las pruebas, se realizó un análisis de los datos de precisión, exhaustividad y tiempo de ejecución obtenidos. Una vez procesada esta información se determinó cuál de los algoritmos obtuvo una mayor precisión y exhaustividad en las diferentes granularidades contra un tipo específico de plagio. Se analizó el desempeño de los algoritmos al detectar plagio literal, plagio disfrazado y plagio resumen.

3.3. Metodología de las pruebas

El corpus de entrenamiento que se utilizó para hacer las pruebas incluye 1827 documentos sospechosos, 3230 documentos fuentes y 3 listas de parejas de documentos, una para cada uno de los diferentes tipos de plagio a probar, propuestos en el objetivo general de la sección 1.1. Se le llama documento sospechoso a los documentos que deben ser revisados por los algoritmos, para averiguar si tienen información extraída de los documentos fuente. Cada pareja de las listas esta compuesta de un documento fuente y un documento sospechoso. El documento sospechoso tiene texto que se extrajo

Cuadro 3.1: Fragmento de la lista de parejas de plagio con ofuscación aleatoria

```
suspicious-document00006.txt source-document02829.txt
suspicious-document00020.txt source-document01371.txt
suspicious-document00020.txt source-document03225.txt
suspicious-document00027.txt source-document01053.txt
suspicious-document00027.txt source-document01362.txt
suspicious-document00027.txt source-document02493.txt
suspicious-document00029.txt source-document00473.txt
suspicious-document00029.txt source-document00740.txt
suspicious-document00029.txt source-document00744.txt
suspicious-document00029.txt source-document01851.txt
suspicious-document00040.txt source-document02150.txt
suspicious-document00042.txt source-document00024.txt
```

del documento fuente correspondiente. Estas parejas se utilizaron para encontrar los verdaderos positivos y falsos negativos de los resultados de las pruebas.

La organización PAN, autor del corpus, utilizó en las listas el término *ofuscar*, que significa encubrir o alterar el significado de algo haciéndolo más confuso. Las tres listas son: plagio sin ofuscación (plagio literal), plagio con ofuscación aleatoria (plagio disfrazado) y plagio resumen (de apropiación de ideas). Las dos primeras listas las hicieron creando documentos con plagio utilizando un método automático, mientras que la lista de parejas de plagio resumen fue hecha a partir de documentos sospechosos creados a mano por PAN. En el cuadro 3.1 se puede apreciar el formato que siguen las listas, con un fragmento de la lista de parejas de ofuscación aleatoria. Como se puede apreciar, el documento sospechoso 20 está emparejado con los documentos fuente 1371 y 3225, lo que significa que obtuvo información de ambos documentos.

Las listas de parejas no incluyen todas las fuentes desde las cuales un documento sospechoso obtuvo información, lo que impide saber cuándo un resultado es un falso positivo. Por esta razón, se decidió agregar más parejas a la lista de plagio literal, para poder obtener un mejor aproximado de la precisión que logran los algoritmos. Se desconoce por qué PAN no incluyó las listas completas en el corpus de entrenamiento, pero se supuso que en las listas dejaron parejas de texto sospechoso-fuente que no eran tan obvias, luego de encontrar que muchos de los fuentes, que no estaban en las lista de plagio literal, tenían una similitud muy alta con los documentos sospechosos. Las nuevas parejas de documentos sospechoso-fuente se agregaron de la siguiente forma:

1. Por cada documento sospechoso, se obtuvo el valor mínimo y el valor promedio

de similitud con los documentos fuente con los que está emparejado en la lista. Estos valores se identificarán como similitud mínima con pareja (SMP) y similitud promedio con pareja (SPP).

2. Por cada documento sospechoso, se obtuvo el valor máximo y el valor promedio de similitud con los documentos fuente con los que no está emparejado en la lista, que tengan una similitud mayor a 0. Estos valores se identificarán como similitud máxima con documento no pareja (SMNP) y similitud promedio con documentos no pareja (SPNP).

3. Se calculó el promedio, entre todos los documentos sospechosos, de cada uno de estos cuatro valores y se construyó un gráfico de puntos.

4. Se revisaron varios documentos fuentes y sospechosos que no se encontraban en la lista de parejas y que obtuvieron un valor de similitud que sobrepasaba por mucho el promedio.

5. Si los documentos sospechosos habían tomado texto de estos documentos fuentes, se agregaron a la lista de parejas junto con otras parejas de documentos que habían obtenido valores similares.

6. El proceso se aplica a cada granularidad y hasta que no sobresalgan similitudes muy por encima del promedio.

En la mayoría de los casos solo se agregaron parejas una o dos veces. Se decidió no agregar más, para evitar agregar parejas incorrectas, que sesgarían los resultados de las pruebas. En la Figura 3.1 se muestran los SMP, SPP, SMNP y SPNP de cada documento sospechoso incluido en la lista de plagio literal. Las lineas horizontales, que se ven en el gráfico, representan los promedios de estos cuatro valores entre todos los documentos sospechosos. La Figura 3.2 muestra los mismos valores luego de introducir las nuevas parejas de texto.

Con el promedio de SMP y el promedio de SPNP, se obtuvo un rango de valores con los cuales probar los umbrales de plagio de los algoritmos. El umbral de plagio es el parámetro que indica el limite de similitud entre documentos, a partir del cual los algoritmos dan como resultado que el documento sospechoso obtuvo texto o ideas de un documento fuente (resultado positivo). El umbral de plagio para el algoritmo de AOT

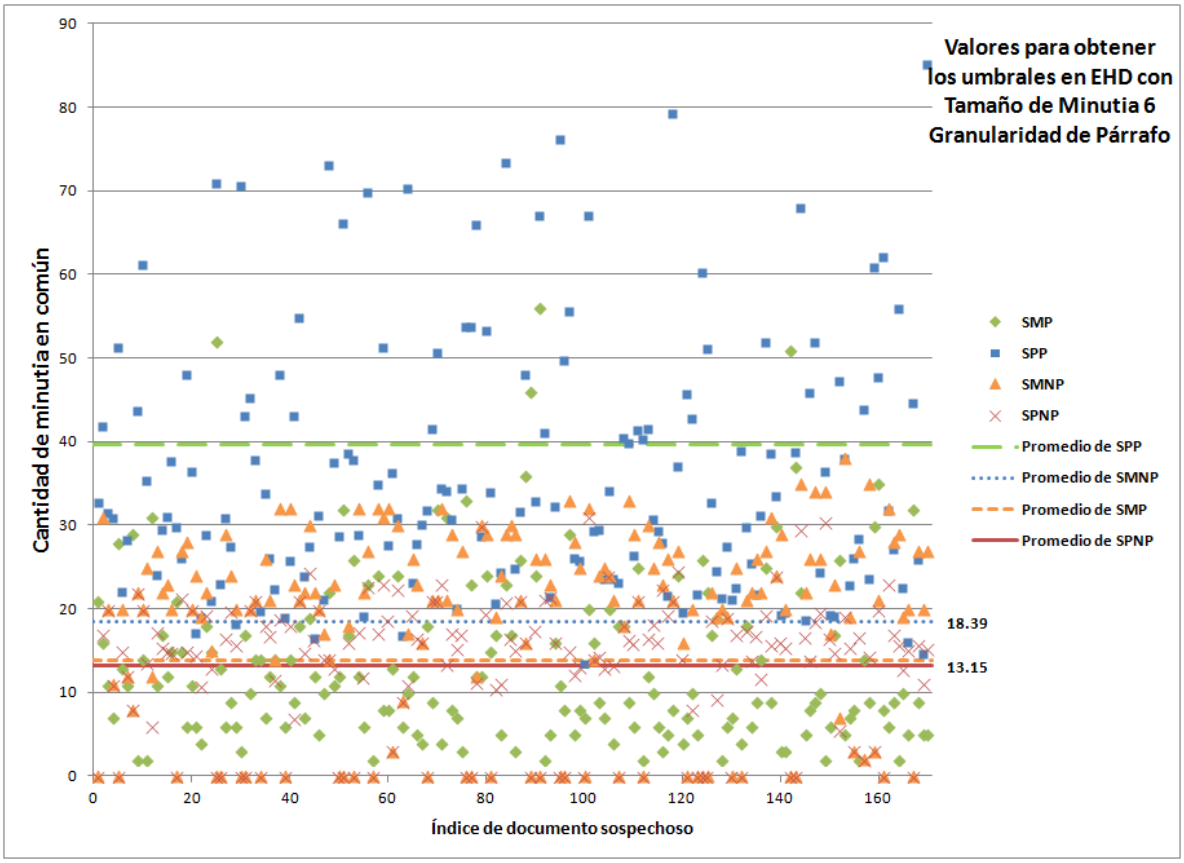


Figura 3.1: Gráfico de puntos de los documentos antes de agregar parejas.

es un numero racional entre 0 y 1, mientras que el umbral de plagio para el algoritmo de EHD es un numero natural.

Con ayuda de estos rangos de valores se pudo limitar las pruebas a valores en los cuales se esperaban altos resultados de precision o de exhaustividad. Por ejemplo, si se trata del algoritmo de AOT en granularidad de 3 párrafos y el rango está entre 0.19 y 0.33, si se sitúa el umbral de plagio en una similitud de 0.19 se obtienen la mayoría de los documentos fuente de los que se obtuvo texto (pocos falsos negativos y alta exhaustividad), pero con el costo de obtener muchos fuentes que no están en la lista de parejas (muchos falsos positivos, baja precisión). Si el umbral se sitúa en 0.33, se obtienen pocos fuentes de los que se obtuvo texto (más falsos negativos, poca exhaustividad), pero la mayoría de estos sí se encuentran en la lista de parejas (menos falsos positivos y alta precisión).

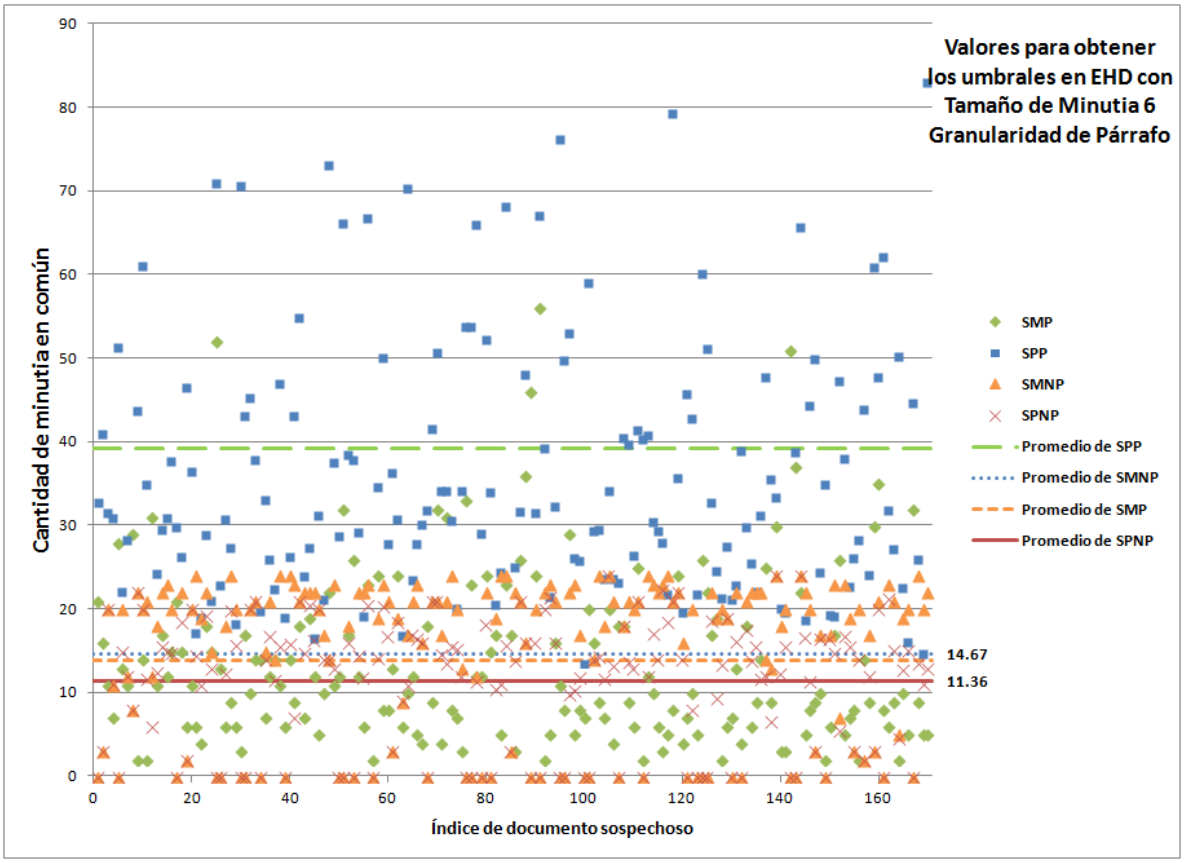


Figura 3.2: Gráfico de puntos de los documentos después de agregar parejas.

Para medir la eficacia de los algoritmos al detectar plagio disfrazado y plagio de ideas, se decidió utilizar las listas de parejas originales y calcular solo la exhaustividad, para evitar comprometer las pruebas con un sesgo, ya que discernir si ocurren estos tipos de plagio en los documentos es más propenso a interpretación.

Capítulo 4

Desarrollo

En este capítulo se describe el trabajo que se realizó para implementar los algoritmos, realizar las pruebas, elaborar los cuadros comparativos y analizar y graficar los resultados.

4.1. Implementación de los algoritmos

Los algoritmos se programaron utilizando el lenguaje Java, con el ambiente de desarrollo integrado NetBeans 8.0.1. Las pruebas de los algoritmos se realizaron de la misma forma. Los resultados se graficaron utilizando Microsoft Office Excel 2007.

4.1.1. Preparación de los documentos

Antes de aplicar los algoritmos de AOT y EHD, un algoritmo prepara los documentos, tanto fuentes como sospechosos, para dividirlos en párrafos y eliminar palabras vacías y números solos. Las palabras vacías se eliminan para que no influyan al comparar documentos, ya que estas no aportan contenido importante. Se separaran los documentos en párrafos para facilitar su manejo en las tres diferentes granularidades como se propuso en el objetivo general en la sección 1.1. Además, cuando se separan los documentos, se eliminan puntuaciones, espacios y cambios de línea innecesarios para evitar problemas a la hora de crear subcadenas de texto para el algoritmo de EHD.

Como primer paso de los algoritmos se toman los párrafos de los documento preparados y se agrupan en bloques según la granularidad elegida. Por ejemplo, con una

granularidad de 1 párrafo, un documento de 7 párrafos tendría 7 bloques, pero con una granularidad de 3 párrafos, el mismo documento tendría 3 bloques: 2 bloques de 3 párrafos y 1 bloque de 1 párrafo, por último con la granularidad de texto completo, el documento tendría un único de bloque de 7 párrafos.

4.1.2. Algoritmo de análisis de ocurrencia de términos

Por cada bloque de un documento, se crea un vector, el cual lista las diferentes palabras que aparecen en su respectivo bloque y la cantidad de veces que aparecen. Esto se hace para todos los documentos del corpus y el documento sospechoso. Luego de crear los vectores de los documentos, se crea el vector total del corpus, vector que lista todas las palabras en el corpus y la cantidad de documentos en los que aparecen estas palabras. Con estos valores se puede dar un peso a cada palabra utilizando la frecuencia inversa en documentos (en inglés *inverse document frecuency*), este peso p_i se obtiene al calcular el logaritmo de la cantidad total de documentos (N) dividida entre la cantidad de documentos en los que la palabra en particular aparece aparece (n_i).

$$p_i = \ln \frac{N}{n_i}.$$

Con esta fórmula se le da más peso a las palabras que son menos comunes en el corpus y que son más relevantes cuando se comparan documentos, ya que es más útil saber si dos documentos comparten palabras muy raras, a saber si comparten palabras que muchos documentos tienen en común. Se calcula con un logaritmo, para no crear más disparidad, entre palabras raras y muy raras, de lo que es necesario. Además, cuando una palabra es muy común entre los documentos, tanto que n es cercano a N, el logaritmo tiende a 0. Esto es similar a remover las palabras vacías, que no ayudan a diferenciar los documentos, reduciendo casi todo su peso.

Luego de obtener los pesos para cada palabra, se calcula la frecuencia de términos-frecuencia inversa en documentos (en inglés *term frecuency-inverse document frecuency*). Los pesos p_i se multiplican en los vectores de los documentos del corpus por la cantidad de veces que las palabras correspondientes aparecen (c_i). Con esto se obtienen los valores finales v_i para los vectores de cada bloque. La Figura 4.1 se muestra el vector

de un documento, que contiene la cantidad de veces que aparece cada palabra en dicho documento. El vector del corpus contiene el peso de cada palabra. Para obtener el vector final del documento, se calcula el producto punto entre el vector original y el vector del corpus. El vector final es el que se utiliza para comparar los documentos.

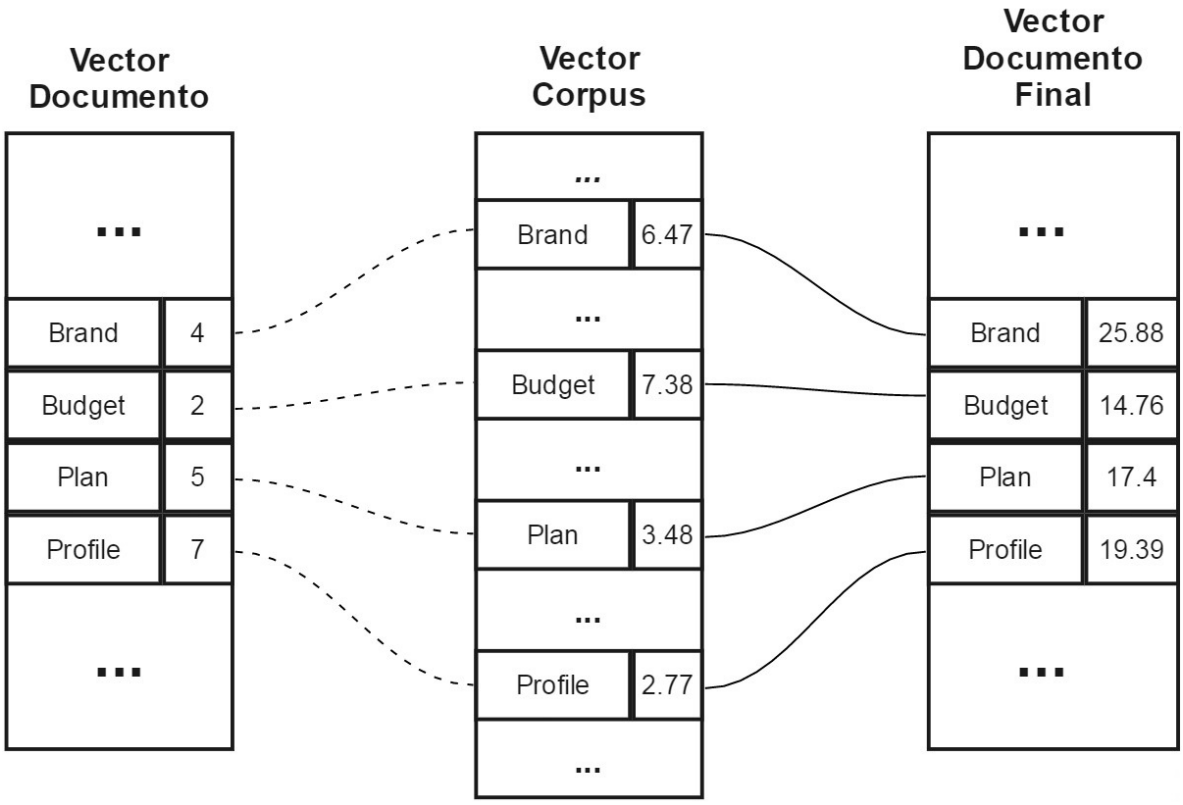


Figura 4.1: Ejemplo de construcción del vector final del bloque.

Para comparar dos documentos utilizando el algoritmo de AOT se calcula la similitud por coseno de sus vectores, con un resultado de cero los documentos son completamente diferentes y con un resultado de uno los documentos son iguales. Sean A y B los vectores a comparar y A_i y B_i los valores para una misma palabra i en los vectores correspondientes. La similitud por coseno se calcula como

$$\cos(\alpha) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}.$$

Cuando la granularidad es texto completo, para comparar dos documentos se obtiene

el coseno de los únicos vectores de ambos documentos y el resultado es la similitud entre estos. Cuando la granularidad es de 1 párrafo o de 3 párrafos, se compara cada vector de un documento con cada vector del otro documento, y el resultado máximo será considerado la similitud entre los documentos.

Estructuras de datos

Para facilitar el manejo de los vectores de palabras y sus respectivos valores, se utilizó la estructura de datos de Java *HashMap*. Este diccionario permite agregar llaves y valores de forma eficiente y facilita la búsqueda de dichas llaves y la recuperación de los valores respectivos. Un *String* de cada palabra se utiliza como llave y el valor correspondiente es de tipo *Double*. Como los documentos pueden tener una cantidad variable de bloques, para representar todos los vectores de palabras de un documento se utilizó la estructura de datos de Java *ArrayList*. En estos arreglos se pueden introducir, ordenadamente, la cantidad de diccionarios que sean necesarios para representar un documento. En el Cuadro 4.1 se muestra parte del código utilizado para crear la lista

Cuadro 4.1: Estructura de datos de MEV

```
//Se crea un diccionario de las palabras con la cantidad de
//veces que aparecen para cada bloque y se guardan en una lista.
iterator = blocks.listIterator();
ArrayList<HashMap<String,Double>> mevs = new ArrayList();
HashMap<String,Double> mev;
while(iterator.hasNext()){              //Por cada bloque del documento
    mev = new HashMap();                //se crea un diccionario y
    text = (String) iterator.next();
    words = text.split("\\s");          //se divide en palabras.
    for(String word : words){           //La ultima palabra de cada oracion
        if(word.contains(".")){         //contiene un punto que se elimina
            word = word.substring(0, word.lastIndexOf('.'));
        }
        if(word.length() > 0){
            if(mev.containsKey(word)){  //y se introducen al diccionario.
                mev.put(word, 1.0 + (double)mev.get(word));
            }else{
                mev.put(word,1.0);
            }
        }
    }
    mevs.add(mev);
}
```

de diccionarios que representa un documento. Como se ve a partir de la sexta línea de código, por cada bloque del documento, se crea un diccionario en el cual se cuenta la cantidad de veces que aparecen las palabras de su respectivo bloque.

Implementación de la comparación de vectores

Para lograr comparar y obtener un valor de similitud entre dos documentos, es necesario obtener los cosenos entre los vectores de dichos documentos. A diferencia de la teoría, no se agregaron todos las palabras encontradas en el corpus a todos los vectores (haciendo vectores dispersos) para lograr compararlos. No es necesario hacer esto porque las palabras que aparecen en solo uno de los vectores, producen que cuando se calcula el numerador, los productos de estas palabras sean 0. Para obtener el denominador se multiplica las raíces de las sumatorias de los valores de las entradas de ambos vectores al cuadrado. Si una o más palabras del otro documento no aparecen, no afectan el resultado de la sumatoria de su vector.

No tener que agregar todas las palabras a los vectores, favorece la eficiencia del algoritmo y disminuye la cantidad de memoria necesaria para ejecutar el programa. En el Cuadro 4.2 se muestra el código utilizado para comparar los vectores de dos documentos. Para obtener la medida de similitud, se compara cada diccionario de un documento con cada diccionario del otro documento, calculando el coseno de los vectores de palabras, como se puede ver entre las líneas 32 y 47 de este cuadro. El coseno más alto obtenido se considera como la similitud entre los documentos.

4.1.3. Algoritmo de extracción de huellas digitales

Para este algoritmo, cada bloque de un documento se divide en subcadenas de texto cuyo tamaño depende del tamaño de minutia elegido. Se toman todas las subcadenas superpuestas posibles del tamaño elegido siempre y cuando no tengan un punto en medio. Por ejemplo, con un tamaño de minutia de 4, la oración sin palabras vacías: “química orgánica rama química estudia clase numerosa moléculas mayoría contienen carbono.”, se divide de la siguiente forma: “química orgánica rama química”, “orgánica rama química estudia”, “rama química estudia clase”, “química estudia clase numerosa”, “estudia clase numerosa moléculas”, “clase numerosa moléculas mayoría”, “numerosa moléculas mayoría contienen”, “moléculas mayoría contienen carbono.”. Con esto

Cuadro 4.2: Comparación entre vectores de MEV

```
/*
* Compara los bloques de documentos y obtiene los cosenos correspondientes a
* su similitud. La similitud entre los documentos sera el maximo coseno
* encontrado.
*/
compareMev(List<HashMap<String,Double>> suspMev, List<HashMap<String,Double>> srcMev){
    HashMap<String,Double> srcBlock;
    HashMap<String,Double> suspBlock;
    double max = 0;
    double cosine;

    Iterator srcIterator;
    Iterator suspIterator = suspMev.iterator();
    while(suspIterator.hasNext()){
        suspBlock = (HashMap) suspIterator.next();  //Se obtiene cada diccionario
        srcIterator = srcMev.iterator();            //del documento sospechoso.
        while(srcIterator.hasNext()){
            srcBlock = (HashMap) srcIterator.next();//Para compararlos con cada
            cosine = cosineMev(suspBlock, srcBlock);//diccionario del documento fuente
            if(cosine > max){                       //y se obtiene el coseno maximo.
                max = cosine;
            }
        }
    }
    return max;
}

/*
* Calcula y obtiene el coseno entre los vectores representados por los
* diccionarios de los bloques de dos documentos.
*/
cosineMev(HashMap<String,Double> mev1, HashMap<String,Double> mev2){
    Set<String> terms = new HashSet();      //Se crea un conjunto de todas las
    terms.addAll(mev1.keySet());            //palabras de ambos bloques.
    terms.addAll(mev2.keySet());
    Iterator iterator = terms.iterator();   //Se itera sobre las palabras
    double numerator = 0.0, denominator1 = 0.0, denominator2 = 0.0, v1, v2;
    String term;
    while(iterator.hasNext()){              //y se obtienen los valores
        term = (String) iterator.next();    //encontrados en ambos mapas
        v1 = (double) mev1.getOrDefault(term,0.0);  //Si no existe la palabra
        v2 = (double) mev2.getOrDefault(term,0.0);  //se le asigna un valor de 0.
        numerator += v1*v2;
        denominator1 += Math.pow(v1, 2); denominator2 += Math.pow(v2, 2);
    }//por ultimo se introducen los valores en la formula correspondiente al coseno.
    return numerator/(Math.sqrt(denominator1)*Math.sqrt(denominator2));
}
```

se puede ver que, al comparar las huellas digitales, si ambos documentos tienen largas cadenas de texto idénticas o con fragmentos idénticos, van a compartir varias minutia y por ende tener una mayor similitud.

Luego se codifican las subcadenas de texto utilizando un algoritmo *hash* y se agrupan para formar la huella digital del bloque. Se toma la resolución completa de la huella (no se limita el número de subcadenas), con la excepción de subcadenas de texto repetidas, las cuales no se vuelven a introducir en la huella del bloque. Un documento tiene una cantidad de huellas igual al número de bloques en que se divide, por lo que para granularidad de texto completo, como cada documento solo tiene un bloque, en ese caso cada documento tiene una única huella. A las subcadenas de texto se les aplica un algoritmo *hash* para facilitar su comparación y como garantía de que mantienen su orden dentro de la subcadena. Como función *hash* se utilizó el algoritmo de encriptación SHA-256.

Estructuras de datos

Las minutia resultantes de aplicar la función *hash* a las subcadenas de texto se almacenan en arreglos de byte (byte[]) y estos arreglos de byte se almacenan en estructuras de datos *ArrayList* que representan las huellas de los bloques de un documento. Como los documentos pueden tener más de un bloque, el documento completo se representa con un *ArrayList<ArrayList<byte[]>>*. En el Cuadro 4.3 se muestra parte del código que se utilizó para construir las huellas. Por cada bloque del documento, se crea una lista que contiene las minutia resultantes de aplicar el algoritmo de *hash* a las subcadenas de texto. Para lograrlo, se agrupan las palabras según el tamaño de minutia elegido, de manera que no queden puntos en medio de las agrupaciones. Luego, se les aplica el algoritmo de *hash* y se guardan en las listas de los bloques a los que pertenecen.

Implementación de la comparación de huellas

Para comparar dos documentos utilizando EHD, se compara cada huella del documento sospechoso contra cada huella del documento fuente, en cada comparación se obtiene un entero que representa la cantidad de minutia que dos huellas tienen en común. Por cada huella del documento sospechoso, se guarda la cantidad máxima de minutia que comparte con alguna de las huellas del documento fuente. Por último, el

Cuadro 4.3: Estructura de datos de EHD

```
//Se dividen los bloques en hileras de texto  y se codifican.
//Si se encuentra un punto antes de que las hileras tengan el tamano
//minimo necesitado, se descarta y se continua despues del punto.
MessageDigest md = MessageDigest.getInstance("SHA-256");
iterator = blocks.listIterator();
ArrayList<byte[]> blockMinutia;
boolean foundPeriod = false;
byte[] minutia;
while(iterator.hasNext()){              //Por cada bloque del documento se crea una
    blockMinutia = new ArrayList();     //lista que va a contener los minutia
    text = (String) iterator.next();    //resultantes de codificar el texto.
    words = text.split("\\s");          //Primero se separan las palabras
    text = "";
    for(int j=0; j<words.length; ++j){  //Para agruparlas en grupos del tamano de
    minutia
        if ((j+minutiaSize) < words.length){
            for(int k=0; k < minutiaSize; ++k){
                foundPeriod = (words[j+k].contains(".") && words[j+k].length() > 1
                && k != minutiaSize -1);
                if(foundPeriod){    //Si encuentra un punto entre un grupo de palabras
                    j = j+k;        //Salta a la palabra despues del punto y
                    k = minutiaSize;//fuerza la salida del ciclo.
                }else{              //Si el punto esta al final de la subcadena
                    if(words[j+k].contains(".") &&
                       words[j+k].length() > 1){
                        text += " " +words[j+k].substring(0, words[j+k].lastIndexOf('.
    '));
                    }else{          //o si no hay, se agrupan las palabras normalmente
    .
                        text += " " +words[j+k];
                    }
                }
            }
            if(!foundPeriod){       //Si la subcadena no tiene un punto en medio
                md.update(text.trim().getBytes("UTF-8"));   //Se codifica
                minutia = md.digest();  //Se guarda en un vector de bytes
                if(!findMinutia(blockMinutia,minutia)){
                    blockMinutia.add(minutia);
                }
            }
            text ="";
        }
    }
    fingerprint.add(blockMinutia);
}
```

máximo de las cantidades de todas las huellas del documento sospechoso se toma como el valor final de similitud entre los documentos. En los cuadros 4.4 y 4.5 se muestra el código utilizado para la comparación de las huellas. Para obtener la medida de simili-

Cuadro 4.4: Comparación entre huellas de EHD, parte 1

```
/*
* Devuelve las cantidades maximas de minutias coincidentes entre cada bloque
* de la huella sospechosa y los bloques de la huella de un documento fuente.
*/
compareFingerprints(ArrayList<ArrayList<byte[]>> suspFp, ArrayList<ArrayList<byte[]>>
    srcFp){
    ArrayList<byte[]> block;
    Iterator blockIterator = suspFp.listIterator();
    ListIterator minutiaIterator;

    Integer[] maxSuspMatches = new Integer[suspFp.size()];
    Integer[] srcBlockMatches = new Integer[srcFp.size()];

    int suspBlock = 0;
    int srcBlock;

    byte[] minutiae;

    while(blockIterator.hasNext()){             //Itera sobre bloques del sospechoso
        block = (ArrayList<byte[]>) blockIterator.next();
        minutiaIterator = block.listIterator();
        Arrays.fill(srcBlockMatches,0);
        while(minutiaIterator.hasNext()){       //y sobre las minutia de los bloques.
            srcBlock = 0;
            minutiae = (byte[]) minutiaIterator.next();
            //Ahora se busca el minutiae en cada bloque
            for(boolean found: findMinutiae(minutiae, srcFp)){
                if(found){
                    ++srcBlockMatches[srcBlock];//Si se encuentra, incrementa la
                }                               //la cantidad de coincidencias
                ++srcBlock;                     //para el bloque en particular.
             }
        }
        //Ahora guarda el maximo de coincidencias entre el bloque sospechoso
        //y los bloques del fuente.
        maxSuspMatches[suspBlock]= Collections.max(Arrays.asList(srcBlockMatches));
        ++suspBlock;                            //Pasa al siguiente bloque sospechoso.
    }
    //El resultado es el maximo entre todos los bloques del sospechoso.
    return Collections.max(Arrays.asList(maxSuspMatches));
}
```

Cuadro 4.5: Comparación entre huellas de EHD, parte 2

```
/*
* Devuelve un vector de booleanos (un booleano por bloque del fuente)
* que indican si el minutiae fue encontrada en los bloques del fuente.
*/
findMinutiae(byte[] suspMinutiae, ArrayList<ArrayList<byte[]>> srcFp){
    boolean[] found = new boolean[srcFp.size()];
    //Cada entrada del vector representa si se econtro el minutiae en un bloque
    //Por ejemplo, "found[0] = true" significa que se encontro en el bloque 0.
    byte[] srcMinutiae;
    int i =0;

    ArrayList<byte[]> blockMinutia;
    Iterator iterator = srcFp.listIterator();
    ListIterator minutiaIterator;
    while(iterator.hasNext()){          //Itera sobre los bloques
        blockMinutia = (ArrayList<byte[]>) iterator.next();
        minutiaIterator = blockMinutia.listIterator();
        while(minutiaIterator.hasNext() && !found[i]){
            srcMinutiae = (byte[]) minutiaIterator.next();
            if(Arrays.equals(suspMinutiae, srcMinutiae)){   //Compara las minutia
                found[i] = true;
            }
        }
        ++i;
    }
    return found;
}
```

tud entre dos documentos, se cuenta la cantidad de minutia que comparte cada bloque de un documento con cada bloque del otro documento, como se puede ver entre las lineas 18 y 32 del Cuadro 4.4. El máximo de minutias compartidas entre dos bloques, se considera la similitud entre los documentos.

4.1.4. Almacenamiento de las huellas y vectores

Para mejorar la eficiencia de los algoritmos, una vez que los vectores de los documentos y del corpus se calculan y las huellas de los documentos se crean, estos se guardan en una base de datos evitando repetir el proceso de crearlos cada vez que se quiera verificar un documento sospechoso.

4.2. Resultados de las pruebas

Los resultados de las pruebas se dividieron según algoritmo, tipo de plagio, umbral de plagio y granularidad. El algoritmo de EHD tiene un parámetro adicional, el tamaño de minutia. Para poder obtener un aproximado de la precisión de los algoritmos al momento de detectar plagio literal se hicieron pruebas utilizando la lista de parejas en la cual se incluyeron más entradas y se compararon con los resultados de las pruebas hechas utilizando la lista de parejas originales del corpus. En los gráficos de las siguientes subsecciones se utilizan lineas de mejor ajuste para ayudar a visualizar los cambios en los resultados a medida que se cambian los parámetros de las pruebas.

4.2.1. Plagio literal

Este tipo de plagio es común y fácil de identificar con algoritmos, principalmente cuando se copian frases u oraciones completas de manera idéntica al texto fuente. El mayor problema es evitar la generación de falsos positivos que ocurren cuando se baja mucho el umbral de plagio.

Algoritmo de análisis de ocurrencia de términos

Comenzando con la lista de parejas modificada, se puede apreciar, en la Figura 4.2, que a medida que el umbral de plagio aumenta, también aumenta la precisión, pero disminuye la exhaustividad. Además, se puede distinguir que, utilizando granularidad de párrafo, se obtiene una mayor exhaustividad en comparación a las otras granularidades en el mismo umbral. En umbrales bajos, la granularidad de texto completo obtiene más precisión, pero en umbrales más altos, la granularidad de párrafo obtienen tanto mayor precisión como mayor exhaustividad, mostrando ser la mejor opción en ambos aspectos.

En el corpus, las parejas en las que ocurre plagio literal aparecen bajo una categoría llamada plagio sin ofuscación. Utilizando la lista de parejas original de esta categoría, se obtienen resultados similares a los obtenidos con la lista de parejas modificada. Esto indica que agregar nuevas parejas, para obtener un aproximado de la precisión, no alteró el comportamiento normal de los resultados obtenidos de exhaustividad. En la Figura 4.3 se ve que, de igual manera que en la Figura 4.2, la granularidad de párrafo produce mejores resultados de exhaustividad, mientras que la granularidad de texto

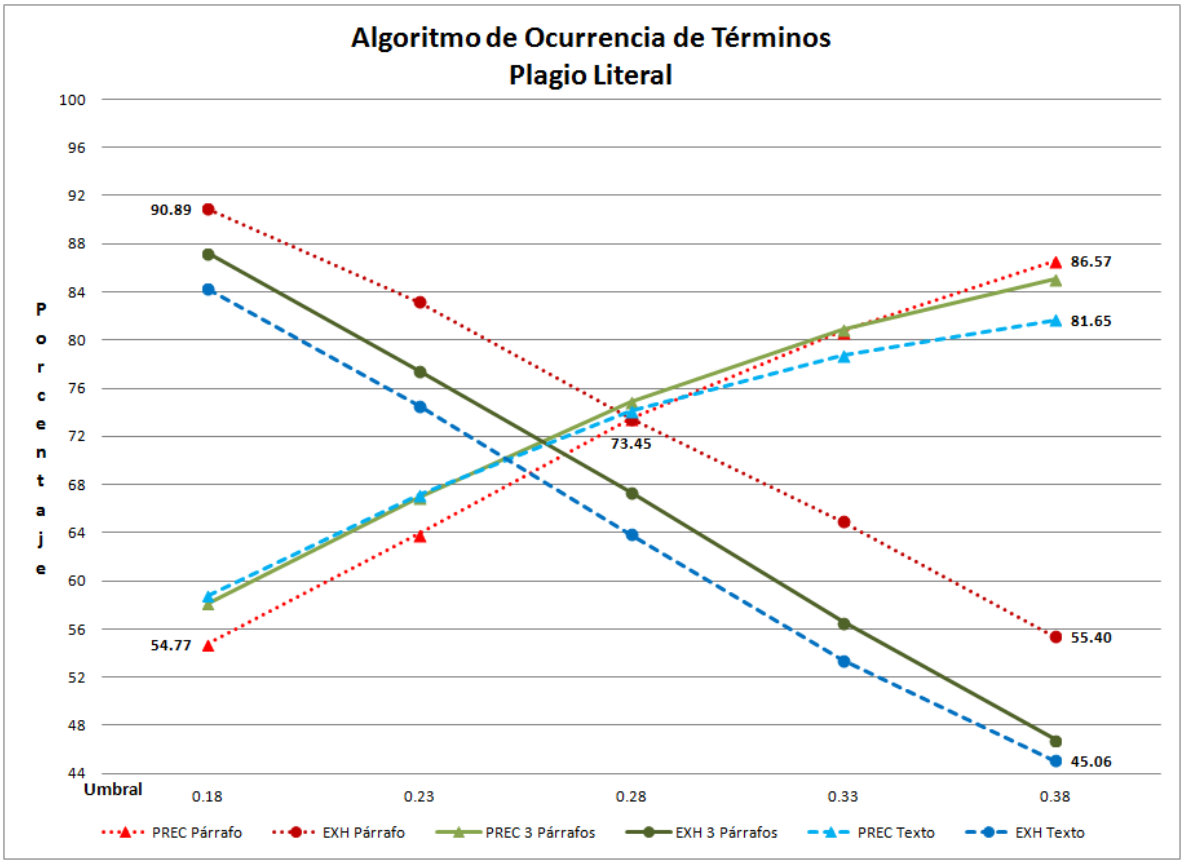


Figura 4.2: Resultados AOT con granularidad de texto completo utilizando la lista de parejas modificada.

completo produce resultados más bajos, dejando la granularidad de tres párrafos con valores intermedios.

Algoritmo de extracción de huella digital

Utilizando la lista de parejas modificada, se producen resultados que muestran diferencias entre los algoritmos como se puede apreciar en la Figura 4.4. A diferencia de los resultados del algoritmo de AOT, la granularidad de párrafo no llega a ser superior en ambos rubros de medición. La principal diferencia radica en que la granularidad de párrafo producen resultados más altos de precisión en todos los umbrales probados, mientras que la granularidad de texto completo produce resultados más altos de exhaustividad. Además, la granularidad de texto completo muestra menos separación

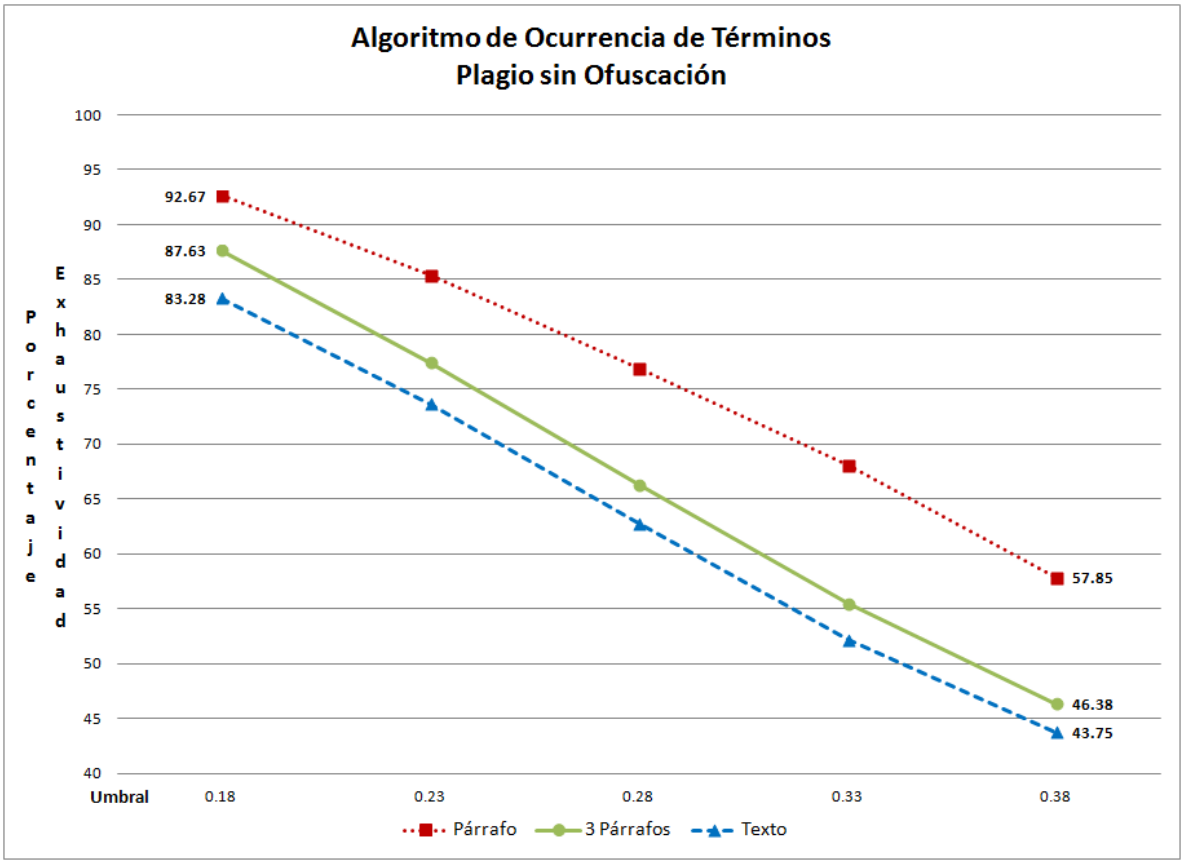


Figura 4.3: Resultados de AOT con granularidad de texto completo utilizando la lista de parejas original.

entre ambos valores, mientras que la granularidad de párrafo favorece la producción resultados altos de precisión, pero bajos de exhaustividad. Este caso específico utiliza EHD con un tamaño de minutia de 5 palabras. El último aspecto importante que se encontró es que a medida que se incrementa el tamaño de minutia incrementa la precisión, pero disminuye la exhaustividad y por ende la separación entre ambos resultados se agranda. En general, utilizar el algoritmo de EHD brinda resultados más balanceados que el algoritmo de AOT, principalmente cuando se utiliza granularidad de texto completo. Los gráficos que muestran los resultados de utilizar el algoritmo de EHD con tamaños de minutia 6 y 7 para detectar plagio literal se encuentran en el apéndice, en las figuras A.1 y 1.2.

Los resultados obtenidos al usar la lista de parejas original de nuevo muestran que

incluir las parejas adicionales no cambió el comportamiento general de los resultados. Esto se puede apreciar en la Figura 4.5. Este gráfico muestra los resultados de utilizar un tamaño de minutia de 7 palabras. El algoritmo de EHD no pierde exhaustividad tan rápidamente al incrementar el umbral de plagio, como el algoritmo de AOT. La constante que se mantiene es que a medida que incrementa el umbral de plagio, disminuye la exhaustividad y, aunque no se calcule cuando se usan las listas originales del corpus, se infiere que la precisión aumenta. Este comportamiento ocurre para ambos algoritmos y con todas las configuraciones de parámetros.

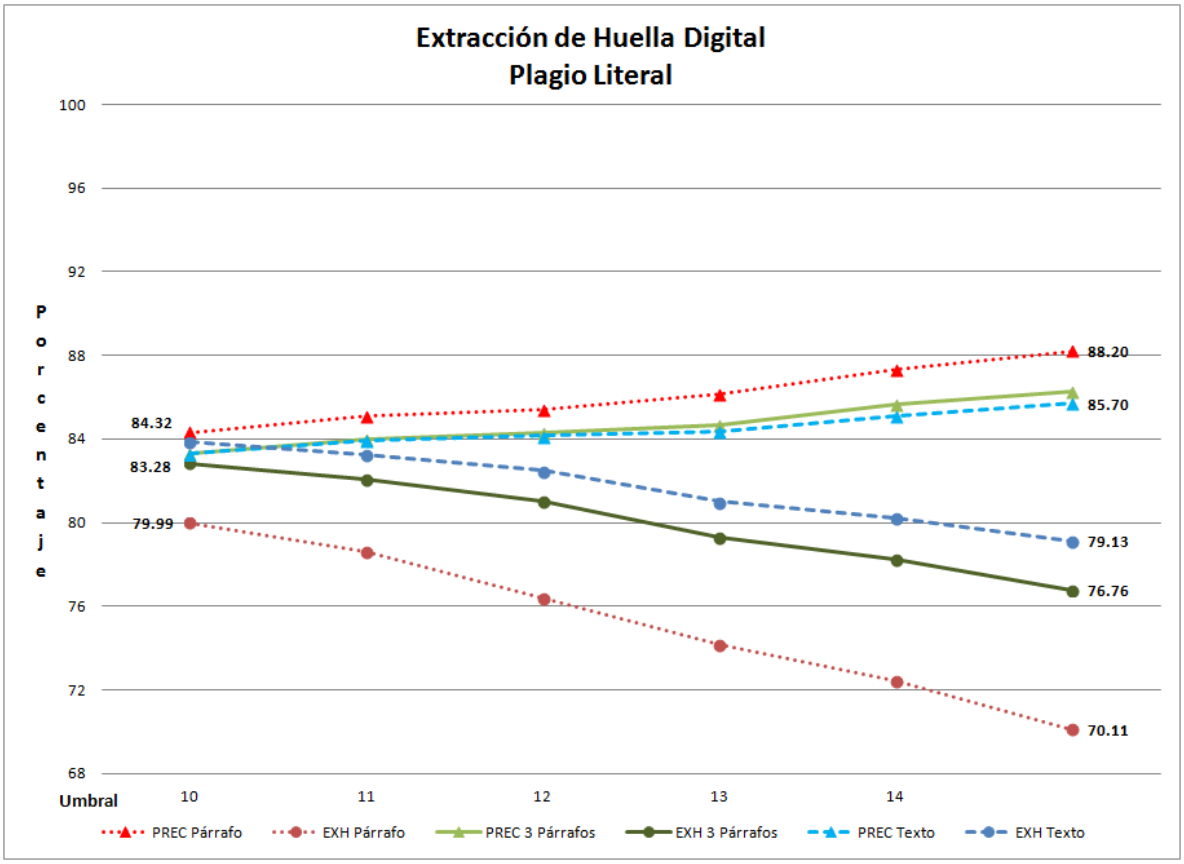


Figura 4.4: Resultados de EHD con tamaño de minutia de 5, utilizando la lista de parejas modificada.

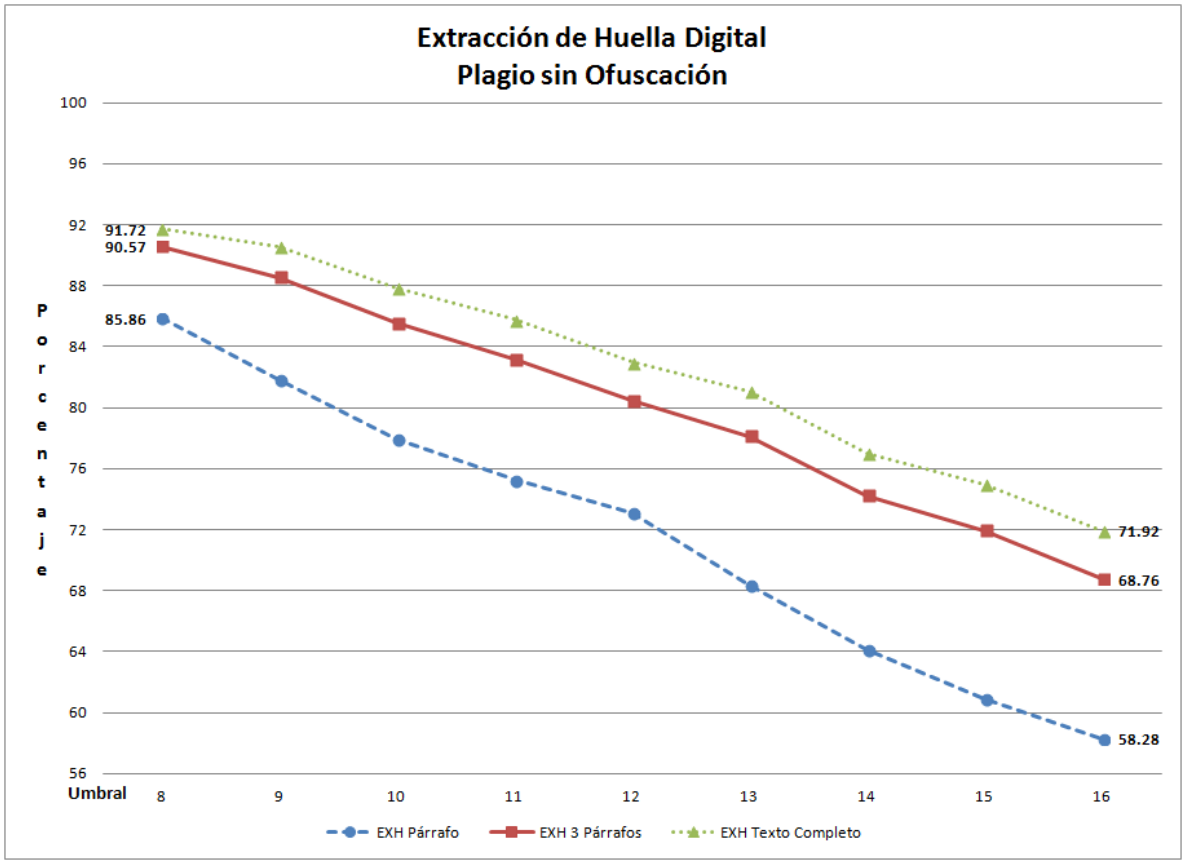


Figura 4.5: Resultados de EHD con tamaño de minutia de 7, utilizando la lista de parejas original.

4.2.2. Plagio disfrazado

Este tipo de plagio aparece en el corpus como los documentos a los que se les aplicó una ofuscación aleatoria. Es decir, documentos a los que se les cambiaron, agregaron o quitaron algunas palabras de manera aleatoria, para dificultar la detección de plagio y simular la forma en que ocurre el plagio disfrazado en la vida real.

Algoritmo de análisis de ocurrencia de términos

En la Figura 4.6 se pueden ver buenos resultados para el algoritmo de AOT, el cual no pierde demasiada exhaustividad a la hora de detectar plagio disfrazado. Esto se debe a que el orden de las palabras no es importante a la hora de comparar documentos

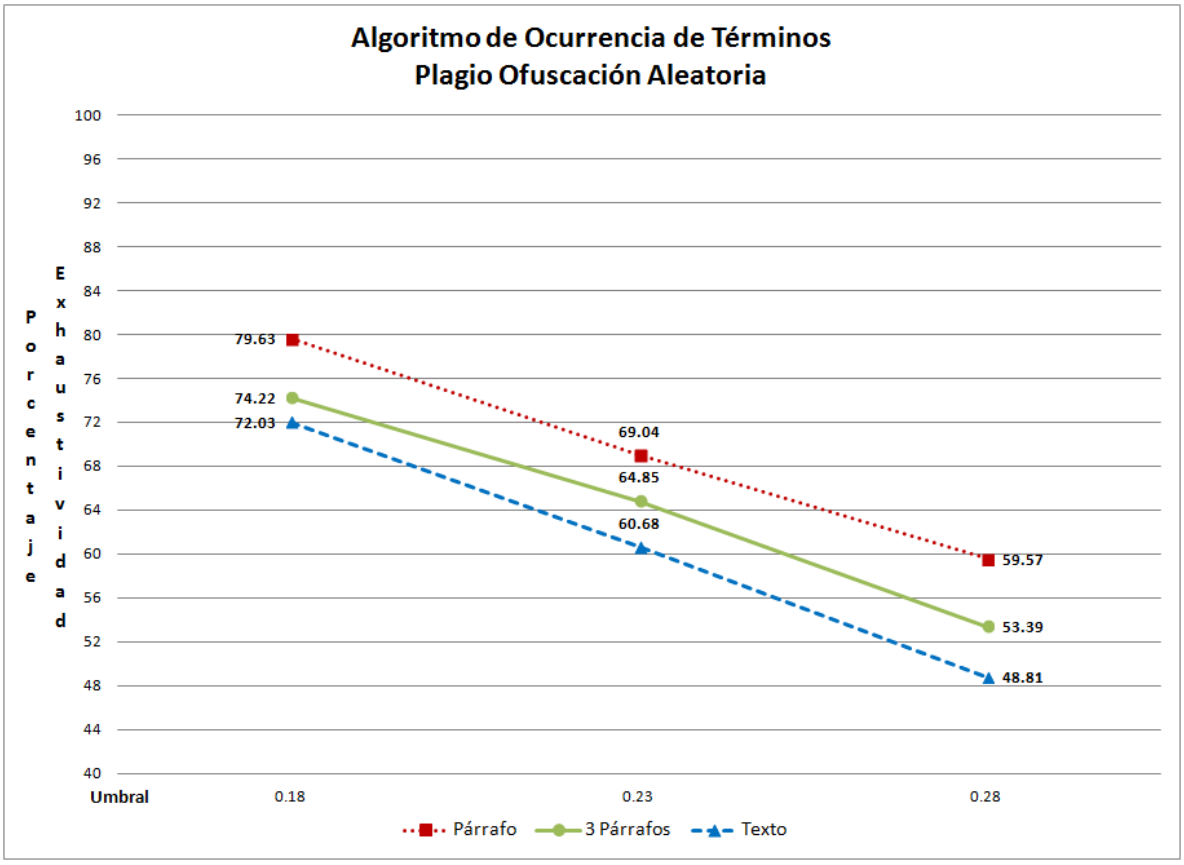


Figura 4.6: Porcentajes de exhaustividad del algoritmo de AOT con plagio disfrazado o de ofuscación aleatoria.

con este algoritmo. Además, en los casos reales de plagio disfrazado, las palabras que se cambian más fácilmente de esta forma son palabras vacías, las cuales se filtran de antemano durante la preparación de los documentos. Sin embargo, se nota que la exhaustividad decae mucho más rápidamente en comparación con la detección de plagio literal, a medida que se incrementa el umbral de plagio.

Algoritmo de extracción de huella digital

El algoritmo de EHD, al depender del orden de las palabras para formar sus minutia, se ve sumamente afectado con el plagio disfrazado, ya que el mínimo cambio de una palabra significativa, hace que se generen minutia totalmente diferentes. Los resultados se pueden observar en la Figura 4.7, en la cual se utiliza la configuración de tamaño

de minutia 5 con granularidad de texto completo, que es la que produce mejores resultados de exhaustividad para la detección de plagio literal. Sin embargo, para detectar plagio disfrazado, no logra detectar ni un 3 % de las parejas utilizando el umbral más bajo probado anteriormente. De esta forma queda claro lo impráctico que es usar el algoritmo de EHD para la detección de plagio disfrazado. Aún así, es probable que si se usan tamaños de minutia más pequeños y umbrales más bajos, se obtenga una mayor exhaustividad, pero al ser la pérdida de la precisión tan alta, los resultados no serían fiables.

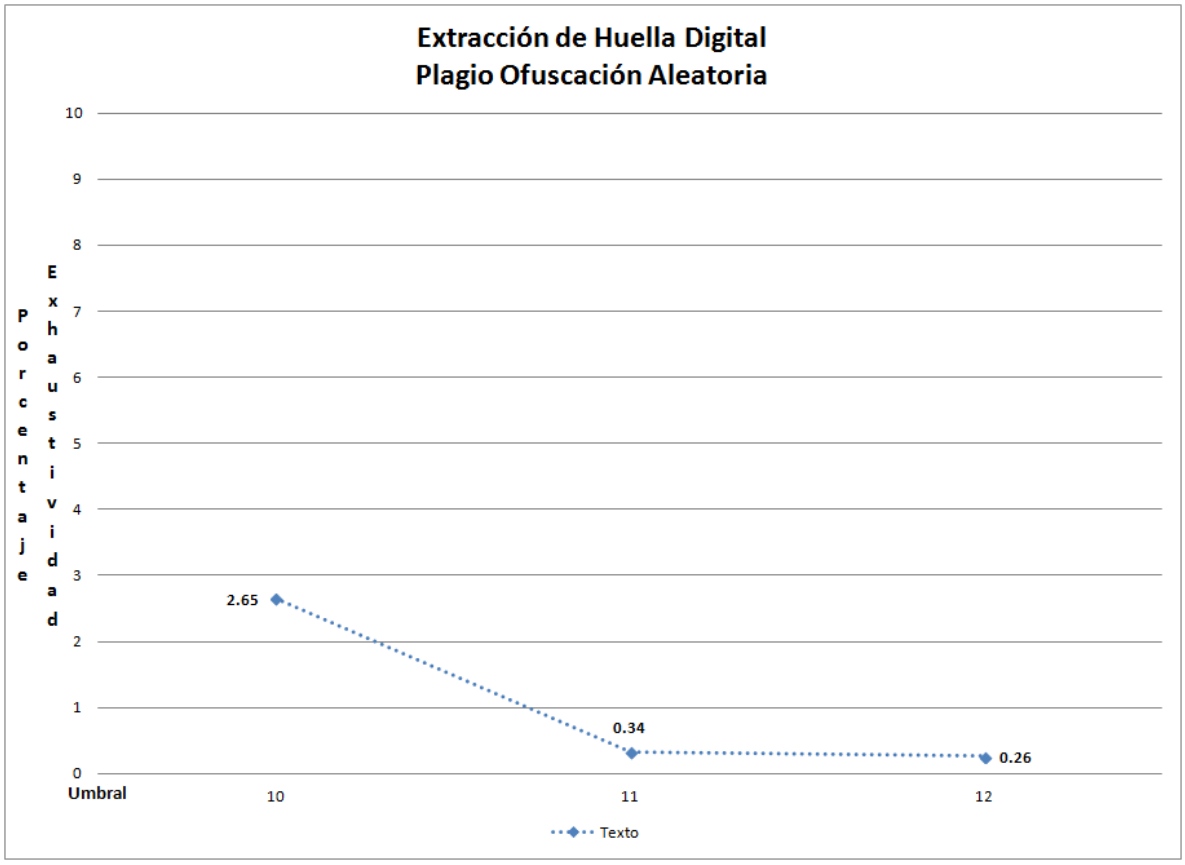


Figura 4.7: Porcentajes de exhaustividad del algoritmo de EHD con plagio disfrazado o de ofuscación aleatoria utilizando un tamaño de minutia de 5.

4.2.3. Plagio de ideas

Este tipo de plagio es muy difícil de detectar, porque que se toman ideas de un documento fuente y se replantean en otras palabras, esto complica que ambos algoritmos produzcan buenos resultados. En el corpus, este tipo de plagio aparece bajo el nombre de ofuscación resumen, y empareja 1185 documentos sospechosos con documentos fuentes en una relación de uno a uno. Los documentos sospechosos contienen fragmentos de texto con ideas plagiadas de su respectiva pareja, que fueron creados por los expertos de PAN.

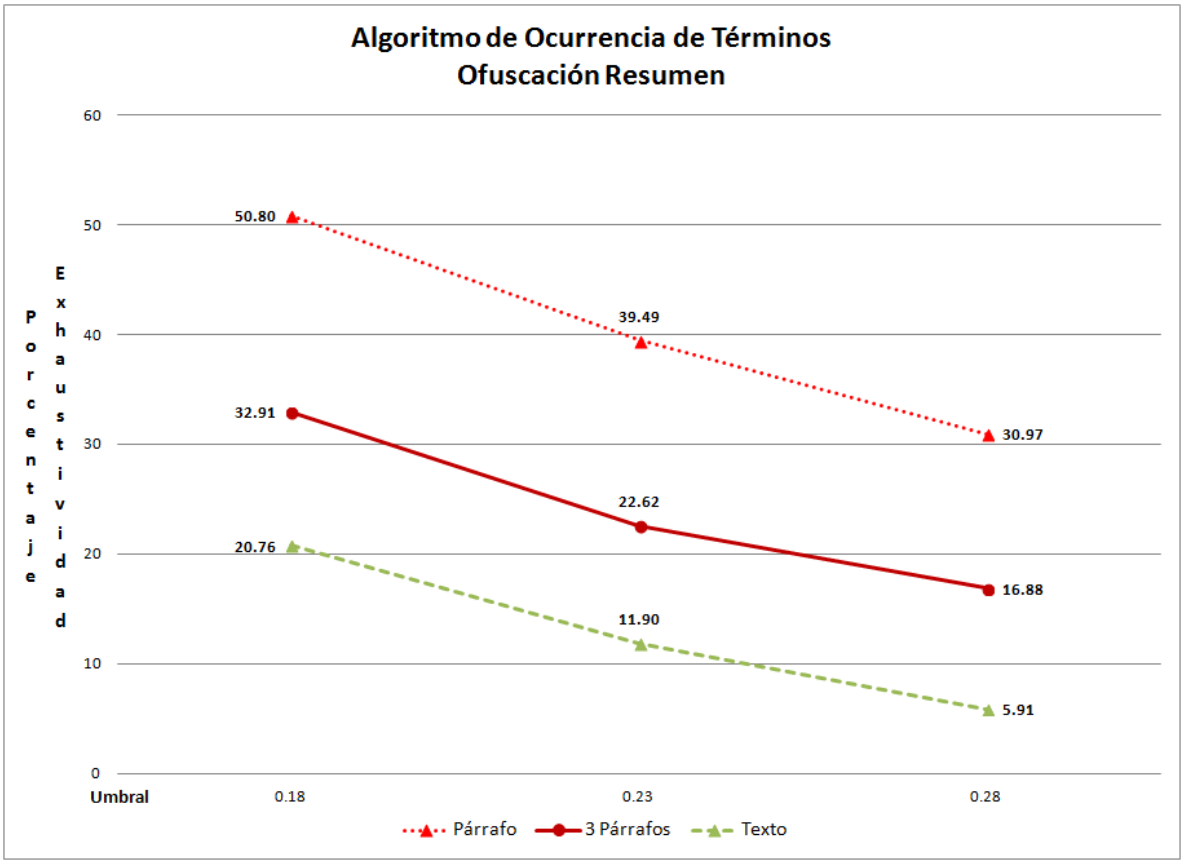


Figura 4.8: Porcentajes de exhaustividad del algoritmo de AOT con plagio de ideas o de resumen.

Algoritmo de análisis de ocurrencia de términos

A pesar de la dificultad de detectar este tipo de plagio, el algoritmo de AOT, con un umbral de 0.18, obtuvo una exhaustividad de más de 50 % utilizando la granularidad de párrafo, como se puede apreciar en la Figura 4.8. Sin embargo, es evidente que las otras granularidades producen resultados pobres y que la exhaustividad decae aún más rápidamente que con el plagio disfrazado y el plagio literal, a medida que se incrementa el umbral de plagio.

Algoritmo de extracción de huella digital

El algoritmo de EHD no logró detectar el plagio de ideas en más que un solo documento. Por la forma en que este tipo de plagio cambia las palabras y el orden del texto original, se imposibilita que el algoritmo de EHD con un algoritmo de *hash* criptográfico arroje buenos resultados. En la Figura 4.9 se aprecia como el algoritmo logró detectar solo una de las parejas, lo que equivale a un 0.17 % de exhaustividad.

4.2.4. Porcentaje de certeza de plagio

A partir de los resultados de las pruebas, usando la lista de plagio literal, se obtuvieron valores de precisión que indican la probabilidad que un documento sospechoso señalado como plagio por los algoritmos realmente contenga plagio. En el caso que el algoritmo detecte que el documento sospechoso obtuvo información de una única fuente, el porcentaje de certeza sería igual a la precisión del algoritmo. Pero en el caso que el algoritmo encuentre más de un documento fuente, este porcentaje se debe calcular utilizando la probabilidad binomial de que al menos uno de los fuentes haya sido plagiado. Esto se debe a que, es suficiente con que el documento sospechoso haya tomado información de un solo documento fuente, para ser considerado plagio. Si n es la cantidad de documentos fuente encontrados por el algoritmo, y p la precisión del algoritmo, el porcentaje de certeza se calcula como

$$\sum_{i=1}^{n} \binom{n}{i} p^i (1-p)^{n-i}.$$

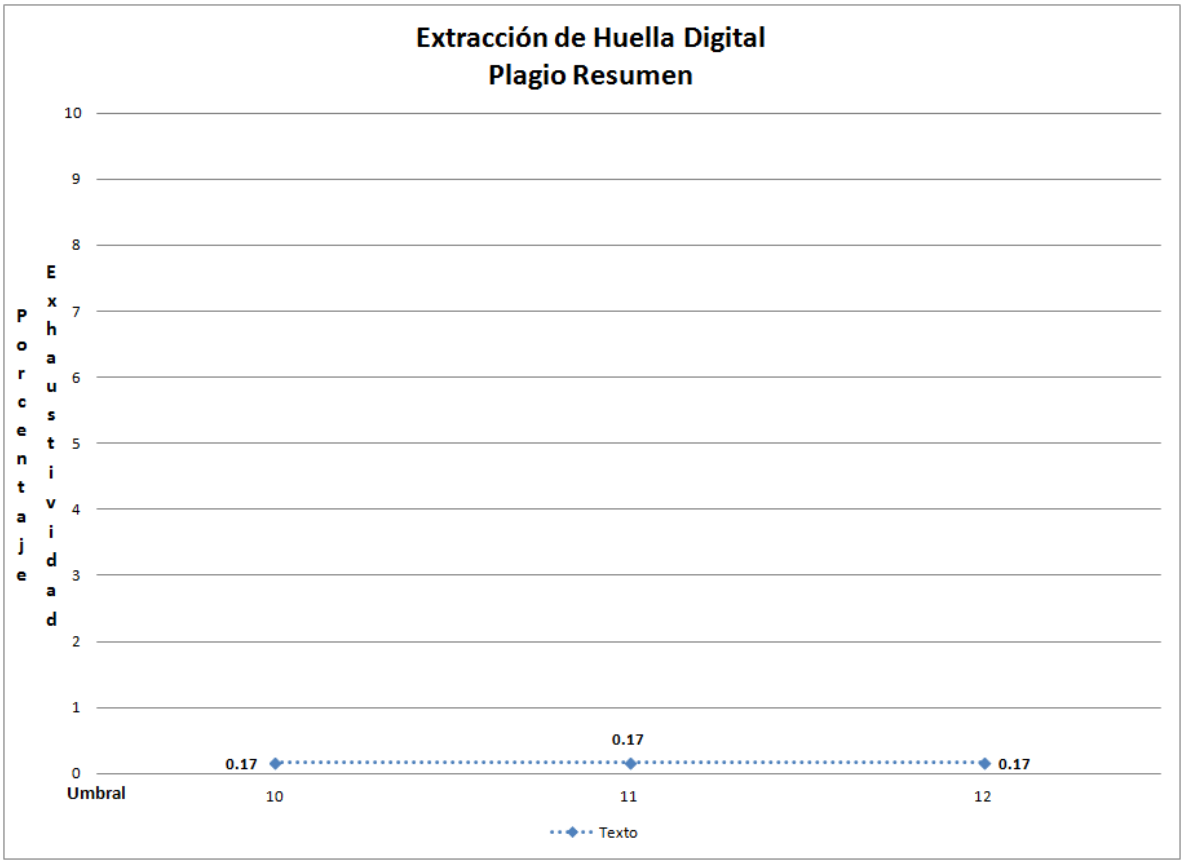


Figura 4.9: Porcentajes de exhaustividad del algoritmo de EHD con plagio de ideas o de ofuscación resumen utilizando un tamaño de minutia de 5.

donde $\binom{n}{i}$ es la combinatoria de n en i. El porcentaje de certeza se calcula asumiendo que se extrajo la información de los documentos fuente sin darles el reconocimiento apropiadamente. En el caso que no sea así, se deben descontar de n los documentos fuentes que hayan sido correctamente citados. Por ejemplo, si un algoritmo, utilizando determinados parámetros, arroja que un documento sospechoso obtuvo información de tres documentos fuente y tiene una precisión de 70 %, el porcentaje de certeza de que exista plagio es $\binom{3}{1}(0{,}7)^1(0{,}3)^2 + \binom{3}{2}(0{,}7)^2(0{,}3)^1 + \binom{3}{3}(0{,}7)^3(0{,}3)^0 = 0{,}973$ o sea 97.3 %. Esto muestra la importancia de la exhaustividad de los algoritmos, porque aunque 70 % no es una precisión alta, la certeza de que un documento sospechoso haya cometido plagio incrementa rápidamente a medida que más documentos fuente son encontrados por los algoritmos.

4.2.5. Eficiencia de los algoritmos

Se dividió la medición de la eficiencia de los algoritmos en dos aspectos, el tiempo que toma comparar dos documentos entre sí y el tiempo que toma crear la base de datos con todos los vectores o huellas de los documentos fuente. Las pruebas se ejecutaron en una computadora de escritorio con sistema operativo Windows 10 Pro, procesador AMD Phenom(TM) II x4 945 de 3 GHz, disco duro Western Digital Blue WD5000AAKS de 7200 RPM y 4 GB de RAM.

Comparación entre documentos

Para tomar las mediciones de tiempo, primero se eligió un documento sospechoso con el cual se compararían todos los documentos fuente del corpus. El documento seleccionado fue el 445, por tener una tamaño cercano a la media de los documentos del corpus. Luego, se obtuvieron los tiempos que toma comparar este documento con cada uno de los documentos fuente y se calculó el promedio. Este promedio se calculó 30 veces y luego se obtuvo un promedio de los promedios, para las 3 granularidades, en ambos algoritmos. Además, se midió EHD utilizando tamaños de minutia de 5, 6 y 7 palabras.

Como se puede observar en la Figura 4.10, el algoritmo de AOT demora menos tiempo que el algoritmo de EHD para comparar los documentos, pero se ve mucho más afectado por el cambio de granularidad. La diferencia de tiempo que toma la comparación entre granularidades usando el algoritmo de EHD es mínima. Ambos algoritmos logran hacer comparaciones más rápidas con granularidad de texto completo y se vuelven más lentos a medida que se subdividen los documentos en bloques de tres párrafos y un párrafo. Esto se debe a que las comparaciones se hacen entre cada uno de los bloques de los documentos, lo que resulta en comparaciones más complejas y lentas. También se puede apreciar que, usando el algoritmo de EHD, a medida que disminuye el tamaño de la minutia, las comparaciones toman más tiempo. Esto se debe a que, al disminuir el tamaño de minutia, la huella requiere de más minutia para representar el mismo documento.

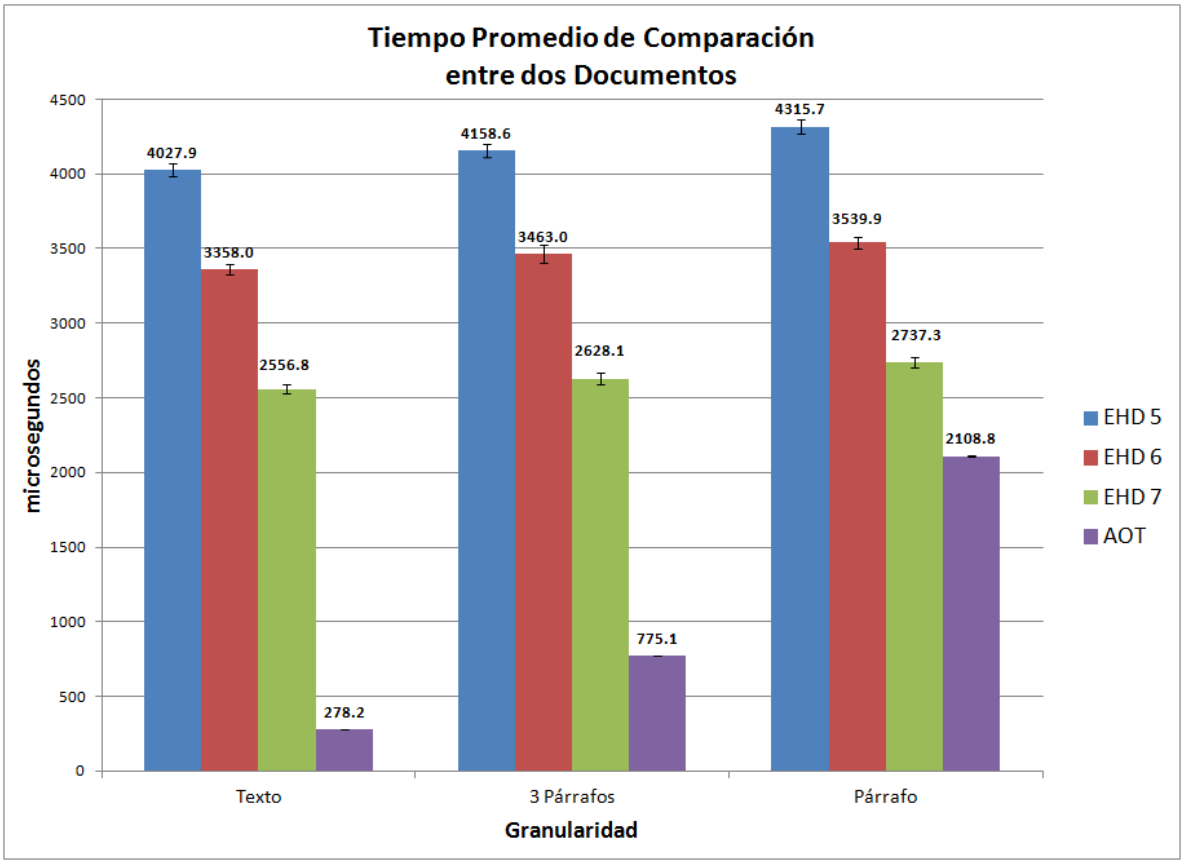


Figura 4.10: Promedio del tiempo que toma comparar el documento sospechoso 445 contra un documento fuente.

Creación de la base de datos

Se midió el tiempo promedio necesario para crear cada vector y huella de los documentos fuente y guardar estas estructuras en la base de datos. Al igual que la mediciones anteriores, se calcularon los promedios para ambos algoritmos usando las tres granularidades, dividiendo EHD en los tamaños de minutia de 5, 6 y 7 palabras.

Como se aprecia en la Figura 4.11, de nuevo, el algoritmo de AOT es más rápido que el de EHD, pero en este caso, ambos se ven afectados de manera similar cuando cambia la granularidad. Además, se nota que, al usar el algoritmo de EHD, incrementar o disminuir el tamaño de minutia no afecta demasiado el tiempo requerido para crear la base de datos, pero sigue siendo más eficiente utilizar tamaños de minutia más pequeños. Sin embargo, ahora es mucho mayor la diferencia de tiempo demorado en-

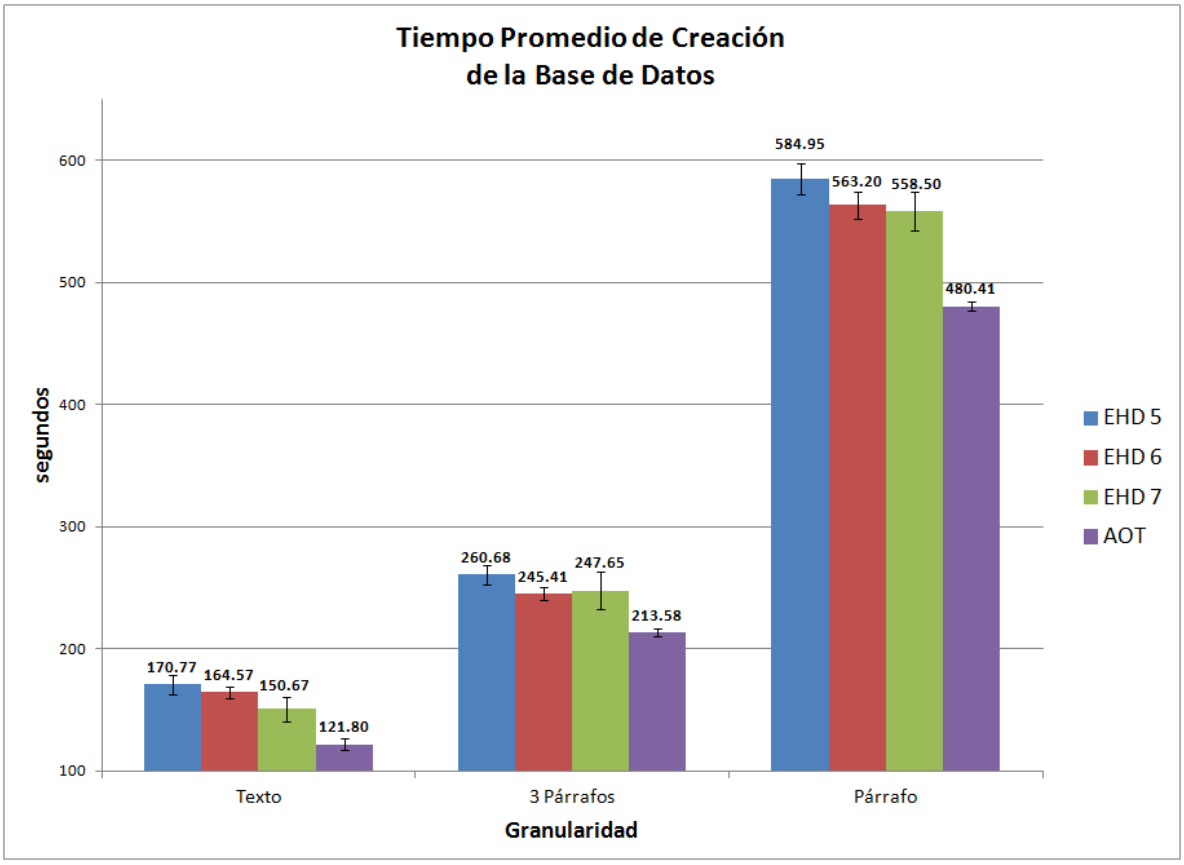


Figura 4.11: Promedio del tiempo que toma crear los vectores o huellas de los documentos fuente y guardarlos en la base de datos.

tre las granularidades, siendo las granularidad de texto completo la más rápida y la granularidad de párrafo la más lenta.

4.2.6. Cuadros comparativos

A continuación se comparan los resultados más altos de exhaustividad, tiempo de comparación y tiempo de creación de la base de datos, según la configuración de parámetros de los algoritmos, y un cuadro para la precisión y para el valor F de los resultados obtenidos de las pruebas de plagio literal. Los valores más altos de cada configuración se aparecen subrayados y el valor máximo total se encuentra en negrita. El valor F es una medida que se obtiene calculando la media armónica de los resultados de precisión y exhaustividad [van Rijsbergen, 1979]. Sea P la precisión y E la exhaustividad de un

algoritmo en un umbral determinado. La medida F es su media armónica, que se puede calcular de la siguiente manera:

$$F = 2\frac{PE}{P+E}$$

Por ejemplo, si el algoritmo de EHD con un tamaño de minutia de 5 palabras, usando granularidad de texto completo, obtuvo una precisión de 83.92 % y una exhaustividad de 83.24 %, en el umbral de 11 minutia compartidas, su valor F es 83.58 %.

En el Cuadro 4.6 se comparan los resultados más altos de precisión de los algoritmos. Se puede observar que los mejores resultados se obtuvieron al usar la granularidad de párrafo. El mejor resultado se obtuvo usando el algoritmo de EHD con un tamaño de minutia alto (7 palabras) y un umbral alto (13 minutia compartidas).

En el Cuadro 4.7 se comparan los resultados más altos de exhaustividad de los algoritmos. Los mejores resultados se obtuvieron con granularidad de texto completo, cuando se usa el algoritmo de EHD, pero en el caso del algoritmo de AOT la granularidad de párrafo arrojó el mejor resultado total.

En el Cuadro 4.8 se comparan los valores F más altos arrojados por los algoritmos. Este valor indica que estas configuraciones dan resultados de exhaustividad y precisión más equitativos. Los valores más altos se obtuvieron al usar granularidad de texto completo con el algoritmo de EHD, pero con el algoritmo de AOT, vuelve a arrojar mejores resultados la granularidad de párrafo. El valor máximo se da al usar el algoritmo de EHD con un tamaño de minutia bajo (5 palabras) y un umbral bajo (11 minutia compartidas). También se puede observar que los valores del algoritmo de EHD son más altos que los arrojados por AOT. Esto refleja el comportamiento que se observó en las figuras 4.4 y 4.6, en los cuales el algoritmo de EHD arrojó resultados de precisión y exhaustividad más cercanos, mientras que el algoritmo de AOT arrojó resultados más

Cuadro 4.6: Cuadro comparativo de los resultados de precisión al detectar plagio literal con la lista de parejas modificada.

Algoritmo/Granularidad	Texto	3 Párrafos	Párrafo
AOT Umbral 0.38	81.65	85.04	86.57
EHD Tamaño 5 Umbral 15	85.70	86.26	88.20
EHD Tamaño 6 Umbral 14	86.35	86.97	88.71
EHD Tamaño 7 Umbral 13	87.83	88.33	**89.83**

Cuadro 4.7: Cuadro comparativo de los resultados de exhaustividad al detectar plagio literal con la lista de parejas modificada.

Algoritmo/Granularidad	Texto	3 Párrafos	Párrafo
AOT Umbral 0.18	84.21	87.19	**90.89**
EHD Tamaño 5 Umbral 10	83.86	83.28	79.99
EHD Tamaño 6 Umbral 9	82.13	81.25	78.31
EHD Tamaño 7 Umbral 8	80.72	79.92	76.70

Cuadro 4.8: Cuadro comparativo de los valores F al detectar plagio literal con la lista de parejas modificada.

Algoritmo/Granularidad	Texto	3 Párrafos	Párrafo
AOT Umbral 0.28	68.60	70.95	73.48
EHD Tamaño 5 Umbral 11	**83.58**	83.04	81.07
EHD Tamaño 6 Umbral 9	82.83	82.47	81.31
EHD Tamaño 7 Umbral 8	82.41	82.13	80.84

polares.

En el cuadro 4.9 se compara el tiempo promedio que le toma a los algoritmos hacer una comparación entre dos documentos. Al usar la granularidad de texto se obtienen los tiempos de comparación más rápidos, siendo el algoritmo de AOT el más eficiente.

En el cuadro 4.10 se compara el tiempo promedio que tarda la creación e introducción, a la base de datos, de las estructuras de datos de los algoritmos. Nuevamente, al usarse la granularidad de texto completo, se obtienen los tiempos más cortos, también resultando, el algoritmo de AOT, el más eficiente.

Cuadro 4.9: Cuadro comparativo del tiempo promedio que tarda la comparación entre dos documentos en micro segundos.

Algoritmo/Granularidad	Texto	3 Párrafos	Párrafo
AOT	**278**	775	2109
EHD Tamaño 5	4027	4158	4315
EHD Tamaño 6	3358	3463	3540
EHD Tamaño 7	2557	2628	2737

Cuadro 4.10: Cuadro comparativo del tiempo promedio que tarda la creación de los vectores y huellas de los documentos y su introducción a la base de datos en segundos.

Algoritmo/Granularidad	Texto	3 Párrafos	Párrafo
AOT	**121.80**	213.58	480.41
EHD Tamaño 5	170.77	262.90	584.95
EHD Tamaño 6	164.57	245.41	563.20
EHD Tamaño 7	150.67	247.65	558.50

Capítulo 5

Conclusiones

A manera de conclusión, en las siguientes tres secciones damos recomendaciones sobre el uso de los algoritmos y sus parámetros, según las necesidades del usuario. Además, especificamos el cumplimiento de los objetivos y proponemos mejoras a considerar en trabajos futuros.

5.1. Recomendaciones

Si el usuario busca que los algoritmos arrojen resultados precisos, se recomienda usar el algoritmo de EHD con un umbral de plagio alto, por ejemplo 13 minutia compartidas, y un tamaño de minutia alto, por ejemplo 7 palabras, usando granularidad de párrafo. Los valores de estos parámetros arrojan los mejores resultados para la detección de plagio literal. Sin embargo, como este algoritmo no logra detectar plagio disfrazado y plagio de ideas eficazmente, se recomienda usarlo junto con el algoritmo de AOT. El algoritmo de AOT debe usar un umbral de plagio alto, alrededor de 0.38, y granularidad de párrafo.

De otra forma, si el usuario prefiere alcanzar exhaustividad por encima de precisión, se recomienda usar el algoritmo de AOT con un umbral de plagio bajo, alrededor de 0.18, en granularidad de párrafo. Con estos parámetros el algoritmo arroja los mejores resultados de exhaustividad para los tres tipos de plagio.

Por último, si el usuario desea un balance entre los resultados de precisión y exhaustividad, se recomienda utilizar el algoritmo de EHD con un umbral bajo, cercano a 11 minutia compartidas, y un tamaño de minutia bajo, por ejemplo de 5 palabras, usando

granularidad de texto completo. Se recomienda usarlo junto con el algoritmo de AOT con granularidad de párrafo y con un umbral medio, cercano a 0.28. Sin embargo, hay que recordar que al usar el algoritmo de EHD con esta granularidad, este se vuelve susceptible a arrojar más falsos positivos a medida que el tamaño del documento aumenta. Esto debido a que al realizar la comparación entre documentos, entre más extensos sean los textos, es más probable que compartan una cantidad mayor de minutia. Por ejemplo, en las pruebas realizadas se obtuvo que el umbral de 11 minutia arroja los mejores resultados de exhaustividad, pero esto valor podría ser fácilmente superado al comparar documentos muy extensos, aunque no tengan una relación de plagio entre sí. Para mitigar este efecto, se debe incrementar el umbral de plagio, sin embargo, no hay valores definidos para este, que puedan garantizar la detección precisa de plagio.

5.2. Cumplimiento de objetivos

En este proyecto el primer objetivo fue alcanzado en su totalidad, mientras que el segundo, se completó parcialmente. Para el primer objetivo, se logró programar ambos algoritmos de forma que logran detectar los diferentes tipos de plagio, en las tres granularidades. Para el segundo objetivo, se logró realizar el estudio comparativo de la precisión y exhaustividad de los algoritmos, utilizando diferentes configuraciones de parámetros, contra el plagio literal. Sin embargo, no se logró obtener la precisión para el plagio de ideas y el plagio disfrazado, esto debido a que las listas de parejas del corpus no incluyen todos los verdaderos positivos, sin los cuales no se puede medir la precisión real de los resultados.

5.3. Trabajo a futuro

Uno de los aspectos que no se tomaron en cuenta, al comparar los documentos, es la distancia que existe entre las minutia (en el caso de EHD) y entre las palabras (en el caso de AOT) en un mismo bloque. Esto puede ser un factor a considerar en próximas investigaciones para mejorar los resultados de precisión y exhaustividad de ambos algoritmos. Otro aspecto que se puede considerar, es la forma en que se codifican las minutia, para el algoritmo de EHD. Es posible que, si se se codificaran las subcadenas de texto con un algoritmo de *hash* perceptual, se obtengan mejores resultados en la

detección de plagio de ideas y plagio disfrazado.

A P É N D I C E S

Apéndice A

Información adicional

A.1. División del trabajo

La división de trabajo se realizó de la siguiente forma. Joshua Briceño se encargó del diseño, implementación y elaboración de pruebas para el algoritmo de EHD. Andrea Solano se encargó del diseño, implementación y elaboración de pruebas para el algoritmo de AOT. Ambos estudiantes colaboraron para realizar el estudio comparativo de los algoritmos, así como el análisis de los resultados y la elaboración del documento final.

A.2. Figuras adicionales

A continuación aparecen los gráficos de los resultados arrojados por el algoritmo de EHD, al usar los tamaño de minutia de 6 y 7 palabras.

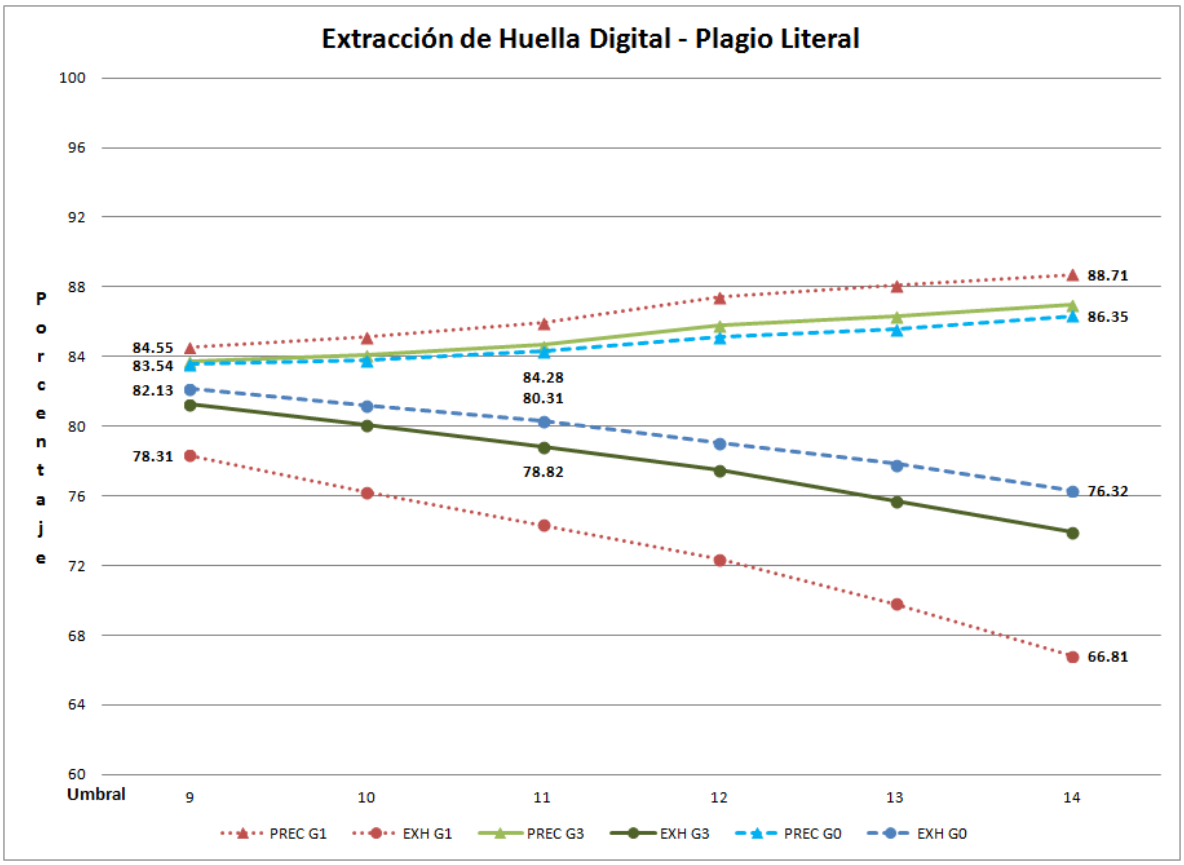


Figura A.1: Resultados de EHD con tamaño de minutia de 6, utilizando la lista de parejas modificada.

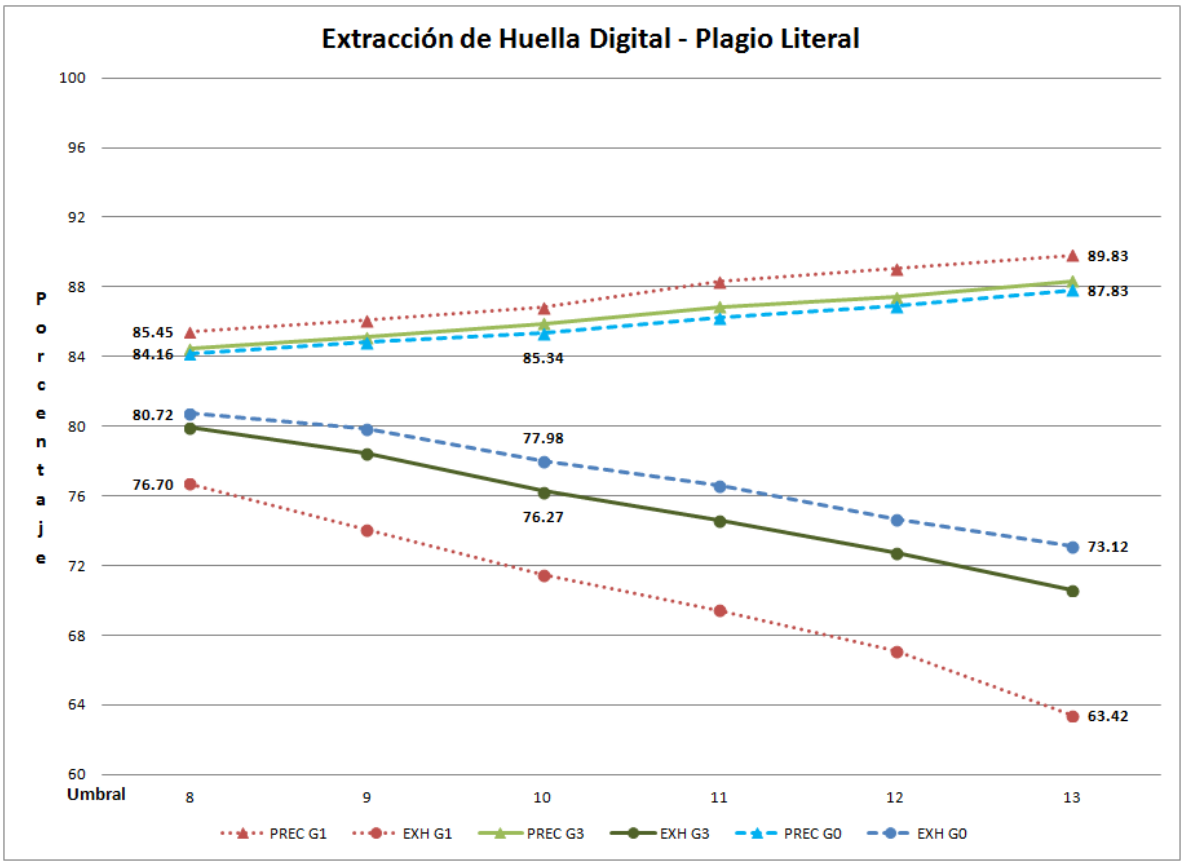


Figura A.2: Resultados de EHD con tamaño de minutia de 7, utilizando la lista de parejas modificada.

Bibliografía

[RAE, 2014] (2014).

[Born, 2003] Born, A. D. (2003). How to reduce plagiarism. *Journal of Information Systems Education*, 14(3):223–224.

[Clough, 2000] Clough, P. (2000). *Plagiarism in natural and programming languages: an overview of current tools and technologies*. PhD thesis, University of Sheffield.

[Ercegovac and Richardson, 2004] Ercegovac, Z. and Richardson, J. J. (2004). Academic dishonesty, plagiarism included, in the digital age: A literature review. *College and Research Libraries*.

[Feistel, 1973] Feistel, H. (1973). Cryptography and computer privacy. *Scientific American*, 228(5):15–23.

[Gipp, 2014] Gipp, B. (2014). *Citation-based Plagiarism Detection - Detecting Disguised and Cross-language Plagiarism using Citation Pattern Analysis*. Springer Vieweg Research.

[Heintze, 1996] Heintze, N. (1996). Scalable document fingerprinting (extended abstract). *USENIX Workshop on Electronic Commerce*.

[Hoad and Zobel, 2003] Hoad, T. and Zobel, J. (2003). Methods for identifying versioned and plagiarised documents. *Journal of the American Society for Information Science and Technology*.

[Juola, 2006] Juola, P. (2006). Authorship attribution. *Foundations and Trends in Information Retrieval*, 1(3):233–234.

[Lancaster, 2003] Lancaster, T. (2003). *Effective and Efficient Plagiarism Detection*. PhD thesis, South Bank University.

[Manber, 1993] Manber, U. (1993). Finding similar files in a large file system. *Proceedings of the USENIX Winter Technical Conference*.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schutze, H. (2008). Scoring, term weighting, and the vector space model. In *Introduction to Information Retrieval*, pages 100–123. Cambridge University Press. Cambridge Books Online.

[Maurer et al., 2006] Maurer, H., Kappe, F., and Zaka, B. (2006). A survey. *Journal of Universal Computer Science*.

[Meuschke and Gipp, 2013] Meuschke, N. and Gipp, B. (2013). State of the art in detecting academic plagiarism. *International Journal for Educational Integrity*, 9(1):50–71.

[Potthast et al., 2014] Potthast, M., Hagen, M., Beyer, A., Busse, M., Tippmann, M., Rosso, P., and Stein, B. (2014). Overview of the 6th international competition on plagiarism detection. *CLEF 2014 Evaluation Labs and Workshop Working Notes Papers*.

[Rajaraman and Ullman, 2011] Rajaraman, A. and Ullman, J. D. (2011). Data mining. In *Mining of Massive Datasets*, pages 1–17. Cambridge University Press. Cambridge Books Online.

[Salton and McGill, 1986] Salton, G. and McGill, M. J. (1986). *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

[Salton et al., 1975] Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620.

[Shivakumar and Garcia-Molina, 1999] Shivakumar, N. and Garcia-Molina, H. (1999). Finding near-replicas of documents on the web. In Atzeni, P., Mendelzon, A., and Mecca, G., editors, *The World Wide Web and Databases*, volume 1590 of *Lecture Notes in Computer Science*, pages 204–212. Springer Berlin Heidelberg.

[Small, 1973] Small, H. (1973). Co-citation in the scientific literature: A new measureof the relationship between two documents. *Journal of the American Society for Information Science*, 4(24):265–269.

[Stamatatos et al., 2015] Stamatatos, E., Potthast, M., Rangel, F., Rosso, P., and Stein, B. (2015). Overview of the PAN/CLEF 2015 evaluation lab. In *Lecture Notes in Computer Science*, pages 518–538. Springer Nature.

[van Rijsbergen, 1979] van Rijsbergen, K. (1979). *Information Retrieval*. Butterworth, 2 edition.

[Weber-Wulff, 2010] Weber-Wulff, D. (2010). Test cases for plagiarism detection software. *Proceedings of the 4th International Plagiarism Conference*.

[Wong et al., 1985] Wong, S. K. M., Ziarko, W., and Wong, P. C. N. (1985). Generalized vector spaces model in information retrieval. In *Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '85, pages 18–25, New York, NY, USA. ACM.

[Zauner, 2010] Zauner, C. (2010). Implementation and benchmarking of perceptual image hash functions. Master's thesis, Hagenberg.