Universidad Politécnica de Valencia

Departamento de Sistemas Informáticos y Computación

Máster en Ingeniería de Software, Métodos Formales y Sistemas de Información.

Tesis de Máster

Herramientas visuales para la búsqueda, la recuperación de información y la búsqueda de contenido relevante aplicado a páginas web.

Alumno: Sergio López Romero

Director de la tesina: Josep Silva Galiana

Índice

Resumen

La información accesible en internet crece de manera descontrolada y se actualiza más rápido que nunca. Debido al problema de encontrar la información actualizada y deseada aparecieron los motores de búsquedas siendo Google su máximo exponente. Estos buscadores son capaces de encontrar páginas concretas en la inmensidad de la red a partir de un criterio de búsqueda pero tras encontrarlo, no son capaces de mostrar de manera precisa al usuario dónde se encuentra lo que buscaba dentro de la página, obligando al usuario a buscarlo y desperdiciando así parte de su tiempo. Dicho problema se acentúa con la complejidad de información y apariencia de las páginas web convirtiendo la búsqueda en una tarea pesada. En esta tesina proponemos una técnica mediante una extensión instalable en Firefox que permite al usuario hacer más cómoda y rápida su búsqueda permitiéndole mostrar la información según la necesidad del momento. Tras obtener la información relevante con el criterio de búsqueda del usuario, observamos que parte de la información contenida en la página web, como menús o el pie de la página web, podría modificar la experiencia del usuario al mostrar más información de la que el usuario necesita realmente. Debido a dicho problema, se propone un algoritmo que se encarga de buscar el bloque de contenido principal relevante de una página web, ignorando u ocultando el resto de la página web irrelevante.

1. Introducción.

La globalización y el abaratamiento de las conexiones han hecho que el proyecto ARPANET que nació como una simple prueba para conectar una comunicación entre dos puntos se convirtiera en lo que es mundialmente conocido como Internet. En ella, cualquier persona del mundo puede conectarse a internet y puede buscar información y comunicarse con otras personas, e incluso introducir más información al alcance de otros usuarios. La facilidad con la que las personas pueden crear más información accesible a otros usuarios del mundo creó un problema serio en la red, ya que toda la información era añadida y actualizada sin seguir una indexación u orden.

A partir de dicho problema surgieron los motores de búsquedas que en la actualidad permiten a todos los usuarios de la web poder encontrar de manera más cómoda la información alojada en la web. Motores de búsqueda como Google, Yahoo o Bing permiten realizar desde una búsqueda mediante un simple criterio hasta completas opciones de búsqueda que combinan información positiva (deseada) y negativa (no deseada). Los motores fueron mejorando encontrando mejor y con más calidad la información de las páginas web. Tras realizar una breve búsqueda observamos que los motores son capaces de encontrar las páginas web donde se encuentra el criterio de búsqueda pero pecan en un gran problema y es que, aunque quitan al usuario el problema de buscar esa información en Internet, no ayudan a facilitar información dentro de una página web obligando al usuario a buscar y leer dentro de la misma. Para solucionar dicho problema, proponemos una técnica implementada en una herramienta para Firefox que nos permite parametrizar de forma fácil un criterio de búsqueda sobre una página web y que se muestre en una nueva página web toda la información relativa al criterio. El usuario puede cambiar la visualización de la información para mostrar más o menos información deseada e incluso aumentar los parámetros de búsqueda. Actualmente, Google es el único motor de búsqueda que es capaz de mostrar de manera escueta y simbólica texto con información del criterio de búsqueda pero sin permitir parametrizar la visualización como nuestra herramienta. Por ejemplo, si se realiza una búsqueda sobre google y pinchamos sobre el icono de la lupa en un resultado, observamos como a la derecha aparece una imagen de la página web con un resaltado sobre nuestra búsqueda pero, al navegar hacia la página no veremos dicho resaltado y tampoco podremos realizar una nueva parametrización.

Al contrario que otras herramientas, nuestra propuesta aprovecha toda la potencia del acceso de datos del navegador, siendo compatible con todos los navegadores que soportan html5, y tener una rápida propuesta de interacción con el usuario. Tras realizar muchas pruebas con diversas páginas, observamos que la mayoría de los navegantes querían buscar la información sólo sobre la información principal de la página web excluyendo de ella los menús, publicidad y los encabezados. Para satisfacer a estos usuarios, se creó otra herramienta para poder buscar de manera automática la información principal de un bloque de texto. Dicha herramienta aprovecha la estructura en forma de árbol de la página web para encontrar y mostrar sólo el nodo que contenga la mejor proporción de texto. La herramienta demostró tener un alto rendimiento para encontrar el bloque de contenido principal además de utilizar un algoritmo de bajo coste computacional. La búsqueda del bloque principal de texto no sólo

es útil para el usuario sino para gran cantidad de sistemas como los indexadores de información ya que les permite ahorrarse un costo computacional para distinguir y separar el contenido del resto de contenido. Combinando ambas herramientas podríamos mostrar toda la información relativa a un criterio pero únicamente sobre el bloque de contenido principal excluyendo todo el ruido posible de la página web como los menús o la publicidad, mejorando la eficiencia de nuestra herramienta de filtrado.

2. Recuperación de Información

Internet es una fuente de gran cantidad de información. Los motores de búsqueda han indexado mucha de esta información y son capaces de extraer las páginas relevantes relacionadas a un patrón de búsquedas. Sin embargo, una vez que los motores de búsqueda recuperan un conjunto de páginas web, el usuario tiene que leer todas las páginas web en orden para encontrar la información importante. Esta tarea consume mucho tiempo por parte del usuario porque las páginas web a menudo mezclan información relevante de diferentes temas y también porque normalmente contienen avisos y publicidad que intenta llamar la atención del lector mediante imágenes, videos, sonidos, etc. En este trabajo definimos una técnica para filtrar la información que nos permite de manera automática filtrar el contenido no deseado del contenido de una página web. La técnica se puede realizar de manera online (sin necesidad de preprocesar una página web para su manipulación).

2.1. Introducción

Internet contiene millones de páginas web con información de prácticamente todos los temas. El esfuerzo de la comunidad científica en definir técnicas eficientes de recuperación de información ha producido buenos resultados [1] y los actuales motores de búsqueda indexan la información que puede ser más tarde recuperada mediante una petición con altos resultados de calidad. Sin embargo, una vez que el conjunto de páginas de páginas web se obtiene, el usuario es forzado a leerlo en orden para encontrar la información deseada.

Sorprendentemente, la tarea de filtrar la información de una página web para eliminar la información no deseada no ha sido automatizada. Existen muy pocas herramientas enfocadas al filtrado de la información de una página web, normalmente son muy limitadas o necesitan de un preproceso de la página web para poder realizar el filtrado. La carencia de aplicaciones en tiempo real capaces de filtrar una página web da una idea de la dificultad de la tarea. Esta dificultad se produce por el hecho que las páginas web están codificados puramente en HTML y este lenguaje no está preparado para manipular la información semántica.

Actualmente, existen varios intentos para solucionar esta situación. Por ejemplo, para la web semántica se utilizan lenguajes como RDF u OWL [2,3] para la construcción del conocimiento. Otras aproximaciones intentan introducir etiquetas semánticas en páginas webs, convirtiendo en explícitas las relaciones semánticas entre los elementos existentes. Estas etiquetas son los llamados microformatos [4,5,6], y son utilizados por las páginas web que incluyen lugares, información de contacto, etc. Desafortunadamente, el 99% de las páginas web son y fueron creados sin tener en cuenta estos modelos.

Una aproximación reciente para la recuperación de información se basa en la creación de una extensión de los motores de búsqueda llamados search engines (motores de respuesta). Estas herramientas construyen un índice de páginas web rastreados donde la información es etiquetada con información semántica que permite extraer la información implícita. Cuando una petición se especifica por el usuario, estas herramientas intentan construir la respuesta a partir de la información almacenada de las páginas webs. La herramienta de este tipo más

conocida es Wolfram Alpha que, desafortunadamente, es todavía muy limitada y sólo es capaz de responder un número limitado de preguntas. Estas herramientas son consideradas el futuro de la recuperación de información pero actualmente son poco utilizadas por su escaso alcance y su falta de filtrado de la información, que en la práctica es poco exitoso.

Otros trabajos relacionados con este trabajo son las herramientas de control parental. Estas herramientas determinan si una página web contiene violencia o contenido pornográfico. Si lo contiene, toda la página web es bloqueada. Contrariamente, en nuestra herramienta no se bloquea completamente la información, simplemente filtra la información ocultando la información no deseada. Las más notables herramientas utilizadas son Webguard , Naomi o Anti-Porn [7,8,9].

Existen unas cuantas herramientas dedicadas a filtrar la información de las páginas web eliminando sólo el contenido no deseado pero frecuentemente necesitan de un preprocesado [10] de las páginas, normalmente usando un proxy [11] para la tarea. Este implica que todas las páginas tienen que permitir este preprocesado y pasarlo, haciéndolo ineficaz si el servidor proxy se cae o si se utiliza de manera offline. Por dicha razón, nuestra herramienta es la primera herramienta que puede filtrar las páginas web sin necesidad de una precompilación o mediante una fase anterior de preparseo.

2.2. Motivación

Esta sección presenta un ejemplo real de filtrado de información mediante la técnica propuesta en la tesina. Consideremos que el usuario está navegando por la página oficial de venta de productos de Apple (store.apple.com) pensando en comprar un iphone. Cuando abrimos la página principal de venta, nos fijamos que hay mucha información y muchos menús de navegación no relacionada con la venta de iphones forzando al usuario a leer información innecesaria para encontrar lo que busca.

Ahora, tenga en cuenta que tenemos nuestra herramienta disponible que es capaz de filtrar toda la información no relacionada con el iPhone. Nuestro algoritmo es capaz de filtrar una página web y sólo muestra la información pertinente según un criterio de filtrado dado. Por ejemplo, nuestra herramienta produce una nueva página web filtrada tal y como se muestra en la Figura 1 (centro). Observe que incluso las imágenes y el menú principal horizontal han sido filtrados (sólo se ve un botón). Si el usuario considera que esta información no es suficiente, o se ha filtrado demasiado, puede aumentar la cantidad de información que se muestra. En este caso, la página web se reconstruye automáticamente incluyendo más información. El resultado se muestra en la Figura 1 (Derecha).

Figura 1. Apple Store (izq.), filtrado con tolerancia 0 (centro) y filtrado con tolerancia 2 (der.).

También es posible filtrar las páginas web de manera opuesta. En lugar de filtrar toda la información no relacionada con el criterio de filtrado, se puede filtrar la información relativa al criterio. Una combinación de ambos nos permite especificar consultas complejas con información positiva y negativa (es decir, filtrar con información que se debe mantener o se quiere eliminar).

2.3. Filtrado de páginas web

Las páginas web se encuentran escritas en un lenguaje de marcado llamado HTML. Si quisiésemos realizar una herramienta con un lenguaje compilado potente, como C++, Java o C#, tendríamos que tener en cuenta que dichos lenguajes no están orientados para la manipulación de archivos HTML (por lo que habría que parchear los archivos HTML a un tipo de archivo o estructura distinto) y, más importante, exigirían estar funcionando en un servidor ajeno con todos los problemas que conlleva de fiabilidad, seguridad y dependencia. Por el contrario, Javascript, a pesar de ser un lenguaje interpretado, es un lenguaje claramente orientado a la manipulación de páginas webs y que goza en las últimas versiones de los navegadores de gran potencia de cálculo. Por tanto, utilizaremos Javascript en nuestra herramienta debido a que para la manipulación de archivos HTML o XML, Javascript goza de DOM (Document Object Model) [12] que es una API que proporciona al programador un conjunto estándar de objetos que representan el documento web y instrucciones para acceder y modificar el contenido o la estructura. Las librerías DOM nos muestran una página web como una estructura en forma de árbol siendo cada elemento de la página web un elemento del árbol. Cada elemento del árbol es un nodo con etiqueta asociado a un texto exceptuando los nodos de la raíz del árbol que sólo son o texto sin etiquetas o bien son nodos con etiquetas pero sin texto. Gracias a esta formalización, podemos enfocarnos en el valor de los atributos de los nodos y en su texto, sin necesidad de hacer tratamientos distintos entre nodos con etiqueta distinta. Es decir, es nuestra aplicación consultamos el valor de los atributos pero en un nodo de tipo *img* sus atributos *alt*, *longdesc* y *src* son exclusivos de dicha etiqueta. Si la página web se encuentra bien estructurada, la raíz del árbol será el nodo HTML.

Para la realización de la búsqueda, permitimos que el usuario realice búsquedas específicas complejas que contengan multitud de palabras y metadatos como “” para la búsqueda exacta, y operadores aritméticos como (AND, OR) para producir combinaciones de textos que fuercen un orden particular de palabras o forzar la existencia o inexistencia de una texto específico.

Por ejemplo, estamos en la página oficial de marca y estamos interesados en toda la información relacionada con el Real Madrid debido a que va a jugar justamente un partido de la liga de campeones y me gustaría que toda la información posible sólo sobre dicho partido. El

marca es una página web muy extensa que suele hablar no sólo sobre todos los equipos de fútbol sino también sobre varios deportes o curiosidades por lo que ver toda la información del partido podría conllevar demasiada lectura. Por tanto, si realizamos un filtrado sobre la página web sobre el 'Madrid' observaremos que la información obtenida es demasiada (figura 2); la labor de lectura del usuario es menor pero acaba leyendo más información de la que necesita.

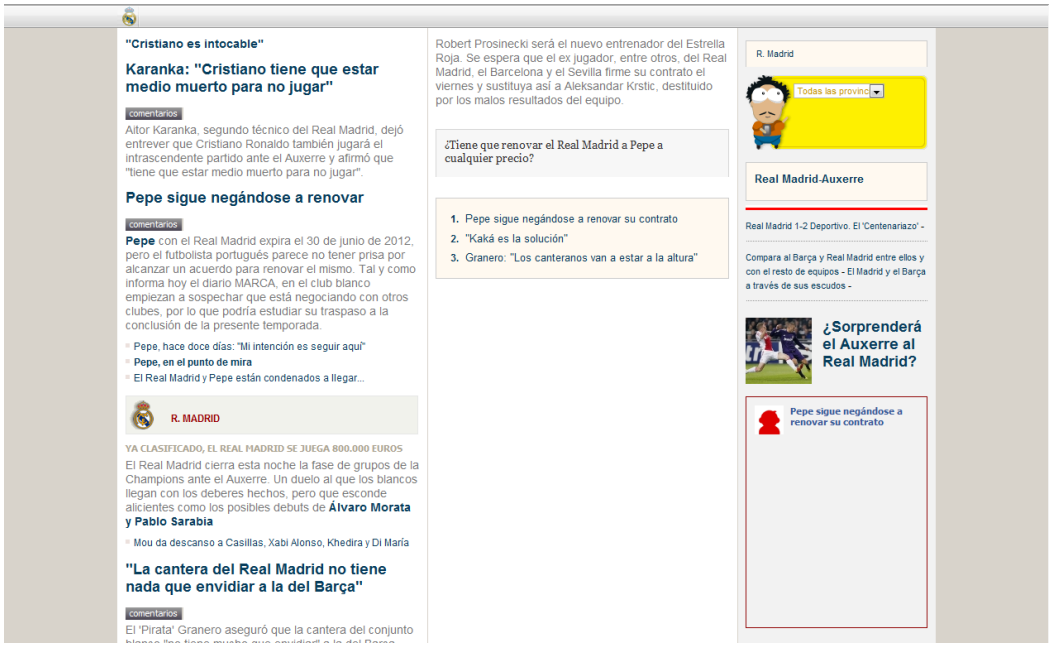


Figura 2. Página oficial de marca con el filtrado "madrid".

Por el contrario, si aprovechamos la potencia de las instrucciones booleanas, podemos afinar la búsqueda mediante el criterio "Madrid and Auxerre" (el auxerre es el equipo que se enfrenta el Madrid) encontrando sólo la información que contengan dicha palabra (figura 3), observando que la información es lo suficiente escueta para ser leído enfocado sólo en el tema que el usuario realmente estaba buscando.

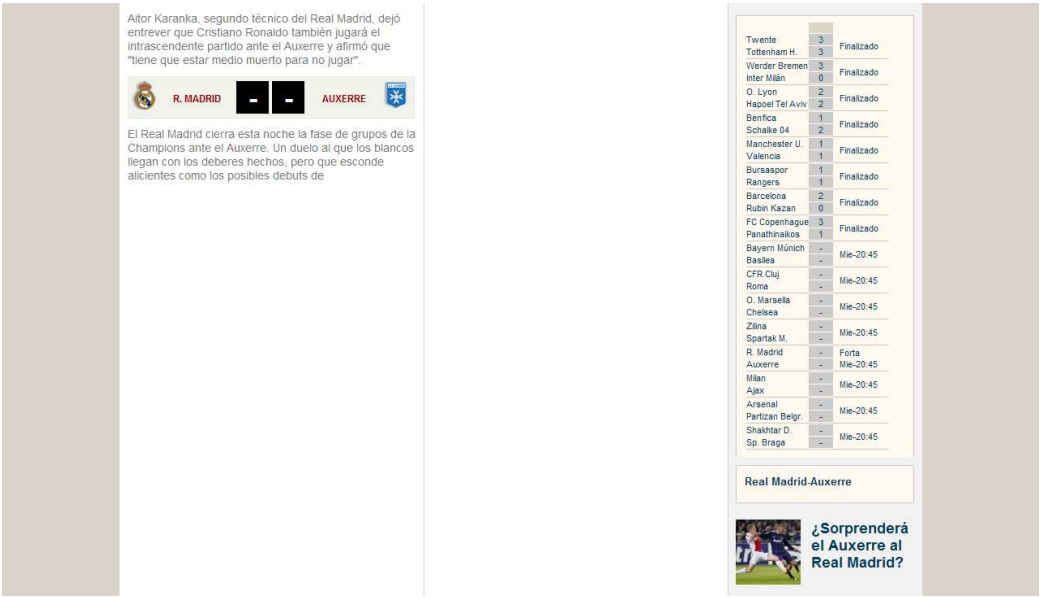


Figura 3. Página oficial de marca con el filtrado "madrid and auxerre".

También podríamos estar interesados en toda la información sobre los equipos valencianos en la liga, así que estaría bien un resumen de toda la información referente a los 3 equipos. Para obtener toda la información sobre los 3 equipos en la página web (figura 4) tenemos que escribir en la búsqueda “valencia or hercules or levante”.

Figura 4. Página oficial de marca con criterio "Valencia or Hercules or Levante".

Además de utilizar las operaciones booleanas, podemos utilizar el criterio inverso para eliminar contenido molesto de una página web. Por ejemplo, estamos visitando, en la página web de facebook, el muro del creador de la aplicación (figura 5) y observamos que a la derecha de la página se encuentra publicidad molesta.

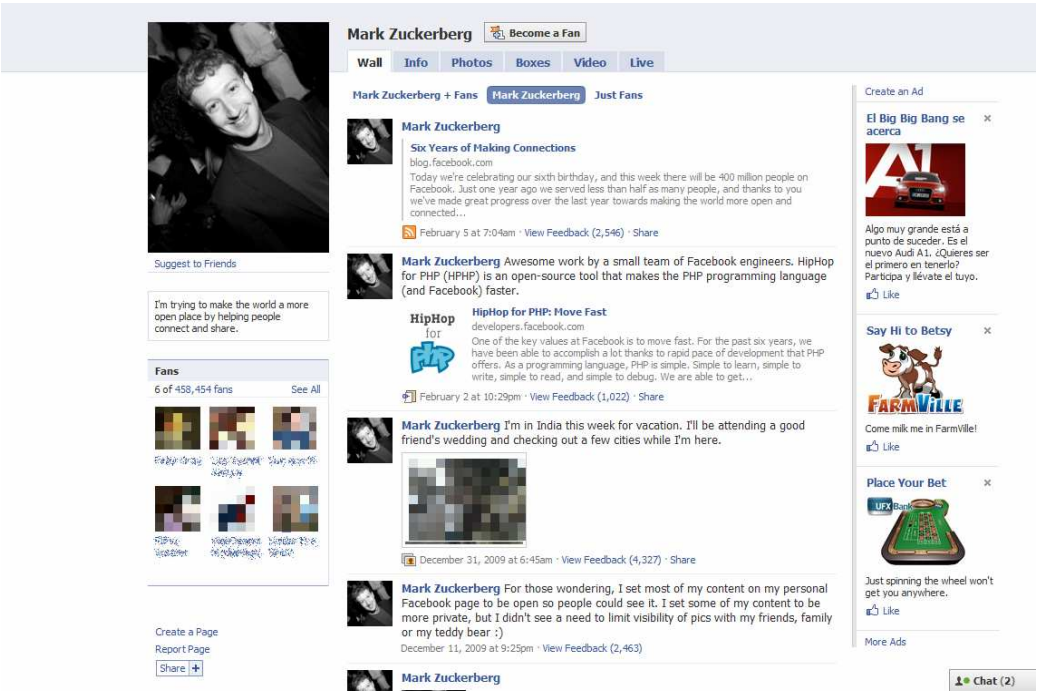


Figura 5. Mark Zuckerberg en Facebook.

Si realizamos un filtrado con el criterio “ads” (término bastante utilizado en Estados Unidos para referirse a la publicidad web debido a la popularidad de la aplicación google adsense) y marcamos la opción filtrado inverso, veríamos como el muro dejaría de tener la información molesta de publicidad (figura 6).

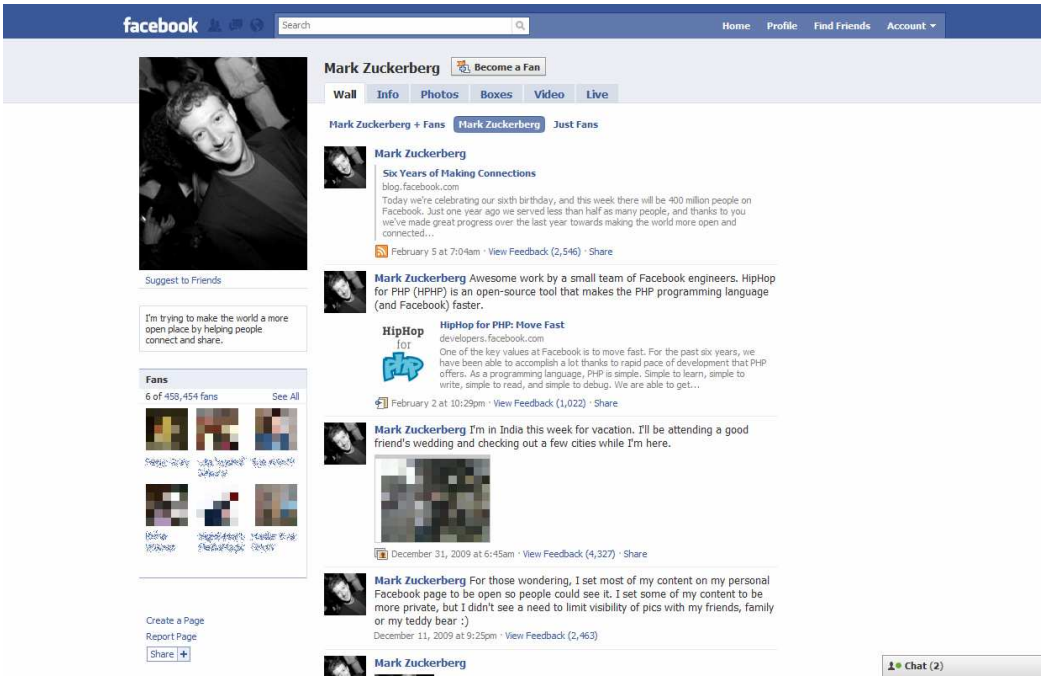


Figura 6. El muro sin la publicidad.

Además de realizar el filtrado mediante un criterio, el usuario puede especificar otros parámetros que configuran la consulta (figura 5). Estos parámetros son los siguientes:

1 **Visuales**. Es una opción booleana ya que sólo deja elegir entre dos opciones. La idea fundamental es cómo filtrar la información que no coincide con el criterio de búsqueda. Si queremos que la información que no coincide con la búsqueda no ocupe espacio, elegimos la opción *colapsar estructura*, que produce en efecto curioso, ya que los nodos relevantes se colapsan acercándose más entre ellos (de manera visual) mientras que si queremos que los nodos relevantes ocupen el espacio relativo a la página original tendremos que elegir *mantener estructura*. La elección de cualquiera de las dos opciones sólo modifica un atributo de cada nodo (atributo *visible* o *display* según la opción) y nunca elimina la relación entre nodos cambiando la estructura en forma de árbol.

2 **Estructurales**. Es una opción booleana que cambia la estructura del árbol DOM para que cuelguen de la raíz del árbol sólo aquellos nodos que sean relevantes con el criterio del usuario. Esta opción modifica la estructura cambiando radicalmente el aspecto visual de la página web. Esta opción invalida cualquier opción visual si se encuentra desactivada (no mantener árbol).

3 **Tolerancia**. Es un valor de tipo numérico relacionado con la cantidad de información que el usuario quiere ver en la página filtrada. La tolerancia es usada para decidir qué elementos del árbol DOM están relacionados con la palabra de búsqueda del usuario. Con una tolerancia 0, sólo los nodos relevantes (y sus descendientes) deberían ser mostrados. Con una tolerancia 1, sólo los nodos relevantes y los nodos con una distancia de 1 respecto a estos nodos tendrían que ser devueltos. El valor máximo de la tolerancia es el mínimo número de desplazamientos desde algún nodo relevante

hasta el nodo body, es decir, si del nodo body cuelga un nodo con etiqueta div y de dicho nodo cuelga un nodo relevente, el número de nodos desde el nodo relevante al nodo body es 2 y, por tanto, la máxima tolerancia es 2.

4 **Filtrado inverso**. Es usado para definir si nuestro filtro tiene que ocultar los nodos que no están relacionados con nuestra búsqueda (*filtrado normal*) o si tiene que ocultar los nodos que coinciden con el texto especificado (*filtrado inverso*).

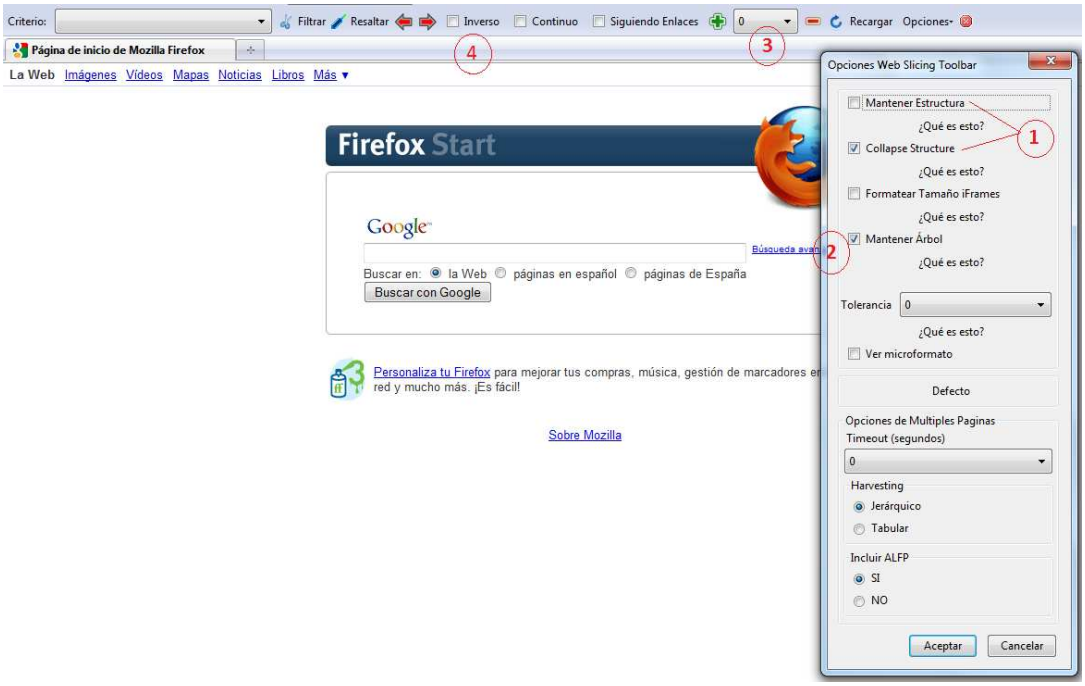


Figura 7. Parámetros de configuración para la consulta.

Por tanto, el filtrado sobre una página web será la combinación tanto del criterio de búsqueda como del conjunto de opciones seleccionadas por el usuario respecto a un nodo (que, por defecto, será el nodo body de la página web actual).

Ahora estamos preparados para presentar el algoritmo de filtrado (algoritmo 1).

Algoritmo 1: Filtrado para páginas web

```
function filtrado_pagina_web(nodo)
{
        switch( coincide_criterio (nodo) )
        {
                case “Irrelevante”:
                        Aplicar_Visuales_y_añadir_nodo_irrelevante(nodo);
                        return “NoEncontrado”;
                case “Relevante_Atributos_nodo”:
                case “Relevante_Union_Atributos_InnerHTML”:
                        añadir_nodo_relevante(nodo);
                        return “Encontrado”;
                case “Relevante_Hijos”:
                        if(filtrado_pagina_web (nodo.hijos) ==”NoEncontrado”)
                                añadir_nodo_relevante(nodo);
                                return “Encontrado”;
```

```
                else    return "NoEncontrado";
        }
}
```

El algoritmo se inicializa pasando como parámetro el nodo con la etiqueta *body*, teniendo en cuenta que realizará una búsqueda de manera recursiva para encontrar los nodos que coinciden con el criterio. Cabe destacar que nuestro algoritmo siempre busca el nodo coincidente con el criterio más bajo de la rama del árbol DOM. El nodo calcula si es un nodo relevante a través de la función *coincide_criterio* que utiliza las palabras introducidas en el criterio y la opción de filtrado inverso para evaluar si un nodo es relevante. La función devuelve cuatro valores:

- Irrelevante. El nodo no coincide con el criterio seleccionado por el usuario y, por tanto, es un nodo irrelevante.

- Relevante_Atributos_nodo: El nodo concide con los criterios seleccionados por el usuario pero sólo coinciden los atributos de dicho nodo.

- Relevante_Union_Atributos_InnerHTML: El criterio coincide con el valor de los atributos y del texto asociado al nodo. Este caso se da cuando se introducen 2 o más palabras en el criterio y, por tanto, una o varias palabras coinciden con los valores de los atributos pero las otras palabras restantes coinciden con el texto asociado a una palabra. Por ejemplo, el usuario busca con el siguiente criterio de búsqueda "*pdf and agenda*":

  ```
  <a href="archivo.pdf">Información de la agenda.</a>
  ```

  Como se puede observar el criterio exige la aparición de las 2 palabras. Vemos que el nodo con etiqueta *a* coincidiría con la palabra *pdf* en el valor de uno de sus atributos pero la palabra *agenda* solo coincide con el valor del texto asociado al nodo.

- Relevante_Hijos: El nodo coincide con el criterio de búsqueda pero sólo en su texto asociado. Nuestro algoritmo busca en profundidad encontrando el nodo más profundo en la rama, por tanto, tiene que ver si alguno de sus hijos coincide con los criterios. Como el texto asociado a un nodo es heredado de la unión del texto de los nodos hijos, puede suceder que el criterio sean 2 o más palabras y que parte del criterio coincide con uno nodo hijo y el resto del criterio coincida en un nodo hijo distinto. Por ejemplo, imaginemos que realizamos un filtrado con el criterio "*agenda and telefonos*" sobre el siguiente código html:

  ```
  <div id="nodoPadre">
          <a id="nodoHijo1" href="Agenda.pdf"> Agenda </a>
          <a id="nodoHijo2" href="Telefonos.pdf">Telefonos </a>
  </div>
  ```

Observamos que el nodo padre con la etiqueta div coincide en su texto con ambos valores de búsqueda pero sus nodos hijos no coinciden plenamente. El primer nodo Hijo sólo coincide con la palabra *Agenda* mientras que el segundo nodo hijo sólo coincide con la palabra *Telefono*. En este caso, el nodo padre sería el relevante.

Las opciones visuales de filtrado como la agregación al vector de nodos irrelevantes se aplica en la función Aplicar_Visuales_y_añadir_nodo_irrelevante (algoritmo 2):

Algoritmo 2: Opciones visuales de filtrado

```
function Aplicar_Visuales_y_añadir_nodo_irrelevante (nodo)
{
        if ( colapsarEstructura)
                nodo.style.display=”none”;
        else
                nodo.style.visibility=”hidden”;

        NodosIrrelevantes.add(nodo);
}
```

Cabe recordar que la opción *colapsarEstructura* significa que los nodos que no son relevantes no tienen que ocupar espacio de la página web mientras que la opción MantenerEstructura significa que los nodos irrelevantes si tienen que mantener su espacio.

Las funciones estructurales de filtrado se realizan después de realizar el filtrado. Si la opción mantener árbol no está activo, entonces se llamará a la función NoMantenerArbol. La función tendrá que eliminar todas las relaciones del nodo con la etiqueta body con sus respectivos hijos e, inmediatamente después, agregar nuevas relaciones con todos los nodos relevantes:

Algoritmo 3: Opciones estructurales de filtrado

```
function NoMantenerArbol()
{
        for (nodoHijos in NodoBody)
                nodoBody.eliminarRelacion(nodoHijos);

        for(nodoRelevante in NodosRelevantes)
                nodoBody.añadirRelacion ( nodoCompleto (nodo) );
}
```

Cabe tener en cuenta una cosa muy importante, antes de añadir la relación del nodo Body con todos los nodos relevantes, se llama al método nodoCompleto que nos devolverá un nodo del árbol que sea un nodo completo. La razón se debe a que existen nodos que necesitan de la existencia de un nodo padre con una etiqueta en concreto para ser nodos completos. A dichos nodos los llamamos nodos huérfanos. Por ejemplo, los nodos de tipo lista (nodos con etiqueta *OL* o *UL*) siempre tienen hijos de elemento de una lista (nodos con etiqueta *LI*). Un nodo con etiqueta *LI* necesita siempre tener como padre a un nodo con etiqueta *OL* o UL sino tanto su información estructural como visual no será completo. Las nodos huérfanos son aquellos que

son del tipo elemento de una lista (li) o del tipo elementos de una table (tr,td). Por tanto, el método se encargará de convertir los nodos que sean huérfanos en un nodo completo con la mínima información posible. Es decir, si realizásemos un filtrado sobre el siguiente código HTML con el criterio *Agenda* observaríamos que sólo un nodo coincide con el criterio y que es el elemento de la lista marcado con un atributo nodoRelevante.

```
<ul type=”circle” sytle=”aspecto_lista”>
    <li>Telefonos</li>
    <li id=”nodoRelevante”>Agenda</li>
    <li>Citas</li>
</ul>
```

Si se le pasa al método nodoCompleto dicho elemento de la lista, este método devolverá el siguiente nodo:

```
<ul type=”circle” sytle=”aspecto_lista”>
    <li id=”nodoRelevante”>Agenda</li>
</ul>
```

Por último, el último parámetro es la tolerancia que se aplica justo al final de todos los métodos. Si la tolerancia es distinta de 0, subiremos tantos niveles en el árbol desde todos los nodos relevantes como valor tenga la tolerancia seleccionada y, posteriormente, de manera recursiva se mostrarán todos los nodos que cuelguen de dicho nodo. La función para mostrar variará según la opción visual, si la opción elegida es colapsar estructura se realizará la instrucción nodo.style.display=””. Si la opción elegida es mantener estructura entonces se ejecutará la instrucción nodo.style.visibility=”visible”.

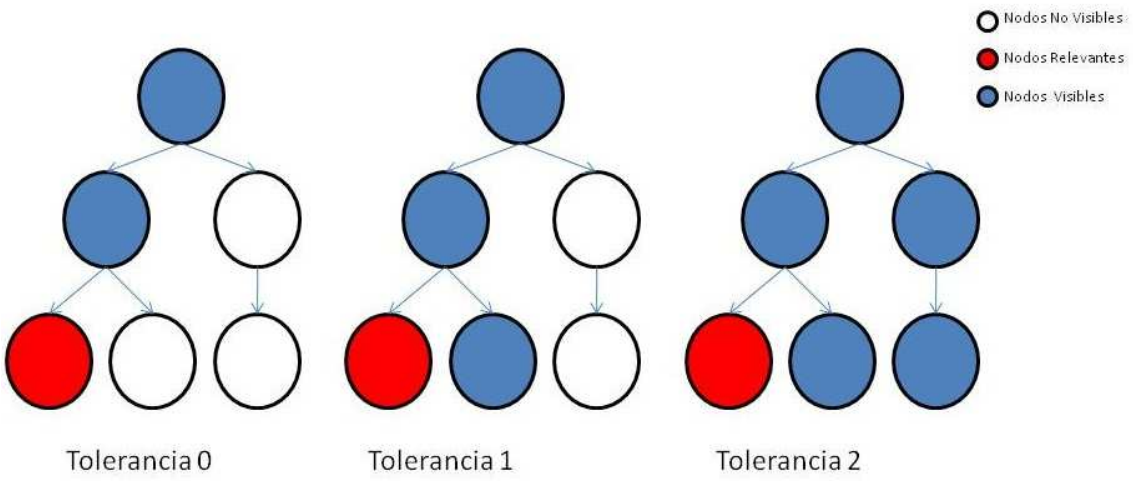


Figura 8. Tolerancia aplicada desde un nodo relevante.

Es posible realizar de nuevo el filtrado sobre una página ya filtrada. Al contrario que el primer filtrado, el siguiente y continuos filtrados se realizan sólo y únicamente sobre los nodos relevantes y nunca de nuevo sobre el nodo raíz. Tras realizar el filtrado, todos los nodos relevantes que no coincidan con el nuevo criterio dejarán de ser nodos relevantes. A partir de este momento se pueden cambiar todas las opciones menos las visuales y estructurales que se mantiene respecto al primer filtrado.

Además de todas las opciones ya comentadas, el usuario puede seleccionar la opción "*continuo*" de la barra de herramientas que permite ejecutar un filtrado automático cada vez que se carga una nueva página web. Esta opción es muy interesante para cuando se utiliza la herramienta como un control parental y queremos que nos oculte la información considerada no apta para niños. Para realizar dicho ejemplo introduciríamos en el criterio las palabras consideradas no aptas para niños (palabras malsonantes o de contenido adulto) separadas por la instrucción OR.

2.4. Extensión del algoritmo a formatos diferentes

Tras realizar pruebas durante meses (el plugin tiene más de dos años de existencia y ahora mismo se encuentra por la versión 1.5), podemos asegurar que el algoritmo funciona de manera eficiente con un coste computacional bajo (no es perceptible el tiempo que tarda en el realizar un filtrado) gracias al uso de las librerías DOM utilizando JavaScript. El problema que observamos mientras testeamos la aplicación fue al comprobar que la mayoría de las páginas web tenían archivos en formatos distintos al html (como el pdf, xml o doc) que son utilizados para almacenar grandes cantidades de información. Como las librerías DOM sólo funcionan sobre archivos de estructuras con formato XML, el algoritmo propuesto no podía funcionar en estos tipos de archivos. Como el algoritmo funciona perfectamente sobre HTML se pensó que en vez de adaptar el algoritmo a cada tipo de formato, sería mejor adaptar cada tipo de archivo a HTML para utilizar el mismo algoritmo. Para la conversión en html utilizamos los siguientes tipos de formatos:

- XML: Aunque es un tipo de formato que es compatible con la API DOM, las librerías sólo permiten consultar y modificar los nodos pero no se le puede aplicar transformaciones visuales como en html para poder utilizar opciones visuales o estructurales en el filtrado. La metodología a seguir fue dividir cada nodo como un atributo de tipo div teniendo como xmlId el nombre que tiene el nodo y sus atributos se añaden dentro del xmlStyle. Por ejemplo:

```
<xml version="1.0" encoding="UTF-8" >
<Mensaje title="Tesina">
    <Remitente Nombre="Sergio López" />
    <Destinatario Nombre="Josep Silva" />
    <Contenido> Mensaje Contenido del mensaje tesina </Contenido>
</Mensaje>
```

```
</xml>
```

Se convertiría en el siguiente tipo de nodo *div*:

```
<div xmlId=”Mensaje“ xmlStyle=”title:Tesina;”>
    <div xmlId=”Remitente” xmlStyle=”Nombre:Sergio López;” />
    <div xmlId=”Destinatario” xmlStyle=”Nombre:Josep Silva;” />
    <div xmlId=”Contenido”> Mensaje Contenido del mensaje tesina </div>
</div>
```

La conversión de archivos XML en HTML no conlleva apenas dificultad más allá de las pautas elegidas para la conversión.

- PDF [13]: Es un tipo de formato convertido en estándar muy utilizado en la web, utilizado para almacenar grandes cantidades de texto con la peculiaridad de que su visualización es independiente de la plataforma donde se ve, es decir, el aspecto visual de un archivo es el mismo en todos los ordenadores (siendo independiente del sistema operativo o resolución de la pantalla).

  La información se encuentra estructurada de la siguiente manera: primero un encabezado que muestra información básica sobre la versión del pdf, posteriormente conlleva distintos objetos con una estructura básica parecida a un *xml*, después un listado con las referencias a todos los objetos y finalmente el tráiler que es la información básica para saber cuál es el objeto de los metadatos y cuál es el objeto *root*. La mayoría de la información es legible exceptuando los objetos de tipo *stream* que contienen, por regla general, la información comprimida en un algoritmo de compresión, como *AsciiDecode* o *HexDecode*. El algoritmo más usado es el *flateDecode*, variación del algoritmo *lwz*, que se encuentra implementado en nuestra herramienta. Los objetos de tipo *stream* contienen la información de las imágenes y de los textos.

  En la conversión en html, nos centramos en los objetos *stream* y descomprimimos la información de estos objetos, para introducirla dentro de un nodo con la etiqueta *div*.

Cabe recalcar que, al programar el algoritmo en una extensión para Firefox, como no se tiene el suficiente nivel de privilegios para la manipulación, se utiliza las librerías AJAX [14] para que, cuando vamos a movernos hacia una página con terminación pdf o xml entonces paramos el desplazamiento y, desde la página actual, eliminamos todos los nodos que cuelgan del nodo body y realizamos la conversión html.

2.5. Resultados obtenidos.

La herramienta nos permite de manera fácil filtrar de manera rápida con la información deseada permitiéndonos seleccionar de un gran texto sólo la información deseada. Si la información mostrada al usuario es demasiado escueta, el usuario puede modificar la tolerancia para mostrar más información acorde al gusto del usuario. Si la información mostrada fuese todavía mínima, el usuario todavía podría realizar más filtrados con gran facilidad para mostrarla hasta el nivel deseado por el usuario.

Los documentos web no sólo contienen información textual sino también otro tipo de información (información estructural de la página web, publicidad, enlaces a otras páginas) por lo que el filtrado no sólo se realiza sobre el contenido de la página web sino sobre todo tipo de nodos.

El problema de realizar el filtrado sobre toda la página es que además de mostrar más información de la deseada por el usuario, añade ruido a la información mostrada cuando se le aplica la tolerancia. Por ejemplo, estamos en la página oficial de la wikipedia en inglés sobre el país de Irak y estamos muy interesados en cualquier información sobre el idioma o idiomas que se hablan en el país asiático.

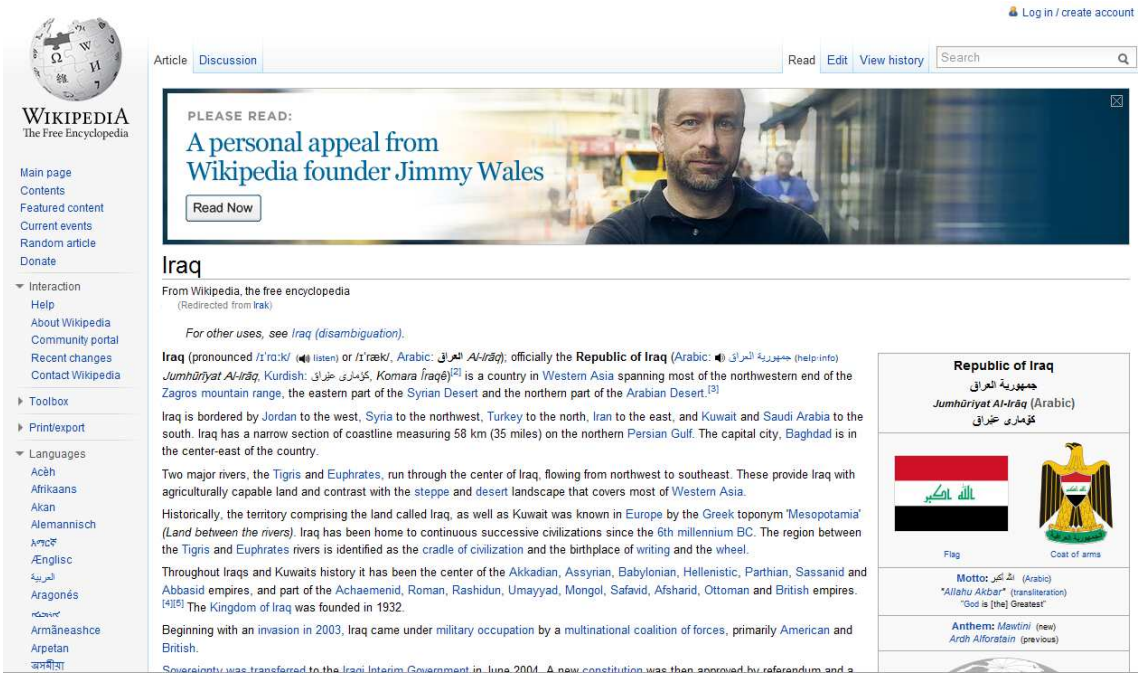


Figura 9. Iraq en la wikipedia inglesa.

Al introducir sobre el criterio la palabra “lang” (ya que estamos en la versión inglesa) para saber toda la información posible sobre el idioma obtenemos la siguiente figura:

Figura 10. Página de irak filtrado por el término lang.

Como podemos observar, se han encontrado bastantes nodos referentes a idiomas, como el árabe o el griego pero ha aparecido también información irrelevante como la sección de la izquierda referente a los distintos idiomas en los cuáles se encuentra escrito la página de información o la campaña especial sobre donativos en el cuál se encuentra sumida actualmente wikipedia. Cómo se puede observar, la información relevante al idioma no es suficiente así que tendríamos que aumentar la tolerancia para tener más texto.

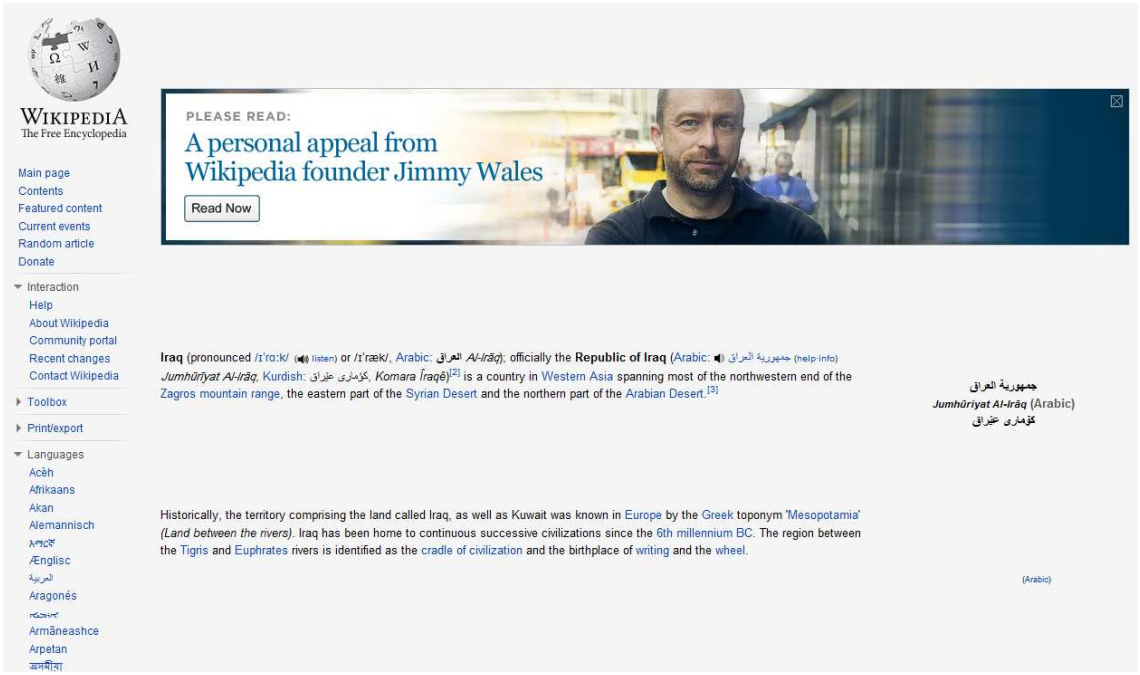


Figura 11. Página filtrada con una tolerancia de 1.

En el ejemplo se puede observar cómo se ha mostrado un poco más de texto (unas 3 líneas incrementando sólo 1 la tolerancia) pero el menú de la izquierda se muestra completamente, además de la publicidad sobre donativos está completa. Si quisiéramos aumentar sólo unas pocas líneas más la información sobre idiomas no sería posible ya que como el menú y la publicidad se encuentran muy cerca del nodo body (se encuentran a tan sólo dos niveles) mostrarían toda la información de la web haciendo el uso de la tolerancia como algo bastante inútil. Si fuésemos capaces de aislar la información importante en una página web de la información irrelevante (publicidad, menús), el algoritmo funcionaría mejor mostrando la información filtrada de una manera visual más legible al usuario. Con este objetivo, comenzamos un estudio sobre la detección de un bloque principal en una página web con el objetivo de mejorar la visualización y eliminar el ruido en la tolerancia.

3. Detección de bloques de contenido

La extracción de contenido es un área de investigación muy interesante para algunas aplicaciones. Básicamente consiste en la detección del contenido principal de un documento web. Es muy exitoso para enseñar en pantallas de pequeño tamaño, como PDAs o smartphones; y también para mejorar las tareas de procesamiento e indexación, evitando el tratamiento de contenido irrelevante como los menús o la publicidad. En este capítulo presentamos una nueva técnica para la extracción de contenido con una alta cobertura y precision. Para la extracción del contenido utilizaremos una interfaz que nos da acceso a la estructura de una página web para obtener la suficiente información precisa.

3.1. Introducción

La extracción de contenido es una de las mayores áreas de interés en la rama científica e industrial que abordan la recuperación de información en la Web. El interés se debe al uso de las aplicaciones prácticas de esta disciplina. Esencialmente, la extracción de contenido es un proceso que determina qué partes de una página contiene el contenido textual principal ignorando por tanto el contexto adicional como sería un menú, una barra de estado, publicidad, información del patrocinador, etc. La extracción de contenido es un caso particular de una disciplina más general llamada *detección de bloques* (aunque está más extendido el anglicismo *Block Detection)* que intenta aislar toda la información interesante de una página web. Por ejemplo, observemos el bloque que forma la página web de la figura 12, y en particular, el bloque principal delimitado por una línea punteada. Midiendo la página web de la figura anterior, observamos que cerca del 40% de la página web puede ser considerado como irrelevante [15]. Además, determinar el bloque principal de una página web es muy útil para indexadores y analizadores de texto que incrementan su rendimiento al sólo procesar información relevante. Otro uso interesante que se le pueda dar a la extracción de contenido principal se puede aplicar a pantallas pequeñas de dispositivos como PDAs o smartphones; y la extracción también hace más accesible la página web, al eliminar información irrelevante, a usuarios con problemas de visión o ciegos.

Algunas técnicas ya han sido propuestas para solucionar el problema de la extracción de contenidos. Algunas de ellas están basadas en la asunción de que la página web tiene una estructura (por ejemplo, que está basado en una etiqueta de tipo *table*) [16], que el principal contenido es continuo [17], que la página web mantiene un tipo de estructura ya predefinido [18], o que la página web está basada en una plantilla que se repite durante varias subpáginas propias [19]. Esto permite al sistema analizar varias páginas e intentar deducir los menus y anuncios viendo si cierto contenido se repite a lo largo de varias páginas.

Figura 12. Bloques de la página web de la CNN.

El principal problema de estas aproximaciones es una gran pérdida de generalidad. En general, requieren un previo conocimiento sobre dicha página web o un preparseado, o requieren que la página web tenga una estructura ya establecida e invariable. Esto es muy inconveniente porque las páginas modernas están principalmente basadas en etiquetas de tipo *div*, y por tanto no requieren estar jerárquicamente organizadas. Es más, actualmente, algunas páginas se generan de manera automática y son dinámicamente regeneradas haciendo aún más difícil (o incluso imposible) analizarlas por adelantado.

Sin embargo, hay otras aproximaciones que son capaces de trabajar online y en tiempo real. Una de estas aproximaciones es la técnica presentada [19]. Esta técnica usa un vector llamado *content code vector* (CCV) que representa todos los caracteres que computa de un determinado documento, almacenando para cada uno de ellos si es un carácter o una etiqueta. Una vez calculado el CCV, se computa un ratio (conocido como *content code ratio*) para identificar la cantidad de código y contenido que rodea los elementos del CCV. Finalmente, con esta información, pueden determinar qué partes contienen el bloque principal. Otra importante aproximación también está basada en el etiquetado de un documento [20]. Esta aproximación se basa en el uso de los llamados *tag ratios* (TR). Dada una página web, el TR se calcula para cada línea como el número de caracteres que no pertenecen a etiquetas HTML dividido por el número de etiquetas de la línea. El principal problema de estas aproximaciones basadas en caracteres o líneas de código fuente como estos dos es el hecho de ignorar completamente la estructura de una página web. Usar caracteres o palabras como unidades de información independientes e ignorando sus interrelaciones produce una importante pérdida de información que está presente y explícita en la página web y que hace al sistema fallar en algunas situaciones.

3.2. Extracción de texto usando la estructura DOM

Esta técnica está basada en la noción de *char-nodes ratio* (CNR), el cual nos muestra la relación entre contenido de tipo texto y el contenido etiqueta de cada nodo del árbol DOM.

Esta métrica tiene una propiedad muy interesante que consiste en considerar los nodos como bloques donde la información interna es agrupada e invisible usando la estructura DOM. Siendo, el CNR de un nodo interno, la suma de las proporciones de la suma de texto y número de hijos de sus descendientes. Por tanto, el CNR de un nodo n, CNR(n), nunca es mayor que todos sus hijos, al ser sus datos una suma aritmética de éstos ya que el propio nodo no contiene texto sino que es heredado del propio texto que tiene cada uno de sus nodos hijos. Esto es muy interesante ya que nos permite detectar un bloque de contenido relevante aunque este posea nodos sin texto pertenecientes al bloque.

En resumen, el método para la extracción de contenido funciona de la siguiente manera: Primero se computa el CNR de cada nodo de manera recursiva en el árbol DOM empezando por la raíz y llegando hasta las hojas. Posteriormente seleccionamos los nodos con más alto porcentaje de CNR y, empezando desde éstos, atravesamos el árbol DOM desde abajo hacia arriba buscando nodos contenedores (por ejemplo nodos de etiqueta *table*, *div*, etc.) que puedan contener a los nodos más relevantes. Cada uno de estos contenedores representa un bloque HTML. Finalmente elegimos el bloque que contenga la mayor cantidad de bloques relevantes. Todos estos pasos pueden hacerse con un coste lineal *n* siendo *n* el tamaño del árbol DOM.

El cálculo de los CNR se realiza de forma trivial aprovechando la API de la estructura DOM que contiene un texto para obtener el texto de un nodo, este método puede discriminar entre diferentes tipos de contenidos de texto (por ejemplo, texto plano, scripts, CSS, comentarios, etc.). Sin embargo, no existe un método para calcular de manera automática el número de nodos descendientes de un nodo dado; por ello, la computación del CNR's se hace de manera acumulativa y mediante un proceso recursivo que explora el árbol DOM contando el texto y el número de nodos descendientes de cada nodo. Éste proceso nos permite detectar nodos irrelevantes que llamaremos '*NoContentNode*'. Son nodos sin contenido de texto (nodos con etiqueta IMG), nodos utilizados principalmente para uso navegacional usados principalmente para menús (por ejemplo, los nodos con etiquetas NAV o A) o nodos totalmente irrelevantes (por ejemplos, nodos con etiquetas SVG, VIDEO o CANVAS). Esto es una importante ventaja sobre otras técnicas que confían en el análisis de caracteres simples o de texto. Estas técnicas no pueden ignorar el ruido sobre el análisis si no realizan una previa fase de pre-procesado para eliminar este contenido irrelevante. Cabe recalcar que nuestra técnica no elimina ni manipula estas etiquetas sino que simplemente ignora o penaliza en sus CNRs estos nodos irrelevantes.

Ejemplo 1: Página web en la wikipedia en inglés sobre "Information Retrieval"

```
<body>
  <h1 id="firstHeading" class="firstHeading">Information retrieval</h1>
  <div id="content">
          <p><b>Information retrieval</b>(IR)is the science of searching for documents,
        for information within documents, and for metadata about documents(...)</div>
  <div class="portal" id="p-lang">
  <h5>Languages</h5>(....)</div>
```

```
<div id="footer">
        <li id="footer-info-lastmod">This page was last modified on 5 October
         2010 at 23:32. </li>
        <li id="footer-info-copyright">Text is available under the
       <a rel="license">Creative Commons Attribution-ShareAlike License</a>
     Additional terms may apply. See Terms of Use for(....)</div>
</body>
```

El Algoritmo 1 obtiene recursivamente el CNR de cada nodo empezando desde un nodo raíz del árbol DOM. Por cada nodo se añaden tres nuevos atributos al nodo siendo el peso computado, el número de caracteres y el CNR. El número de caracteres computados ignora caracteres especiales tales como espacios o saltos de línea. Esto hace al algoritmo independiente del formateo de páginas web (por ejemplo, estas páginas que organizan el código en poco espacio no influyen en el CNR).

El algoritmo distingue entre tres tipos de nodos:

- Los *textNode*, los cuales son nodos de tipo DOM que contienen únicamente texto plano y que siempre son una hoja siendo su peso de 1.
- Los *nonContentNode*, que representan todos aquellos nodos irrelevantes con un CNR de 0 y un peso de 1.
- El resto de nodos cuya etiqueta no pertenezca al grupo de nodos irrelevantes. Todos los métodos, como *addAttribute*, o atributos, como *innerText*, usados en el algoritmo pertenecen al estándar usado en el modelado DOM.

Algorithm 1 Algoritmo para computar los ratios carcteres por nodos

Entrada: Un Arbol DOM $T = (N, E)$ y un nodo root T, $root \in N$
Salida: Un Arbol DOM $T' = (N', E)$

```
function ComputeCNR(nodo n)
  case n.nodeType of
  "textNode": n.addAttribute('weight',1);
              n.addAttribute('textLenght',n.innerText.length);
              n.addAttribute('CNR',n.innerText.length);
              return n;
  "nonContentNode": n.addAttribute('weight',1);
                    n.addAttribute('textLenght',0);
                    n.addAttribute('CNR',0);
                    return n;
  otherwise: descendants = 0;
             charCount = 0;
             for each child ∈ n.childNodes do
                 newChild= ComputeCNR(child);
                 charCount = charCount + newChild.textLenght;
                 descendants = descendants + newChild.weight;
             n.addAttribute('weight',descendants);
             n.addAttribute('textLenght',charCount);
             n.addAttribute('CNR',charCount/descendants);
             return n;

return computeCNR(root)
```

Cuando el CNR ya ha sido calculado, seleccionamos los nodos con el CNR más alto. En nuestra implementación cogemos el 10% de los nodos con mejor porcentaje porque los experimentos demostraron que dicha pequeña cantidad era la idónea ya que la gran mayoría de ellos pertenecerían al bloque de contenido principal de la página web. Entonces, propagamos estos nodos desde abajo hacia arriba, eligiendo el nodo más superior con mejor porcentaje.

El segundo paso está reflejado en el Algoritmo 2. Este algoritmo coge los nodos del paso previo introduciéndolos en una lista y elimina aquellos nodos que son descendientes de otros nodos de la misma lista (línea 1). Entonces, en la líneas 2 y 3 se procede a agrupar dos nodos que comparten el mismo nodo padre (eliminando de la lista ámbos nodos pero añadiendo en la lista al nodo Padre). El proceso iterativo va poco a poco reduciendo el número de nodos de lista produciendo un conjunto de nodos que representan bloques de la página web. Cuando el proceso iterativo no reduce el número de nodos en la lista, seleccionamos el nodo que contenga la mayor cantidad de texto como el bloque final.

Ejemplo. Considerando el código fuente en HTML del ejemplo 1 y su árbol asociado mostrado en la figura 13. El Algoritmo 1 computa el CNR asociado a todos los nodos del árbol DOM. Todos los CNRs se muestran en la figura 14.

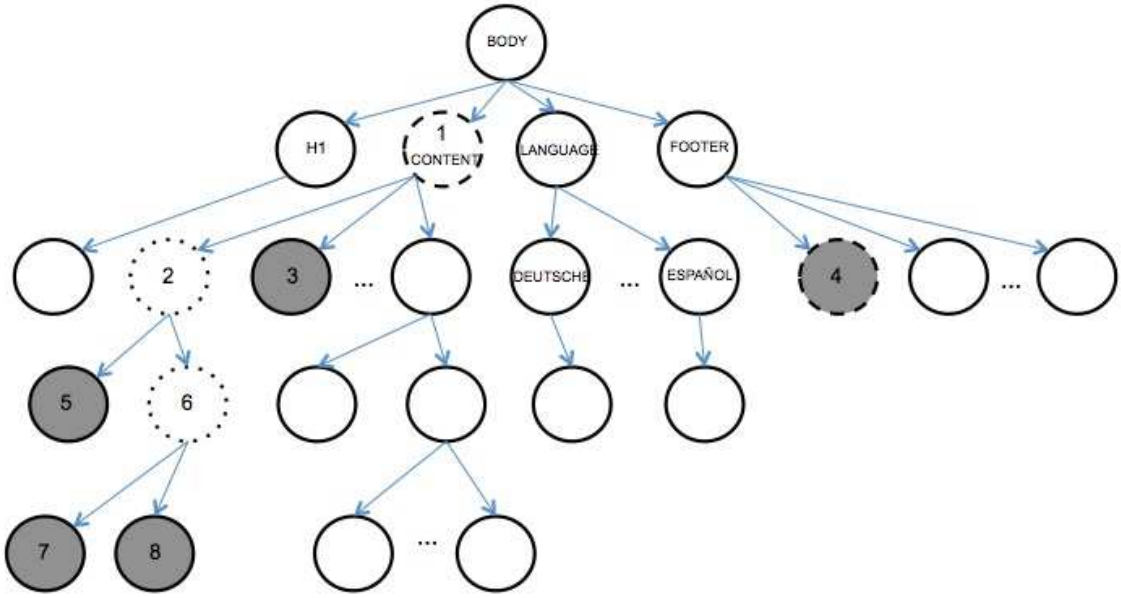


Figura 13. Representación en forma de árbol del ejemplo 1.

Después de haber computado los CNRs cogemos el 10% de los nodos con mejor porcentaje, considerando que los nodos oscurecidos en la figura 13 representan los nodos con mejor ratio. Entonces, usamos el algoritmo 2 para identificar los nodos más relevantes en la página web. Inicialmente, todos los nodos oscurecidos son la lista de bloques principales. Entonces, como el nodo 7 y 8 son hermanos, en la primera iteración serían eliminados y se agregaría el nodo 6 a la lista. En la segunda iteración, los nodos 5 y 6 son eliminados de la lista y el nodo 2 es añadido. Finalmente, en la tercera iteración, los nodos 2 y 3 son eliminados y el nodo 1 es añadido a la lista. Al final, en la lista sólo quedarían los nodos finales 1 y 4. Entre ambos nodos, se seleccionaría el nodo con mayor cantidad de texto de los dos nodos finales.

Seguramente se seleccionaría el nodo 1 como el bloque principal de texto debido a que, por regla general, el nodo que tiene más descendientes que han pertenecido a la lista inicial (del 10% con mejor proporción) suele ser el nodo con más texto debido a la estructura en forma de árbol de la página web.

3.3. Evaluación Empírica

La técnica presentada se encuentra a disposición pública tanto para su testeo como su revisión del código fuente. La implementación permite al programador activar las transformaciones de la técnica y parametrizarlas siguiendo un orden y ajustar la cantidad de nodos confiables cambiando el umbral de detección de estos nodos. Para determinar la configuración por defecto, fue inicialmente testeado con una colección de páginas web que nos permitió configurar y afinar los parámetros. Entonces, realizamos el experimento sobre 50 páginas web reales y online con gran cantidad de visitas para obtener una medida de rendimiento del algoritmo calculando la cobertura, la precisión y la métrica F1.

Para los experimentos seleccionamos páginas de dominios diferentes y con distintas apariencias y estructuras internas con el objetivo de estudiar el rendimiento de la técnica en diferentes contextos (por ejemplo, páginas de compañías, artículos, noticias, foros, etc.). La selección del bloque principal se realiza de manera manual. Después, comparamos el bloque principal observado de manera manual con el obtenido de manera automática con nuestro algoritmo.

La tabla siguiente resume los resultados obtenidos. La primera columna contiene la dirección donde se encuentra la URL de la página web evaluada en el benchmarking. Por cada benchmark, la columna *nodos DOM* muestra el número de nodos totales de la página web; la columna *Bloque Principal* nos muestra el número de nodos que contiene según nuestro algoritmo el bloque principal; la columna *cobertura* nos muestra el número nodos relevantes obtenidos por el algoritmo dividido por el número total de nodos relevantes reales que están en el bloque principal; la columna *precisión* nos muestra el número de nodos relevantes reales partido por el número de nodos obtenidos según nuestro algoritmo; finalmente, la columna F1 nos muestra la métrica F1 que se obtiene como (2*Precisión*Cobertura) / (Precisión * Cobertura).

Benchmark	nodos DOM	Bloque Principal	Recall	Precisin	F1
www.wikipedia.org	870 nodos	712 nodes	100 %	100 %	100 %
www.facebook.com	744 nodos	293 nodes	28.6 %	100 %	44.47 %
www.nytimes.com	742 nodos	217 nodes	100 %	49.7 %	66.39 %
www.engadget.com	2897 nodos	1345 nodes	100 %	100 %	100 %
us.gizmodo.com	2205 nodos	1375 nodes	100 %	84 %	91.3 %
googleblog.blogspot.com	1138 nodos	743 nodos	100 %	100 %	100 %
www.bbc.co.uk	401 nodos	111 nodos	100 %	4.98 %	9.49 %
www.vidaextra.com	1144 nodos	602 nodos	100 %	100 %	100 %
www.gizmologia.com	926 nodos	415 nodos	100 %	100 %	100 %
www.elpais.com	3017 nodos	120 nodos	100 %	100 %	100 %
www.elmundo.es	1722 nodos	416 nodos	100 %	100 %	100 %
www.ox.ac.uk	279 nodos	30 nodos	100 %	28 %	43.75 %
www.thefreedictionary.com	1170 nodos	509 nodos	100 %	100 %	100 %
www.nlm.nih.gov	320 nodos	156 nodos	100 %	56.52 %	71.56 %
www.scielosp.org	563 nodos	458 nodos	100 %	100 %	100 %
www.wordreference.com	269 nodos	95 nodos	100 %	57.23 %	72.79 %
en.citizendium.org	1645 nodos	1478 nodos	100 %	100 %	100 %
knol.google.com	601 nodos	219 nodos	100 %	100 %	100 %
www.healthopedia.com	557 nodos	21 nodos	100 %	21 %	34.7 %
www.filmaffinity.com	1198 nodos	153 nodos	100 %	100 %	100 %
www.umm.edu	290 nodos	30 nodos	100 %	22.22 %	36.42 %
www.microsiervos.com	604 nodos	382 nodos	100 %	68.83 %	81.54 %
abcnews.go.com	907 nodos	102 nodos	100 %	44.16 %	61.27 %
www.latimes.com	1056 nodos	22 nodos	100 %	100 %	100 %
www.philly.com	378 nodos	30 nodos	100 %	100 %	100 %
www.blogdecine.com	1567 nodos	24 nodos	100 %	8.33 %	15.38 %
www.cnn.com	597 nodos	248 nodos	100 %	67.21 %	80.39 %
www.lashorasperdidas.com	87 nodos	30 nodos	100 %	100 %	100 %
www.cbc.ca	847 nodos	138 nodos	100 %	100 %	100 %
www.appleweblog.com	1013 nodos	475 nodos	5.9 %	100 %	11.15 %
www.applesfera.com	1215 nodos	721 nodos	7.49 %	100 %	13.94 %
www.guardian.co.uk	1111 nodos	59 nodos	100 %	100 %	100 %
www.news.cnet.com	2023 nodos	169 nodos	100 %	71.01 %	83.05 %
www.venturebeat.com	263 nodos	107 nodos	100 %	100 %	100 %
www.computerworld.com	558 nodos	62 nodos	100 %	100 %	100 %
www.usatoday.com	1118 nodos	523 nodos	100 %	100 %	100 %
www.cbssports.com	1450 nodos	232 nodos	100 %	67.05 %	80.28 %
www.nationalfootballpost.com	565 nodos	23 nodos	100 %	9.62 %	17.55 %
ncaabasketball.fanhouse.com	885 nodos	78 nodos	100 %	40.20 %	57.35 %
www.sportingnews.com	1394 nodos	79 nodos	100 %	72.48 %	84.05 %
www.hoopsworld.com	629 nodos	112 nodos	100 %	100 %	100 %
profootballtalk.nbcsports.com	394 nodos	28 nodos	100 %	45.17 %	62.23 %
www.thehollywoodgossip.com	362 nodos	44 nodos	100 %	100 %	100 %
www.rollingstone.com	993 nodos	29 nodos	100 %	20.42 %	33.92 %
popwatch.ew.com	919 nodos	93 nodos	100 %	100 %	100 %
www.people.com	923 nodos	56 nodos	100 %	32 %	48.49 %
www.cinemablend.com	495 nodos	59 nodos	100 %	37.34 %	54.38 %

Figura 14. Benchmarking.

Los experimentos revelan que en algunos casos, el bloque obtenido es exactamente el mismo que el bloque relevante (F1=100%) y, por general, el cobertura es del 100%. Es decir, normalmente el bloque obtenido suele contener toda la información importante. Se observa una importante propiedad en todos los experimentos: en todos los casos, o la cobertura o la precisión son siempre del 100%. Este fenómeno no sucede por casualidad, es una consecuencia directa de la manera de seleccionar el bloque. Como nuestra técnica es ascendente, es decir, se mueve desde los nodos considerados hoja (se encuentran en los niveles más bajos del árbol DOM) hacia los nodos superiores, agrupándose de manera iterativa alcanzando con cada iteración niveles más altos y cercanos al nodo con la etiqueta *body*. Debido a esto se producen 4 casos:

1.- Cuando detectamos el nodo n como el bloque principal obtenemos una cobertura y una precisión del 100%.

2.- Si elegimos un nodo descendiente del nodo n entonces la precisión es del 100%.

3.- Si elegimos un nodo ancestro al nodo n entonces obtenemos una cobertura del 100%.

4.- Finalmente, si no seleccionamos un nodo ancestro o descendiente entonces obtenemos un 0% tanto en el cobertura como en la precisión. Este caso es muy difícil que suceda ya que tendría que haber un nodo no relevante que contiene más texto y, por tanto, más información que el propio bloque principal. En todos los experimentos realizados no encontramos ninguna página que contuviera este resultado.

Por tanto, podemos obtener la ventaja de tener una técnica con la interesante característica de que obtendremos o bien el 100% en el cobertura o el 100% en la precisión sin importar la página web que seleccionemos. Podríamos acercarnos más a una posibilidad o a la otra según la variación que realicemos en el Algoritmo 2 poniendo un valor más restrictivo (al seleccionar menos bloques nos aseguramos una precisión del 100%) o siendo más flexibles (nos aseguramos seleccionar bloques más superiores y, por tanto, obteniendo una cobertura del 100%).

Toda la información relacionada con el experimento, el código fuente de los benchmarks y el código fuente de la herramienta se puede encontrar en:

http://www.dsic.upv.es/jsilva/CNR

3.5. Resultados.

La extracción de contenido es exitosa no sólo para usuarios finales sino también a algunos sistemas y herramientas como una primera etapa de indexadores. Permite extraer la parte relevante de una página web permitiéndonos ignorar el resto del texto que puede ser irrelevante, inútil e incluso, peor aún, molesto. Hemos presentado una nueva técnica para la extracción de contenido que no sólo analiza las relaciones entre texto y etiquetas sino que usa también la estructura DOM de la página web para identificar los bloques que agrupan estos nodos con mayor proporción de texto.

La estructura DOM no sólo nos permite mejorar la detección de bloques sino que nos permite descartar partes de la página web que contienen gran cantidad de información textual pero no están contenidos en el bloque principal al pertenecer a otros contenedores HTML. Nuestros resultados y experimentos han demostrado que es una técnica muy satisfactoria.

Además de ser útil para la extracción de contenido, la técnica presentada podría ser usada también para la detección de bloques. En este contexto, podría detectar todos los bloques en una página web aplicando el algoritmo CNR presentado de manera iterativa extrayendo un bloque tras otro. Es decir, el algoritmo encuentra el bloque más relevante; entonces, lo eliminamos del árbol DOM dicho bloque principal y volvemos a detectar el próximo bloque más relevante. Aunque aún no hemos formalizado esta idea, hemos podido comprobar con varios experimentos que, por regla general, al menos los tres primeros bloques más relevantes podrían ser extraídos mediante esta técnica. Otra interesante línea de investigación sería usar esta técnica para detectar los menús de una página web. Un estudio preliminar nos mostró información interesante si modificamos nuestro ratio CNR, que

recompensa los nodos con texto y castiga los nodos con enlace, para que funcione de manera totalmente inversa. Es decir, convertir la técnica CNR en una técnica HNR (Hyperlink Node Ratio) para descubrir bloques con grandes cantidades de enlaces. Si buscamos el nodo con el mayor ratio de HNR podríamos encontrar el bloque donde se encuentran los menús de las páginas web.

4. Implementación

4.1. ¿Porqué una extensión Firefox?

Para la implementación de los algoritmos propuestos se ha utilizado una extensión de Firefox ya que permite la manipulación y consulta de los datos de una página web de manera directa. Las extensiones de Firefox se pueden alojar en el servidor oficial de Firefox (http://www.firefox.com/extension). Un complemento de Firefox se puede desarrollar de dos maneras: o bien realizando una extensión o bien un plugin. Una extensión es un conjunto de ficheros CSS y JavaScript, en el que el principal motor de rendimiento es el motor JavaScript del navegador (en nuestro caso es Gekko [23] al realizarse sobre Firefox aunque no habría problema alguno en adaptarlo a Chrome [24]). Este tipo de extensiones necesita de menos permisos para poder instalarlos y, si se ha subido al repositorio oficial, goza de una comprobación de seguridad por parte de profesionales expertos. Por el contrario, los plugins tienen la potencia de una extensión además de poder ejecutarse de manera nativa en un ordenador (por lo que ya no tienen esa independencia del sistema operativo) además de que necesitan de permisos de administrador para ser instalados en un ordenador.

Aunque actualmente existen plugins en diversos navegadores (como Internet Explorer, Opera o Google Chrome), se eligió Firefox por ser de los primeros y donde mejor está testeado y comprobado el funcionamiento de los plugins. Además de gozar de un gran control de seguridad en los plugins (realizado por profesionales que comprueban que la herramienta no tiene oculto malware o phising) y de actualizaciones automáticas. Además, actualmente se puede realizar con gran facilidad extensiones para Firefox lo que hace más atractivo el desarrollo en esa plataforma.

A pesar de haber elegido la plataforma Firefox para la realización del plugin, no supondría un gran problema el integrar la herramienta en otros navegadores, ya que se han utilizado tecnologías y lenguajes estándares. Tal vez el único problema podría ser Internet Explorer por disponer de variaciones importantes a los estándares de JavaScript, pero que con la última versión (Internet Explorer 9) tiene gran compatibilidad con los últimos estándares web.

4.2. Estructura de una extensión firefox

Las extensiones en Firefox son archivos con extensión .xpi (a pesar de esta extensión, es un archivo comprimido en formato zip). Para la creación de una extensión, Firefox contiene una herramienta web en la siguiente dirección:

https://addons.mozilla.org/es-ES/developers/tools/builder

donde permite introducir los datos básicos (datos identificativos de la herramienta, propietario de la herramienta, descripción, versiones de soporte...). A continuación, se crea un archivo xpi que funciona simplemente arrastrándolo y soltándolo sobre el navegador.

El plugin consta en su creación de 2 carpetas (*chrome* y *default* aunque pueden variar a gusto del desarrollador) y de dos archivos necesarios para la identificación del plugin por parte de Firefox (chrome.manifest e install.rdf). Por regla general, la carpeta *content* contiene toda la información sobre información y contenido del idioma como de los archivos importantes en ejecución de la extensión mientras que la carpeta default suele contener información de preferencia por defecto.

Los 2 ficheros contienen la parte más importante para la instalación. El fichero install.rdf contiene toda la información necesaria y descriptiva de la instalación (nombre e identificador de la extensión, número de versión, versiones de Firefox compatibles…) en formato XML; mientras que chrome.manifest contiene toda la información necesaria para instalar la aplicación (cabe destacar que Firefox pide el fichero o carpeta donde se encuentra el contenido, idiomas y apariencia del plugin). El plugin tiene que tener la ruta y un fichero .xul en el cual se encuentra toda la información tanto de la interfaz gráfica como de los archivos JavaScript y CSS.

4.3. Tecnología utilizada dentro de la extensión de Firefox

Para la realización de la extensión permite utilizar gran parte de los lenguajes web estándar además de alguno más que forman internamente el navegador web. En nuestra extensión hemos utilizado las siguientes tecnologías:

4.3.1. XUL

XUL (acrónimo de XML-based User-interface Language) [25] es un lenguaje creado por Mozilla basado en XML que nos permite realizar aplicaciones ricas multiplataformas que pueden funcionar estando o no conectadas a Internet. Estas aplicaciones son fácilmente personalizables con texto alternativo, gráficos o diseño pudiendo ser fácilmente localizables (el propio navegador Firefox se encarga de seleccionar entre los distintos ficheros de idiomas según la localización del navegador).

Actualmente XUL es utilizado por distintos programas del grupo Mozilla como Thunderbird, Firefox o Songbird entre ellos. XUL tiene la siguiente especificación en tres carpetas:

-Content: Donde se encuentran los documentos XUL que definen el diseño de la interfaz.

-Skin: Contiene las hojas de estilos (CSS) y las imágenes que forman parte de la interfaz (en nuestro, caso será la barra de herramientas u opciones de la extensión en Firefox).

-Locale: Los documentos DTD se encuentran aquí, estos documentos facilitan la localización de páginas XUL. Dicha carpeta, tiene tantas carpetas como a idiomas posibles haya sido traducida la herramienta, siendo el nombre de cada carpeta el acrónimo del idioma siendo el acrónimo en-US la versión en inglés (y la versión por defecto) mientras que la versión es-ES es la versión en español.

Las extensiones Firefox contienen estas tres carpetas dentro de la carpeta chrome que se encuentra en la raíz de la extensión. Al cargar la extensión, se carga el fichero XUL que se

encuentra dentro de la carpeta content especificado en el fichero chrome.manifest. Al cargar dicho fichero, en la cabecera se especifica la ruta del fichero CSS (para los estilos) y del DTD (para la localización), el cuál contiene la ruta desde la raíz sin especificar la carpeta de idioma. Por ejemplo:

<!DOCTYPE window SYSTEM "chrome://slicetoolbar/locale/es-ES/slicetoolbar.dtd" >

Pasa a ser, el navegador Firefox se encarga de elegir la respectiva carpeta del idioma:

<!DOCTYPE window SYSTEM "chrome://slicetoolbar/locale/slicetoolbar.dtd" >

4.3.2. DOM

Las páginas web procesadas por nuestra herramienta se encuentran estructuradas en HTML (siendo HTML un lenguaje XML con una estructura básica y estándar) y se necesita de alguna estructura fácil y cómoda para acceder a los datos. Esa estructura es representada con el modelo DOM (Document Object Model) que es esencialmente una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos y una interfaz para acceder a ellos y manipularlos (cita es.wikipedia.org/wiki/Document_Object_Model). Dicha API es accesible desde Javascript. Cabe tener en cuenta que la primera época de la navegación web, la mayoría de los navegadores tenía su propia implementación de DOM por lo que cada uno tenía sus propios métodos y exigía por parte del desarrollador un trabajo extra en hacerlo compatible entre los distintos navegadores. Sin embargo en la actualidad el estándar de DOM es implementado por prácticamente todos los navegadores.

4.3.3. JavaScript

Javascript es un lenguaje de scripting orientado a objetos, basado en prototipos, sin tipos y liviano, utilizado para acceder a objetos de aplicaciones web (cita: es.wikipedia.org/wiki/Javascript). Principalmente, se utiliza integrado en un navegador web permitiendo el desarrollo de interfaces de usuario mejoradas y páginas web dinámicas. Javascript es un dialecto de ECMAscript y se caracteriza por ser un lenguaje basado en prototipos, con entrada dinámica y con funciones de primera clase. Javascript ha tenido influencia de multitud de lenguajes y se diseñó con una sintaxis similar al lenguaje de programación Java, aunque más fácil de utilizar por personas que no programan.

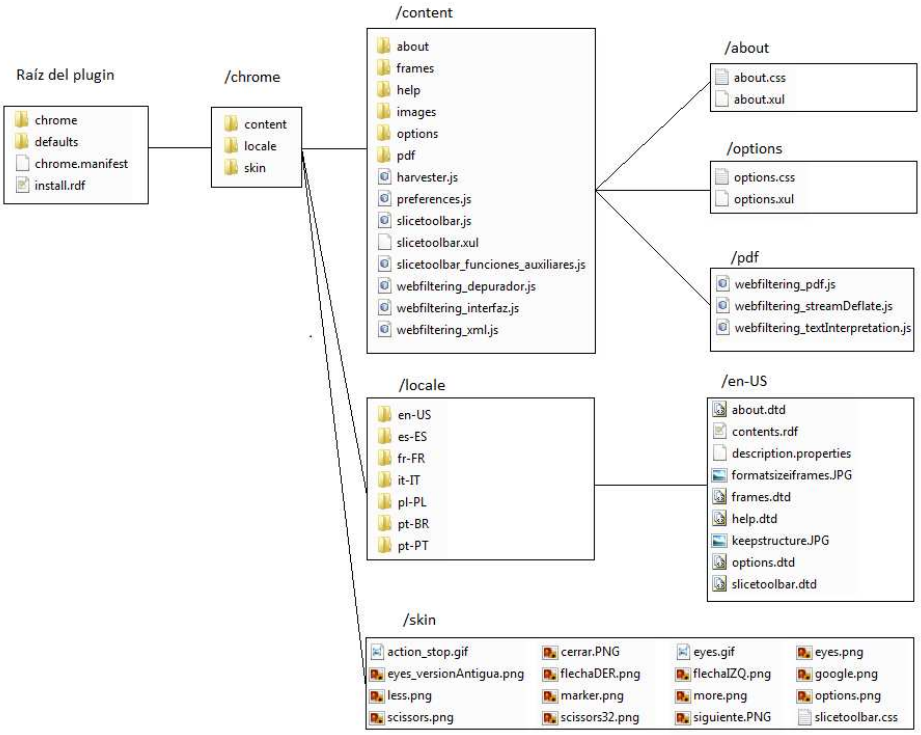
Javascript es bastante utilizado principalmente porque se ejecuta desde el cliente y no desde el servidor. Además, desde la llegada de AJAX a la web, las aplicaciones han ido ganando vistosidad llegando a tener un parecido cercano a aplicaciones de escritorio.

4.4. Webfiltering plugin

4.4.1. Estructura del fichero xpi

El plugin sigue la misma estructura básica que todos los plugins realizados en Firefox mediante XUL, con las únicas diferencias que la estructura de la carpeta content está dividida según las opciones avanzadas del plugin. La carpeta para la localización (/chrome/locale) contiene todos los archivos, principalmente DTD, para los distintos idiomas en los que está traducido el plugin (español, inglés, francés, italiano, polaco y portugués). Principalmente, los archivos son slicetoolbar.dtd y options.dtd que conlleva toda la localización de la parte visual de la barra. La carpeta para el contenido está muy estructurada en carpetas para una fácil y división de las tareas, destacando entre varias la carpeta help (contiene todas las imágenes y archivos web que se visualizan al abrir la opción de ayuda) y pdf (donde se encuentran varios archivos para la conversión de los pdf a html).

Estructura interna resumida del plugin webfiltering.xpi



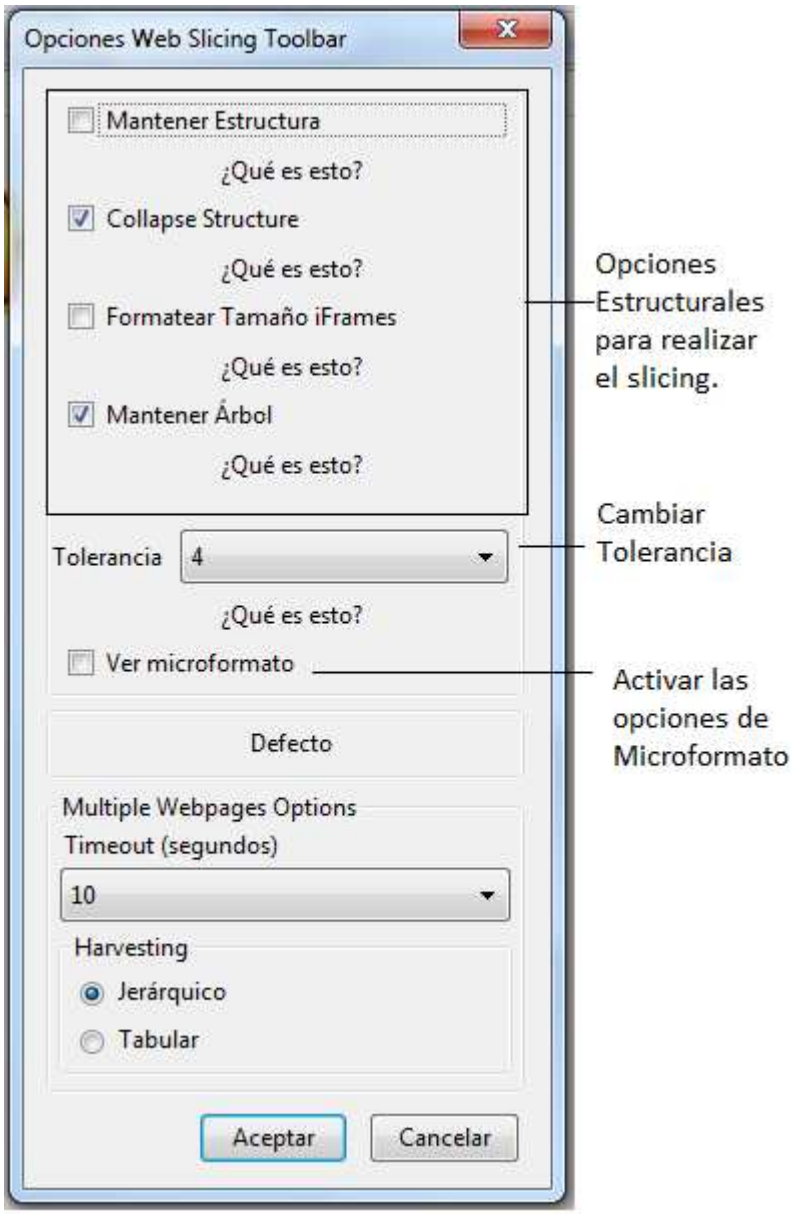
4.4.2. Instalación del plugin

La instalación se puede realizar o bien arrastrando el fichero xpi hacia el navegador web o bien instalarlo (permitiendo los permisos necesarios) desde la web oficial (https://addons.mozilla.org/es-ES/firefox/addon/5823/). La instalación del plugin conlleva la aparición de la siguiente barra superior en el explorador web:

La barra pasa a formar parte de la parte superior del plugin de Firefox. Para realizar una rápida ejecución de la página sólo tendríamos que escribir una palabra a buscar en el criterio y pulsar *intro* o el botón de filtrar. La estructura de la barra es:

- *Criterio para el filtrado*: Se escriben la palabra/s a buscar cuando se realiza el filtrado.
- *Realizar filtrado*: Botón para la realización del filtrado, si la acción no obtiene ningún resultado factible, se avisará al usuario del fallo en el filtrado y se recargará automáticamente la página.
- *Opciones de resaltado*: permite resaltar todos los nodos que coinciden con el criterio. Las flechas sirven para moverse a través de los nodos que coinciden con el criterio.
- *Filtrado inverso*: permite realizar un filtrado pero sobre los nodos que no coinciden con el criterio.
- *Filtrado continuo*: Permite que cada vez que se cambia de página se realice automáticamente el filtrado al cargarse completamente la página.
- *Cambiar Tolerancia*: Permite cambiar la toleración aumentándola o disminuyéndola 1 a 1 o bien aplicando directamente un valor.
- *Recargar*: Se encarga de recargar la página inicializando todos los parámetros ya almacenados.
- *Opciones*: Para modificar opciones de filtrado.

Cuando se selecciona *opciones*, se obtiene la siguiente interfaz:



La opción de formatear tamaño iframes se encarga de convertir, antes del filtrado, las etiquetas de tipo iframe en div. Este cambio provoca un pequeño cambio visual en los iframes.

Las opciones *mantener árbol*, *mantener estructura* y *colapsar estructura* son opciones visuales y estructurales cuando se realiza el filtrado.

4.4.3. Funcionamiento del plugin

El funcionamiento del plugin es muy sencillo, imaginemos que estamos en la página oficial de venta de Apple (figura 15) y estamos interesados en observar toda la información relevante al criterio de búsqueda “ipad”, sólo tendríamos que introducir en el criterio la palabra “ipad” y pulsar la opción de filtrado.

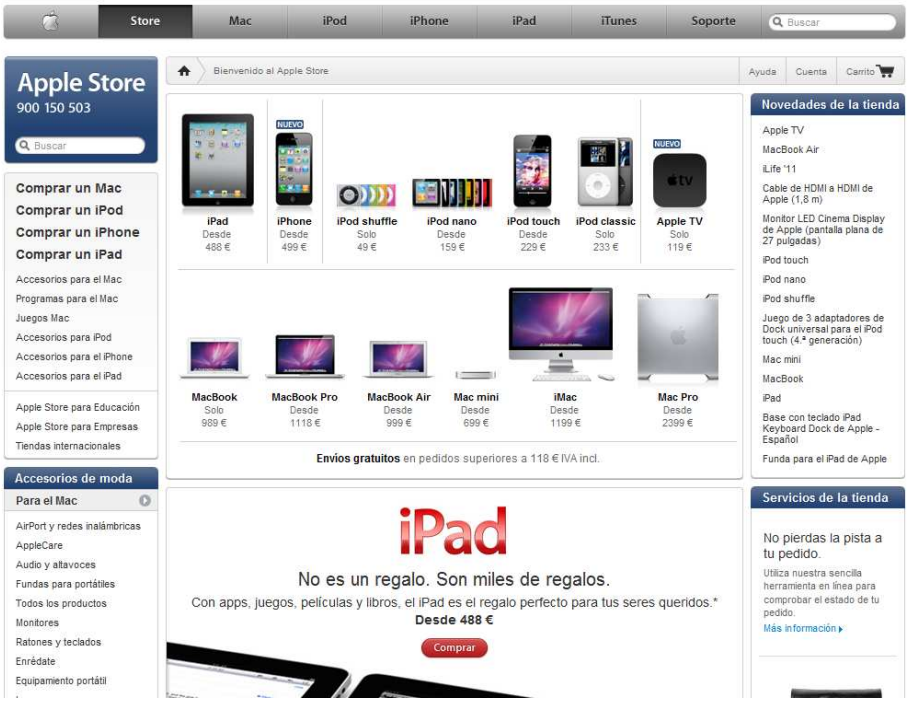


Figura 15. Store oficial de la página de apple

Ahora bien, además de realizar el filtrado, hay que seleccionar una opción de visualización para mostrarnos el contenido relevante con la búsqueda:

- Mantener Estructura (keep structure): se basa en ocultar visualmente los nodos que no coinciden con el criterio, es decir, los nodos no visibles siguen manteniendo el espacio. Por el contrario, los nodos que son padre de manera directa e indirecta con un nodo relevante no son ocultados.

Figura 16. Mantener estructura sobre la apple Store.

- Colapsar estructura (collapse structure): se basa en aplicarles . A los nodos que son ancestro de algún nodo relevante no se les aplica ningún cambio. La estructura visual desaparece pero internamente se mantiene (la relación entre los respectivos nodos se sigue manteniéndo).

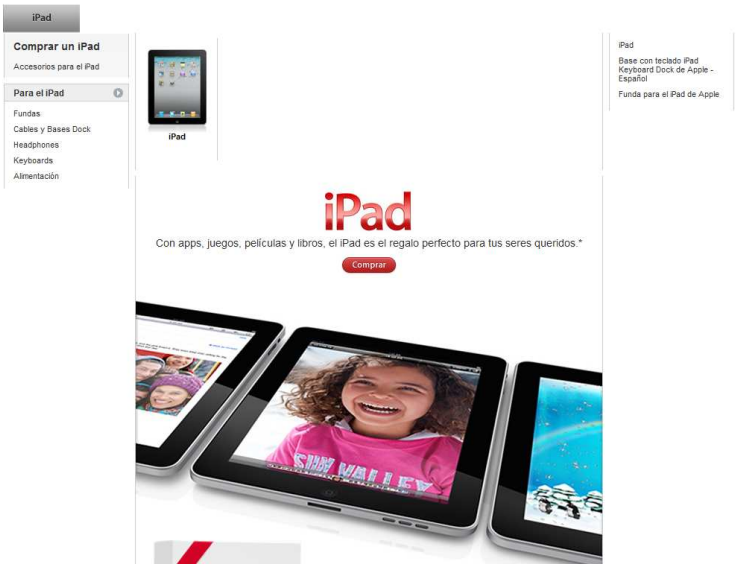


Figura 17. Colapsar estructura sobre la Apple Store.

- No mantener árbol (no keep tree): se basa en eliminar toda relación del nodo body respecto a sus hijos, colgando sólo de él los nodos que son inmediatamente relevantes. La estructura tanto visual como interna no se mantiene. Cambiar la estructura interna hace que las apariencias visuales de los nodos varíe porque muchos padres cambien la apariencia de sus hijos (al estar éstos contenidos dentro de su propio árbol) y al romperse esta relación éstos varían su aspecto visual. También hay que tener que existen ciertos nodos que necesitan de un nodo padre para tener toda su relevación, éstos nodos son llamados 'nodos huérfanos'. Por ejemplo, un nodo huérfano es una etiqueta de tipo 'td' (etiqueta columna) o 'tr' (etiqueta fila) que necesitan de un nodo de tipo *table* para tener todo su significado. Cuando tengamos un nodo de éste tipo intentaremos convertir los nodos huérfanos en nodos completos de significado agregándole un clon del mismo padre que sólo tiene de nodo hijo nuestro nodo relevante.

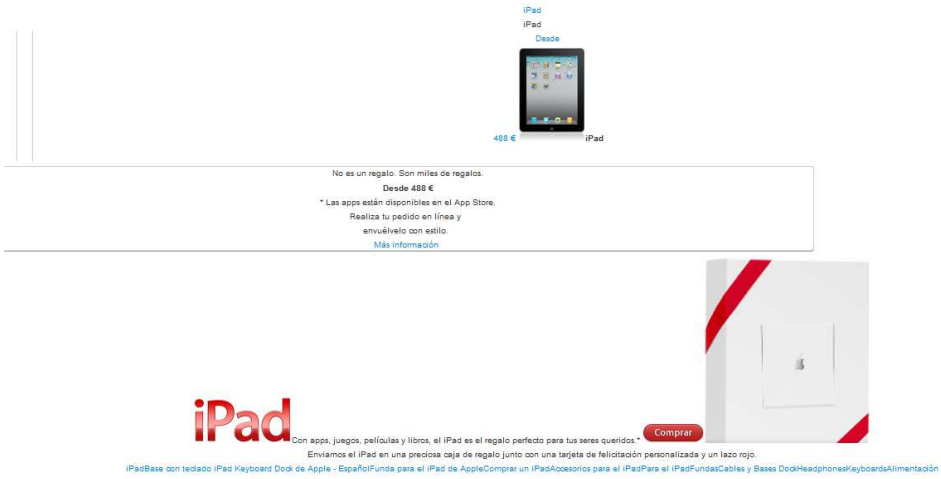


Figura 18. No mantener árbol sobre la Apple Store.

Además de cambiar la visualización del slicing, podemos cambiar el parámetro tolerancia tanto antes como después de la ejecución del slicing. La tolerancia es una ayuda visual para observar mejor los nodos relevantes. La idea es que al aumentar la tolerancia se aumenta el grado de visibilidad desde los nodos relevantes subiendo por el árbol de relaciones. Es decir, si aumenta la visibilidad a 1, se observarán visibles todos los nodos más sus padres directos. Si se aumenta un grado más la visibilidad entonces se observarán los padres de los padres de los nodos relevantes. También se puede disminuir, siempre hasta un valor positivo.

Este tipo de tolerancia sólo se puede realizar respecto a visualizaciones que mantengan el árbol (mantener estructura y colapsar estructura). Por ejemplo, si buscamos con tolerancia 0 la palabra ipad en la store de Apple podríamos ver que sólo veríamos imágenes y alguna palabra suelta con la palabra ipad pero no veríamos adecuadamente todo el contexto. Según fuésemos ampliando la tolerancia veríamos mejor el contexto de los nodos relevantes respecto a la página web.

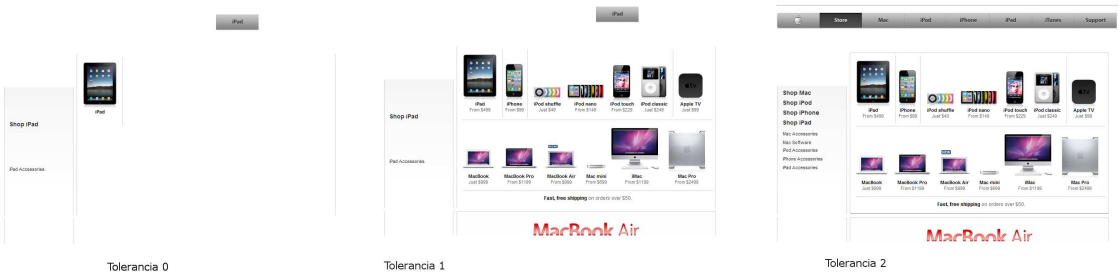
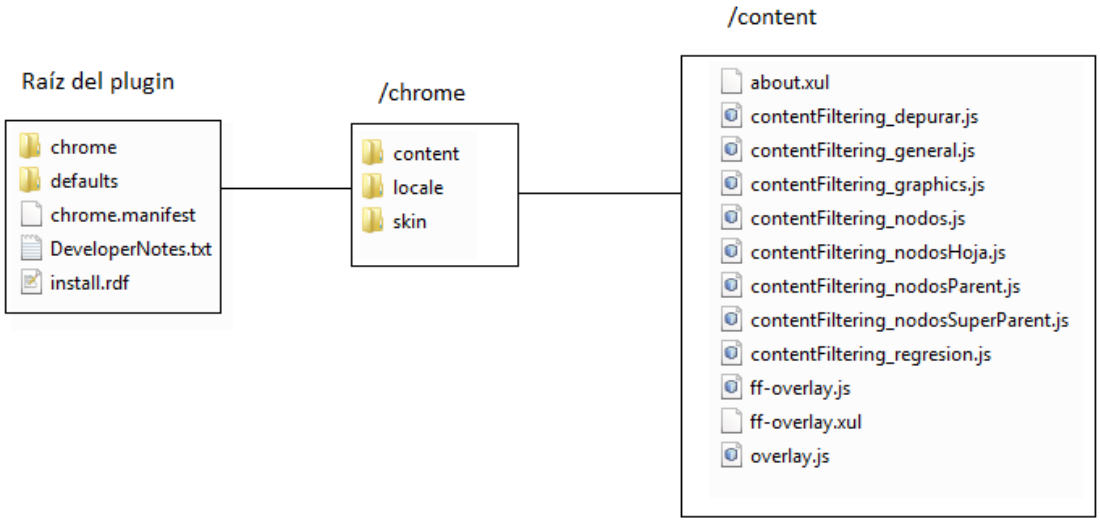


Figura 19. Aplicación de la tolerancia en la página oficial de apple sobre el criterio ipad.

4.5. Estructura y funcionamiento de content filtering

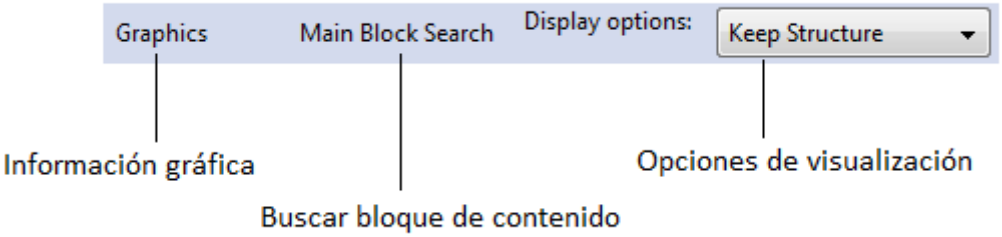
4.5.1. Estructura interna del plugin

El plugin también tiene la misma estructura que el resto de los plugin, sólo que al ser un plugin demasiado reciente sólo tiene un fichero de localización (en-US) y prácticamente todos los ficheros de ejecución se encuentran visibles en la carpeta content.



4.5.2. Instalación del plugin

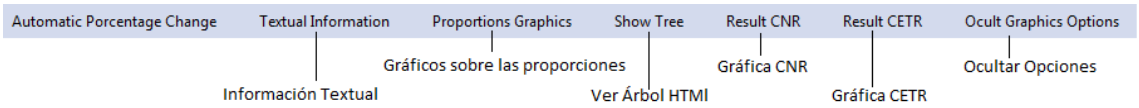
El plugin se puede instalar o bien arrastrando hacia el navegador Firefox o bien instalándolo desde la página oficial de Firefox (de momento se encuentra deshabilitada por Firefox hasta que pase los la revisión oficial). Al instalarse la extensión aparece en la barra superior la siguiente herramienta:



El plugin consta principalmente de las siguientes dos opciones:

- Graphics. Al ejecutarse extrae y muestra de manera visual un resumen sobre la página actual.
- Main Block Search. De manera automática, al ejecutarse busca el nodo que contiene el bloque principal de contenido. Se pueden seleccionar dos modalidades de visualización, keep structure (oculta todo lo que no pertenezca al bloque principal pero sin eliminar relaciones estructurales) y no keep structure (elimina todas la relaciones estructurales colgando del body únicamente el nodo que contiene el bloque principal).

Cuando se ejecuta el botón graphics, el plugin modifica su estructura visual para cambiar las opciones por unas visuales.



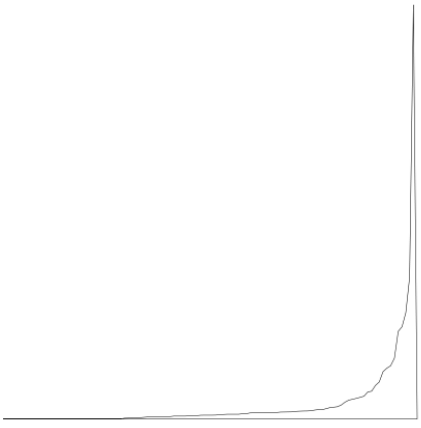
Las opciones de información visual son:

- Textual Information: Muestra la información textual en modo texto de los nodos con las mejores proporciones. La información mostrada es la ruta relativa desde el body hacia el nodo y la proporción.

Elementos totales:109.

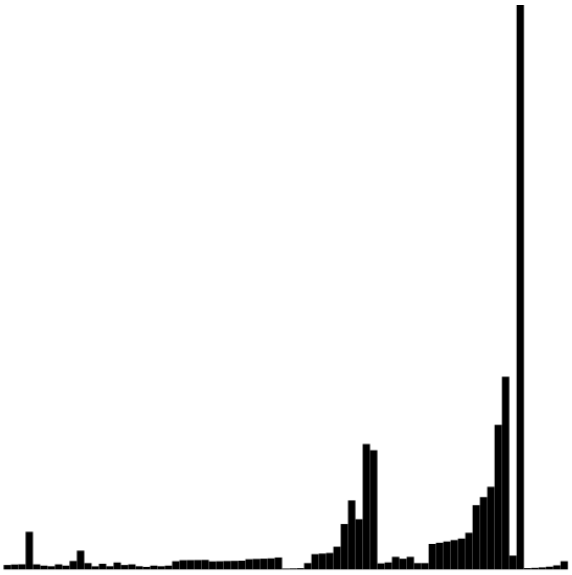
Hacia delante:
El nodo con etiqueta tag:LABEL prop:90 ruta:2-0-0-0-4-0-0-0-0-0-1-1-0-3-0-1
El nodo con etiqueta tag:FONT prop:30.733333333333334 ruta:2-0-0-0-4-0-0-0-0-0-1-1-0-3-0
El nodo con etiqueta tag:TD prop:23.05 ruta:2-0-0-0-4-0-0-0-0-0-1-1-0-3
El nodo con etiqueta tag:LABEL prop:20 ruta:2-0-0-0-2-0-0-0-0-1-0-0-0-1-1-0-0-0-7-5
El nodo con etiqueta tag:LABEL prop:19 ruta:2-0-0-0-2-0-0-0-0-1-0-0-0-1-1-0-0-0-7-8
El nodo con etiqueta tag:TR prop:13.171428571428573 ruta:2-0-0-0-4-0-0-0-0-0-1-1-0
El nodo con etiqueta tag:TBODY prop:11.525 ruta:2-0-0-0-4-0-0-0-0-0-1-1
El nodo con etiqueta tag:LABEL prop:11 ruta:2-0-0-0-2-0-0-0-0-1-0-0-0-1-1-0-0-0-7-0
El nodo con etiqueta tag:TABLE prop:10.244444444444445 ruta:2-0-0-0-4-0-0-0-0-0-1
El nodo con etiqueta tag:LABEL prop:8 ruta:2-0-0-0-2-0-0-0-0-1-0-0-0-1-1-0-0-0-7-2
El nodo con etiqueta tag:FONT prop:7.25 ruta:2-0-0-0-2-0-0-0-0-0-1-0-0-0-1-1-0-0-0-7

- Proportion Graphics: Gráfica que muestra el incremento de los nodos, desde el más pequeño al más grande. Se encuentra escrito internamente en canvas.



- Show Tree: Representación en forma de árbol de todos los nodos (con proporción no nula) donde se refleja la proporción por colores según los nodos.

- Result CNR: Representación aproximada del algoritmo CNR.



- Result CETR: Representación mediante un representación en profundidad de las proporciones obtenidas mediante DOM.

4.5.4. Ejecución del plugin

La ejecución del método es simple. Sólo se tiene que pulsar el botón 'Main Block Search' para que el algoritmo se ejecute y encuentre el bloque principal, teniendo en cuenta que hay que cambiar la visualización según se desee. Por ejemplo, si vamos a la página oficial de filmaffinity (figura 19) y queremos ver el bloque principal del contenido sólo tendrémos que apretar el botón de "Main Block Search".

Figura 20. Filmaffinity.

La única variación depende de dos maneras visuales para mostrar el bloque principal:

- Keep Structure: Se mantiene la estructura del árbol ocultando todos los nodos que no pertenecen al bloque principal (el bloque principal mantiene su posición espacial en la página web).

Figura 21. Filmaffinity con keepStructure.

- No Keep Structure: Se eliminan todos los nodos del árbol dejando colgado del nodo principal solamente el bloque principal (el bloque principal pierde su posición espacial, estando su posición en la parte superior izquierda de la página web).

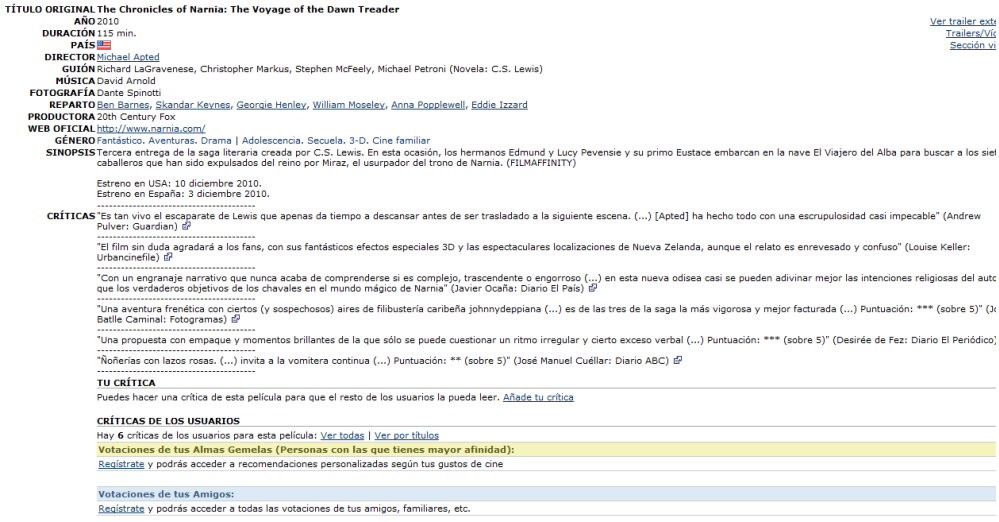


Figura 22. Filmaffinity con No Keep Structure.

Si quisiésemos la información gráfica simplemente hay que apretar, sin haber ejecutado sobre la página ninguna modificación, la opción de graphics. La página se queda en blanco, esperando a que el usuario elija la opción visual que desee de la barra de herramientas.

5. Trabajo Futuro

Se pueden desarrollar diversas mejoras en cada uno de los plugins para mejorar tanto su eficiencia como su variedad de opciones que no fueron introducidas en sus respectivos plugins debido a la falta de tiempo o a su complejidad.

El plugin para Firefox orientado a "Information Retrieval" permite utilizar operaciones booleanas básicas como la operación "AND" y "OR" o la función negación utilizando la opción de filtrado inverso. Se podría aumentar la cantidad de operaciones, permitiendo algunas instrucciones más complejas como la función algorítmica "XOR" o funciones aritméticas como el SUM, MIN o MAX para obligar a que una palabra de búsqueda aparezca un máximo de veces, por encima de una cantidad o un mínimo de veces. Esta variación tendría un coste muy bajo ya que al estar implementado de manera objetual, toda la fuerza de la implementación se ha dividido en métodos. De tal manera que, el método encargado de la evaluación de un nodo (webfiltering_contiene_coincidencia) permite de una manera fácil la manipulación simplemente añadiendo más casos en el *switch* que devuelve un parámetro según evaluación.

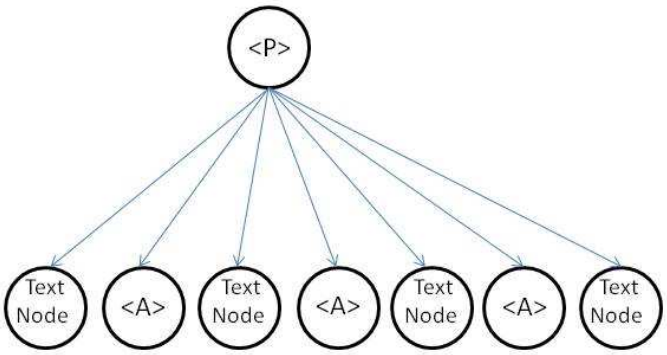
Tras meses de pruebas, observamos en el web filtering slicing que tras realizar un filtrado e ir aumentando la tolerancia, el texto mostrado se incrementaba poco a poco pero en un aumento de tolerancia se producía una gran cantidad de texto mostrado. Es decir, que la tolerancia incrementada no se producía de manera casi proporcional al incremento de la tolerancia sino que se producía de manera desigual pasando de nulos o pequeños incrementos de texto a grandes o totales incrementos de texto. Esto se debe a la estructura interna del DOM, debido a que por regla general las páginas no se encuentran estructuradas de una manera proporcionada y los nodos no están divididos por párrafos o líneas, sino que están divididos por regla general en bloque, como se muestra en la siguiente imagen extraída de la página oficial de la wikipedia en su sección introducción de la página dedicada a tesis:

Texto del apartado introducción.

Derivada del Método científico, una tesis es la aseveración concreta de una idea que, de manera fundamentada, se expone públicamente.

También puede llamársele teoría científica toda vez un sustento teórico puede ser considerado como parte del conocimiento establecido. Hace muchos años, principalmente en el contexto de la Medicina se trataba de una afirmación que el sustentante exponía. Sus ideas se sometían a un interrogatorio, una discusión o prueba dialéctica para sostener en público las posibles objeciones que le oponían los examinadores.

Estructura interna del árbol DOM.



Como se puede observar, la estructura en forma de árbol no está estructurada en ningún caso en párrafos o líneas, ni mucho menos en palabras. Al estar estructurada de esta forma, se produce con pequeñas modificaciones de la tolerancia altos grados de cambios. Lo interesante sería que poco a poco se fuesen viendo las palabras cercanas siguiendo un criterio distinto al criterio de filtrado DOM seguido por el plugin. Para mejorar la tolerancia se proponen las siguientes dos propuestas:

- Tolerancia por cercanía. La idea sería o bien cambiar la estructura DOM por una nuevas estructura que se dividiese visualmente o bien crear una estructura alternativa que estuviera relacionada con la estructura DOM original. La idea sería crear una nueva estructura que estuviese dividida primero por párrafos, posteriormente por frase y finalmente por palabras (figura 23). La idea general se basaría en dividir todo el bloque en párrafos (detectando posibles saltos de línea como la etiqueta br o también se podría ir detectando las etiquetas de tipo listas como las etiquetas UL, LI, OL). Tras detectar los nuevos párrafos dividiríamos las frases en líneas (esto es bastante fácil ya que las líneas se suelen dividir por puntos o puntos y coma). Con este tipo de división hay probabilidad de que fallasen algunas divisiones que de manera visual son obvias pero que requeriría grandes cálculos visuales para dividir las frases. Por último, se dividiría las frases en palabras.

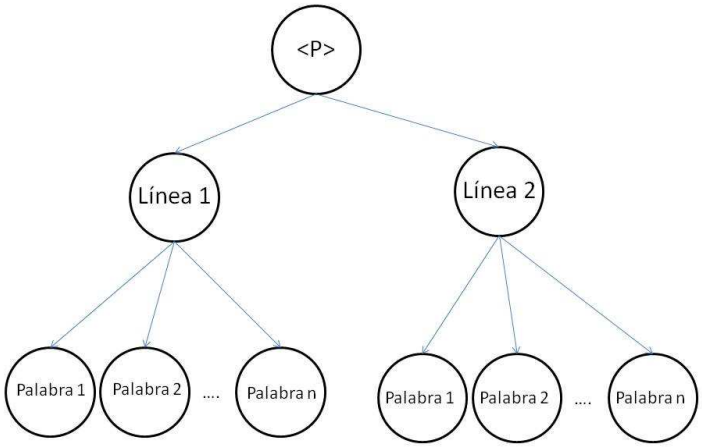


Figura 23. Ejemplo de estructura por semántica.

Tras realizar un filtrado respecto a un criterio se buscarían los nodos que coinciden con la/s palabra/s buscadas y serían marcados en esta nueva estructura. Posteriormente, por cada incremento de la tolerancia, se aumentaría hacia la izquierda y hacia la derecha la visibilidad. Por cada incremento de la tolerancia, se harían visibles nodos más lejanos respecto a la palabra. Cuando el número de palabras hermanas que están en una frase fuesen ya visibles, entonces se harían visibles las frases hermanas a la actual siguiendo el mismo patrón. Finalmente, cuando ya se hiciesen visibles todas las frases entonces se haría visible el correspondiente párrafo y posteriormente todos los párrafos hermanos.

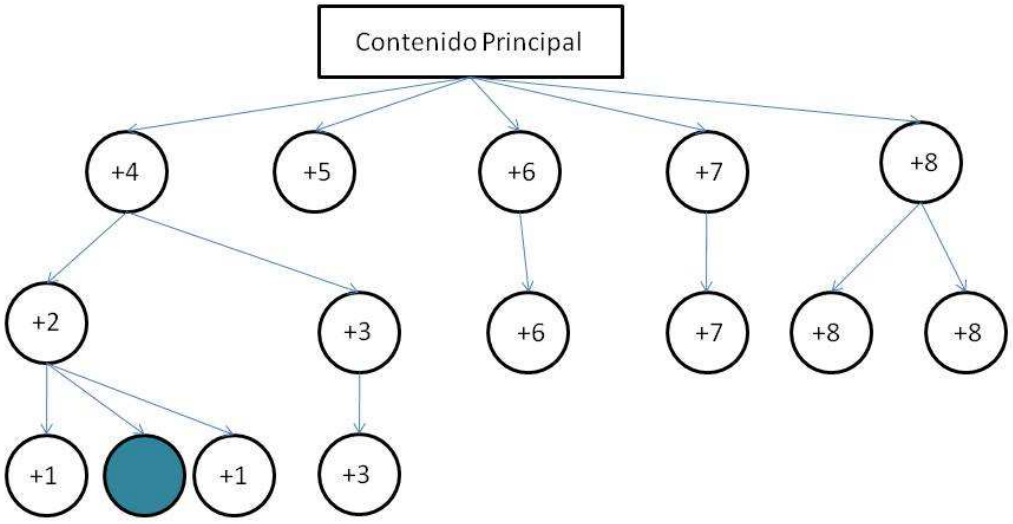


Figura 24. Tolerancia por cercanía.

Cabe destacar que esta técnica sería muy interesante si se le aplica antes la herramienta de búsqueda de un bloque de contenido ya que nos evitaría posible ruido sobre la división del texto. Esta aplicación de la tolerancia posiblemente

tendría un pequeño problema y sería la lentitud de aplicar tolerancias incrementales de +1 si la primera frase que contiene una palabra relevante es una frase con una gran cantidad de palabras.

- Tolerancia semántica. La idea sería aprovechar la tolerancia por cercanía aplicándole también una tolerancia semántica. Se basaría en, tras haber dividido como en el ejemplo anterior todo el bloque contenido en las diferentes unidades, llamar a un proceso que se encargaría de dividir la frase entre palabras con gran importancia semántica (como verbos o nombres) de palabras con nula importancia (como artículos o determinantes). La idea sería marcar primeramente palabras como artículos o adverbios que seguramente no agregan más información relevante. Estas palabras tendrían un coste de cero al aplicarse la tolerancia, es decir, que si al aplicar la tolerancia, uno de sus hermanos es visible entonces dicho nodo también tendría que serlo sin ningún coste. Con esa técnica, nos evitamos la lentitud al aplicar las tolerancias mediante la propuesta anterior cuando tenemos frases muy largas. Además de dicho cambio, tras realizar un filtrado, se generaría mediante un proceso un árbol de proximidad semántica respecto a cualquiera de las palabras introducidas en el criterio. Cuanto más cerca se estuviera de la palabra relevante, menos tolerancia se necesitaría para hacer visible la palabra. Por ejemplo, pensemos que tras dividir la siguiente frase extraída de la wikipedia (sección “inteligencia” de la página oficial sobre perros en la wikipedia):

  > “*Ciertas razas como los ‘Border Collies’ y ‘Golden Retriever’ son por lo común más fáciles de entrenar respecto a otras razas como los perros de caza y de trineo, aunque hay excepciones. Aún el más introvertido, distraído y flojo puede obedecer más fácilmente el entrenamiento que, por ejemplo, un gato.*”

  Si dividiésemos dicho párrafo en frases observaríamos que existen dos frases principales. Para la división y el coste podríamos tener un proceso que se encargase de realizar un análisis sintáctico de la frase y que nos detectase las palabras no relevantes como los artículos y determinantes. La representación gráfica sería como se refleja en la figura 25:

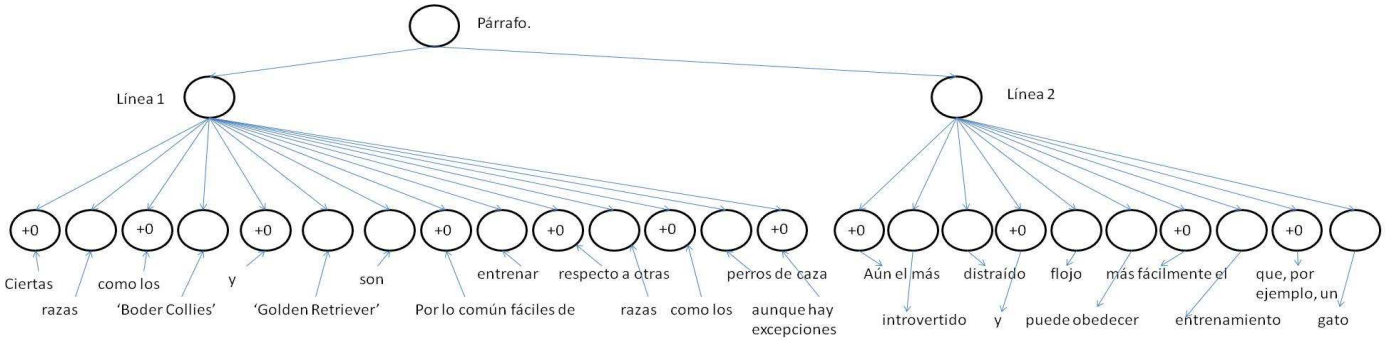


Figura 25. Marcado de palabras con poca importancia semántica.

Como se puede observar, el texto se divide en bloques sintácticos, teniendo valor +0 aquellos bloques que no tienen suficiente información relevante como podrían ser artículos, adverbios o determinantes. Mientras que los verbos, adjetivos y nombres no gozan de un valor para la tolerancia porque necesitan que se les aplique un filtrado para tener un valor respecto a los nodos relevantes.

Si tras obtener la división sintáctica realizamos un filtrado mediante el criterio de búsqueda “gato”, llamaríamos a un servicio que nos devolvería un árbol semántico como el siguiente en forma de xml o json:

```
<criterio valor=”gato”>
        <palabra nivel=”1” valor=”persa“>
                <palabra nivel=”2” valor=”raza” />
                <palabra nivel=”2” valor=”peludo” />
                <palabra nivel=”2” valor=”irak” />
        </palabra >
        <palabra nivel=”1” valor=”deidad”>
                <palabra nivel=”2” valor=”dios” />
                <palabra nivel=”2” valor=”buda” />
        </nivel>
</criterio>
```

Como se observa, cuanto menos bajo fuese el nivel, menor coste de tolerancia sería necesario. En dicho caso observamos que la palabra “raza” se encuentra en un nivel 2 por lo que sería necesario una tolerancia de +2 para ser visible. El resultado de aplicar las tolerancias sería como se muestra en la figura 26 .

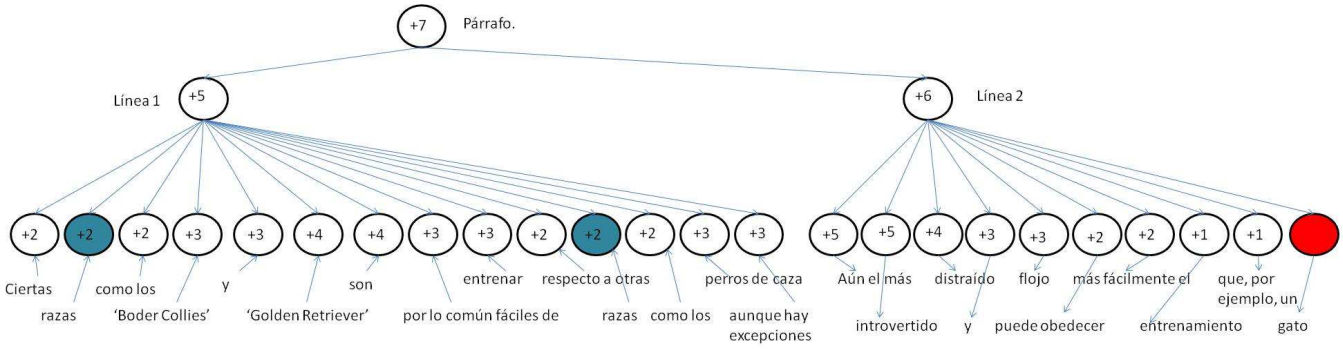


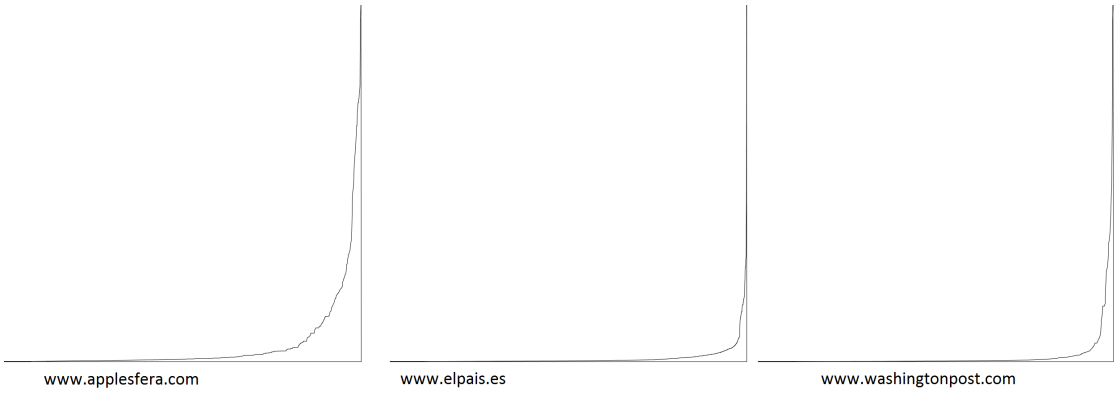
Figura 26. Tolerancia necesario para que el nodo sea visible.

Como se puede observar en el ejemplo, los nodos con fondo rojo representan el nodo relevante en la búsqueda mientras que los nodos de fondo azulado (con una tolerancia de +2 son los nodos con el contenido “razas”). Al comenzar a aplicar la tolerancia se va poco a poco mostrando información cercana al nodo relevante hasta que se llega a tolerancia 2 y se comienza a ver en otra frase información relacionada con razas.

Cabe destacar que las palabras no tendrían porque encontrarse cerca (en el ejemplo es justo la frase siguiente), ya que en el ejemplo sólo hay dos frases por simplicidad. Para implementar esta variación en la tolerancia sería interesante utilizar un proceso web que nos realizase dos funciones. La primera función sería que al pasarle una frase nos devolviese analizada dicha frase de manera sintáctica indicándonos en qué bloques se divide y qué palabras no son sintácticamente de valor. Actualmente existen bastantes analizadores sintácticos que tienen un alto grado de rendimiento (por ejemplo, el VISL [26] de la universidad de Syddansk). La segunda tarea del servicio web externo consistiría en que tras pasarle una palabra nos devolviese el árbol semántico de dicha palabra. Para dicha tarea se necesita tener una gran cantidad de gramática, lo cual nos delimita la posibilidad de introducirlo dentro del plugin debido a su complejidad y a su tamaño físico. Para la gramática se podría utilizar wordnet [27] que es una enorme base de datos léxica del idioma inglés altamente utilizado y de calidad que ocupa aproximadamente 147278 términos estructurados.

El plugin orientado a la búsqueda del contenido principal tiene un alto rendimiento. Destaca porque en su F1 es bastante alto y en prácticamente todos los ejemplos contiene un cobertura cercano al 100% pero una precisión bastante baja. La idea sería parametrizar la búsqueda para que, según el momento, intentar seleccionar entre más precisión o más

cobertura. Si nos fijamos en la gráfica de proporcionalidad de las páginas webs observamos ciertas diferencias entre algunas páginas que nos devolvieron resultados diferentes en los benchmarks (applesfera, elpais y el washingtonPost):



Como se puede observar, aunque leve, se puede ver que su proporcionalidad es distinta, ya que la exponencialidad de la función que engloba en su mayoría a cada una de ellas es muy distinta. Además de la proporcionalidad, las páginas webs varían también en la cantidad de nodos totales:

www.elpais.es: 3015 Nodos.

www.applesfera.com: 1080 Nodos.

www.washingtonpost.com: 1649 Nodos.

La idea principal que se observa es que nuestro algoritmo trata de manera igual a todas las páginas webs cuando tanto la proporcionalidad de toda la página es totalmente distinta como cuando tienen distinta cantidad de nodos. La idea final sería poder elegir maximizar la cobertura o la precisión y, por tanto, se tendrían que utilizar diversos parámetros en nuestro algoritmo. Seguramente, para mejorar la precisión se tendría que coger menos cantidad de bloques de nodos (en nuestro caso se aplica un 10%) y, además, el criterio para unir dos bloques tendría que ser más restrictivo y depender de la cantidad total de nodos. Para obtener unos datos más fiables, sería interesante probar de manera iterativa con todas las páginas utilizadas en los benchmarkings cambiando diversos parámetros para encontrar los parámetros que mejoran la precisión.

6. Conclusiones

En este trabajo nos hemos propuesto como objetivo principal la realización y posterior optimización de un algoritmo que sea capaz de obtener la información relevante dentro una página web a partir de un criterio de búsqueda establecido por el usuario. Esta necesidad de realizar una búsqueda específica dentro de una página surge debido a una incapacidad de los motores de búsqueda actuales (Ask, Bing, Yahoo) de mostrar información concreta tras realizar una búsqueda sobre toda la web. Los actuales motores se enfocan en la búsqueda de las páginas de forma genérica sin prestar la ayuda necesaria para mostrar únicamente la información sobre el criterio de búsqueda, obligando a que sea el usuario quien posteriormente busque la información dentro de cada página la información que le es relevante. Actualmente en Google se están desarrollando herramientas con este objetivo como se pudo observar en su última actualización en Noviembre de 2010, donde fue presentada la herramienta Google Instant Preview, la cual añade una opción que permite una leve visualización del contenido coincidente con el criterio de búsqueda dentro de la página web. Sin embargo, Google Instant Preview es una versión muy inicial que no permite al usuario cambiar la cantidad de texto mostrado o modificar de nuevo el criterio sobre dicha página web. En el algoritmo propuesto para este trabajo nos enfocamos en mejorar esta carencia dando desde un principio todas las posibilidades al usuario para modificar las opciones filtrado.

Tal y como hemos mostrado en el desarrollo de esta memoria, en las primeras versiones de nuestro plugin, el filtrado se enfocaba en la realización de un único filtrado sobre una página web mediante un solo criterio, a través de la experimentación hemos sido capaces de añadir más control al usuario aumentando el número de posibilidades. En primer lugar nos hemos centrado en aumentar la cantidad de palabras posibles que pueden ser utilizadas en el criterio de búsqueda permitiendo utilizar instrucciones booleanas, como el AND y el OR, enriqueciendo con instrucciones de fácil uso que permiten combinaciones con un alto grado de potencia expresiva. Posteriormente, ofrecemos una opción diferente de búsqueda, en vez de mostrar la información según un criterio, mostrar todo lo contrario, es decir, ocultamos la información irrelevante o molesta para el usuario. A la opción de criterio inverso se la ha unido también una opción extra denominada opción de filtrado continuo. Su idea radica en que el plugin realice de manera automática el filtrado según un criterio preestablecido sin necesidad de la intervención del usuario. La idea es que automáticamente, cuando el usuario entra a una página web, o bien sólo muestra la información referente a un criterio (por ejemplo, si estamos navegando en una página deportiva, nos interesara que sólo la información referente a mi club de fútbol) o que oculte cualquier información molesta (como la publicidad, las palabras malsonantes o información sólo apta para mayores de 18 años, pudiendo realizar la función de control parental). Después observamos que tal vez la información mostrada no era del agrado del usuario y se mejoró la opción de la tolerancia permitiendo aumentar o disminuir la información mostrada sin necesidad de filtrar de nuevo la página web. Finalmente, añadimos opciones visuales y estructurales en el filtrado para que el usuario elija si quiere que la visualización mantenga la estructura de la página o si por el contrario desea que toda la información se comprima para ocupar el menor espacio posible.

El algoritmo ha demostrado tener un coste mínimo, el filtrado se realiza sin apenas percepción visual por parte del usuario, y demuestra ser una herramienta muy flexible para el usuario, prueba de ello son los comentarios obtenidos en la página oficial de la herramienta en internet. Tras varios meses de prueba, nos centramos en las páginas web con más tráfico de internet y en este ámbito se puede apreciar que, a pesar de que todas las páginas seguían una estructura basada en HTML, éstas almacenan gran parte de la información en archivos de formatos diferentes al HTML, principalmente debido a que el lenguaje fue creado para la visualización de las páginas en internet y no para el almacenamiento de información. Lamentablemente, las librerías utilizadas están enfocadas a la manipulación de archivos con una estructura interna en XML y no está extendido para la manipulación de otros formatos. Como el algoritmo funciona sin problemas sobre páginas HTML, en vez modificar el algoritmo y las librerías para funcionar con otros formatos, se han adaptado los formatos a la estructura HTML. En primer lugar se ha probado con XML, que tiene un formato fácil de manipular y hemos podido demostrar que fácilmente se podía convertir a HTML pudiendo aplicarle todas las propiedades de nuestra herramienta. Posteriormente probamos con el formato más extendido, los pdf. La conversión conllevó muchas dificultades, principalmente porque estamos hablando de un formato muy complejo y completo que ha ido variando e incrementando su estructura a lo largo de los años. Además, tras obtener las primeras conversiones, observamos que la conversión directa a HTML era difícil debido a que los documentos PDF están orientados a la visualización y no al contenido, era difícil no sólo la extracción del texto sino también la organización del texto para poderle aplicarle el filtrado. Finalmente la herramienta es capaz de realizar un filtrado sobre documentos PDF convertidos en HTML pero que no permite la utilización de la tolerancia para variar la información mostrada, debido al propio formato pdf. Esta limitación se podría mejorar si se implementa una tolerancia semántica o por proximidad que nos permitiría aplicarle la tolerancia.

Tras investigar posibles mejoras en la aplicación de la tolerancia para mejorar la visualización al usuario hemos observado como en algunos ejemplos la tolerancia no se mostraba de la manera deseada. Tras diseccionar dichas pruebas, observamos cómo elementos que forman parte de la página web, como los menús o la publicidad, y que son irrelevantes para el usuario, añaden ruido a la tolerancia mostrando mucha más información de la necesaria por el usuario. En vez de buscar y quitar uno a uno los posibles elementos que distorsionan la tolerancia, nos enfocamos en buscar cuál es el bloque principal de la página web para poder realizar el filtrado sólo sobre dicha página web. Para la realización del algoritmo, hemos mantenido parte de las ideas ya utilizadas en el plugin anterior, utilizando también las mismas librerías que nos permiten acceder y manipular toda la información de una página web.

Partimos de la idea de que la obtención del bloque principal pueda ser integrada o combinada con nuestra herramienta de filtrado. Por esta razón necesitamos que sea un algoritmo rápido y que funcione de forma independiente con cualquier página, por lo que nos vemos en la necesidad de desechar algunas líneas de investigación ya creadas. Desechamos la comparación sobre otras páginas webs para observar estructuras parecidas, que pertenecerían al bloque principal, ya que exigen un coste temporal alto. El algoritmo se enfoca considerando a la página web como una estructura independiente de cualquier otra página y observando la

estructura de la página web. Al contrario que otras investigaciones que extraen la información directamente del código fuente, nuestro algoritmo obtiene la información mediante las librerías DOM que son independientes del código fuente. La utilización de unas librerías independientes del código fuente se debe a que en otras investigaciones se decidió no tratar una página web como una estructura HTML dinámica.

La idea principal del algoritmo se basa en la utilización de un nuevo ratio que denominamos CNR (Content Node Ratio) que nos permitiría valorar a todos los nodos con una valoración utilizando para el cálculo la cantidad de texto asociada a ella dividido por el número de nodos descendientes. La idea principal será seleccionar los nodos que den como resultado un ratio más elevado y buscar el nodo que englobe gran parte de éstos nodos y tenga la mayor parte del texto. Tras realizar varias pruebas variando la cantidad, se observó que seleccionando el 10% de los nodos con mejor ratio obteníamos el resultado idóneo para obtener el bloque principal de texto.

Los primeros resultados mostraron que en la extracción del bloque principal se obtenía, en la gran mayoría de los casos, una cobertura del 100% y una métrica F1 muy cercana al 100%. Aunque sería interesante que la precisión también fuera más alta, hay que tener en cuenta que la funcionalidad de dicha herramienta tiene que ser complementaria con la barra de filtrado web. Es decir, hay un mayor interés en que se incluya información de sobra pero que se quite toda la información irrelevante posible, a que el algoritmo excluya toda la información irrelevante pero quite parte la de información que podría ser relevante al usuario.

La unión de las dos herramientas muestra ser una buena combinación. Por un lado, tenemos una buena herramienta que es capaz de obtener el bloque principal de una página web con coste muy bajo. Con la obtención del texto ya limpio de información irrelevante, podemos utilizar la herramienta de filtrado de información, el cuál ha demostrado ser una herramienta muy completa y flexible en sus criterios cuyo único problema radicaba en el ruido de una página web y de cómo este ruido podía alterar la información mostrada al usuario.

7. Agradecimientos.

Primeramente agradecer a mi director de la tesina, Josep Silva, por ayudarme todo lo posible durante estos dos años, sin tu ayuda seguramente no hubiera enfocado correctamente la investigación en esta tesina. También me gustaría agredecer a mi familia y a todos los compañeros que me han acompañado durante estos tres últimos años de trabajo en el DSIC y que me han ayudado a su manera en la realización de la tesina.

A todos ellos, muchas gracias por la ayuda durante estos años.

8. Referencias

[1] J.M. Gómez Hidalgo, F. Carrero García, E. Puertas Sanz. Named Entity Recognition for Web Content Filtering International Conference on Applications of Natural Language, NLDB2005, pages 286-297, 2005
[2] W3C Consortium, Resource Description Framework (RDF). www.w3.org/RDF
[3] W3C Consortium, Web Ontology Language (OWL). www.w3.oeg/2001/sw/wiki/OWL
[4] Microformats.org. The Official Microformats Site.
[5] R. Khare, T. Çelik Microformats: a Pragmatic Path to the Semantic Web. Proceedings of the 15h International Conference on World Wide Web. International World Wide Web Conference. Poster Sessions pages 865-866, 2006.
[6] R. Khare. Microformats: The Next (Small) Thing on the Semantic Web? I EEE Internet Computing, 10(1):68–75, 2006.
[7] M.Hamammi, Y.Chahir, and L.Chen, WebGuard: A Webfiltering Engine combining textual, structural and visual content-based analysis. IEEE Trans. Knowl .Data Eng., vol.18, no.2, pp.272–284, Feb. 2006.
[8] Anti-Porn Parental Controls Software. Porn Filtering. http://www.tueagles.com/anti-porn/, March 2010
[9] Naomi Internet Filter. http://www.radiance.m6.net/, March 2010.
[10] Po-Ching Li, Mind-Dao Liu, Ying-Dar Lin, Yuang-Cheng Lai Accelerating Web Content Filtering by the Early Decision Algorithm. IEICE – Transactions on Information and Systems vol. E91-D, pages 251-257, 2008
[11] Suhit Gupta, Gail E. Kaiser et al. Automating Content Extraction of HTML Documents. World Wide Archive vol.8 issue.2, pages 179-224, 2005.
[12] W3C Consortium, Document Object Model (DOM). www.w3.org/DOM
[13] Adobe, Portable Document Object (PDF). www.adobe.com/es/product/reader
[14] W3C Consortium, Asynchronous Javascript and XML (AJAX). es.wikipedia.org/wiki/AJAX
[15] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In WWW '05: Special interest tracks and posters of the 14th international onference on World Wide Web. pp. 830–839, 2005.
[16] Xiaoli Li and Bing Liu. Learning to classify text using positive and unlabeled data. In Proceedings of the International Joint Conference on Artificial Intelligence, Acapulco, Mexico, 2003.
[17] J. Arias, K. Deschacht, M.F. Moens. Language independent content extraction from web pages. In Proceedings of the 9th Dutch-Belgian information retrieval workshop, pp. 50-55, The Netherlands, 2009.
[18] B. Krüpl, M. Herzog, and W. Gatterbauer. Using visual cues for extraction of tabular data from arbitrary HTML documents. In Proceedings of the World Wide Web conference, Chiba, Japan, 2005.
[19] T. Gottron. Content code blurring: A new approach to content extraction. In Proceedings of the 19th International Workshop on Database and Expert
[20] Systems Applications, pp. 29-33, Turin, Italy, 2008.
[21] T. Weninger, W.H. Hsu and J. Han. CETR - Content Extraction via Tag Ratios. In Proceedings of the 19th international conference on World Wide Web, pp. 971-980, North Carolina, USA, 2010.

[22] S. Gupta, G. Kaiser, D. Neistadt and P. Grimm. DOM-based content extraction of HTML documents. In Proceedings of the 12th international conference on World Wide Web, pp. 207-214, North Budapest, Hungary, 2003.
[23] Mozilla, Gekko Engine. developer.mozilla.org/en/Gekko.
[24] Google, Google Chrome. www.google.com/chrome
[25] Mozilla, XML User Interface (XUL). developer.mozilla.org/en/XUL
[26] University of Southern Denmark, VISL. beta.visl.sdu.dk
[27] Princeton University, A lexical database for English (Wordnet). wordnet.priceton.edu
[28] J. Silva, Information Filtering and Information Retrieval with the Web Filtering Toolbar. Electronic Notes in Theoretical Computer Science, vol. 235, 125-136, 2008.
[29] F. Finn, N. Kushmerick and B. Smyth. Fact or fiction: Content classification for digital libraries. DELOS-NSF Workshop on Personalisation and Recommender Systems in Digital Libraries, Dublin, 2001.
[30] T. Gottron. Evaluating content extraction on html documents. In Proceedings of the 2nd International Conference on Internet Technologies and Applications, pages 123–132, September 2007.
[31] Po-Ching Li, Mind-Dao Liu, Ying-Dar Lin, Yuang-Cheng Lai. Accelerating Web Content Filtering by the Early Decision Algorithm. IEICE Transactions on Information and Systems vol. E91-D, pages 251-257, 2008.
[32] Micarelli, F. Gasparetti, Adaptative Focused Crawling. The Adaptative Web, pages 231-262, 2007.
[33] Jakob Nielsen. Designing Web Usability: The Practice of Simplicity; New Riders Publishing, Indianapolis ISBN 1-56205-810-X; 2010.