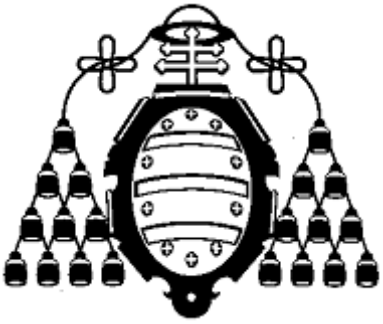


UNIVERSIDAD DE OVIEDO

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

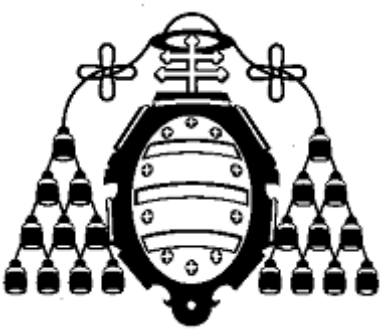
MÁSTER EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

BÚSQUEDA AVANZADA EN APLICACIONES E-COMMERCE A GRAN ESCALA

DAMIÁN GARCÍA GARCÍA

JULIO 2013



UNIVERSIDAD DE OVIEDO

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

BÚSQUEDA AVANZADA EN APLICACIONES E-COMMERCE A GRAN ESCALA

DOCUMENTO Nº 1

MEMORIA

DAMIÁN GARCÍA GARCÍA
JULIO 2013

CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL
TUTOR: ENRIQUE A. DE LA CAL MARÍN

Índice

Lista de figuras

Ilustraciones

Tablas

Gráficas

Abstract

Las aplicaciones de comercio electrónico o *e-commerce* se encuentran inmersas en el triángulo de velocidad, constante actualización de los datos y facilidad de uso. Con el auge del *e-commerce* en los últimos años, los sistemas de comercio manejan millones de productos, cada vez más complejos de escalar correctamente.

El presente trabajo aborda este problema de escalabilidad sobre la funcionalidad de búsqueda de una plataforma de comercio real, que maneja más de 18 millones de productos.

El objetivo del trabajo es mejorar la existente funcionalidad de búsqueda en términos de exactitud, relevancia, actualización de los datos, velocidad y sostenibilidad.

La funcionalidad de búsqueda está compuesta por un servicio de sugerencias en tiempo real que provee nombres de categorías significativos como resultado, basados en la entrada de usuario y un servicio de búsqueda de productos que genera una lista de productos ordenados por relevancia en base a una cadena de búsqueda dada.

Introducción

El presente proyecto, realizado en la empresa *Visual Meta GmbH* con sede en Berlín (Alemania), se basa en la creación de un nuevo sistema de sugerencias para la caja de búsqueda del producto comercial que dicha empresa gestiona, esto es, un sitio web de comercio online que permite la búsqueda de diversos tipos de artículos puestos en venta por tiendas online asociadas.

Este sitio web se encuentra presente en la actualidad en 16 países distintos, y se denomina *ShopAlike* en todos ellos (*www.shopalike.es* para la versión española), salvo en Alemania, sitio web original, y donde claramente se sitúa el mayor mercado del producto actualmente, donde recibe el nombre de *LadenZeile* (*www.ladenzeile.de*).

Debido a dicha mayor cuota de mercado del servicio alemán, pese a que un alto porcentaje del trabajo realizado es común a los 16 sitios web y a sus respectivos idiomas, existen ciertos ajustes concretos que por el momento sólo se han aplicado para el sistema alemán por petición expresa del supervisor interno de la empresa, aunque en el futuro puedan ser ampliados a más países en función de la evolución de sus cuotas de mercado.

El trabajo realizado parte de una implementación anterior ya iniciada, realizada en el lenguaje de programación *Java*, en estado muy básico y no testeada con datos reales, realizada meses atrás por otro empleado de la empresa y detenida abruptamente por la necesidad de desarrollo de otra actividad más urgente para los planes de negocio.

Dicha implementación, se basa en una arquitectura ya existente y funcional de indexación de documentos web del sitio mediante la tecnología *Apache Lucene* para el lenguaje de programación *Java*.

Objetivos

El proyecto a desarrollar debía cumplir una serie de requisitos, de cara a hacer viable su integración en el sitio web:

- Sugerencia de resultados útiles y relacionados con la entrada de usuario.
- Tasa de acierto como mínimo igual a la producida por el sistema actualmente utilizado.
- Rendimiento de tiempo real, similar al del sistema actual, que se adapte a la interacción de los usuarios del sitio web.
- Independencia del lenguaje. Más allá de criterios sintácticos propios de cada idioma como normalización de caracteres específicos, el sistema debería funcionar con todos los lenguajes manejados en la compañía.
- Extensión del sistema para adaptarse a entradas de usuario no regulares:
    - Tolerancia a errores tipográficos o búsqueda *fuzzy*, de forma que términos mal escritos en la entrada de usuario se reconozcan como la palabra intencionada.
    - Independencia del orden de las palabras en la consulta.
    - Incorporación de conocimiento de dominio específico, esto es, adaptar el código a la estructura organizativa interna de cada país, ya que los sistemas alemán e internacional difieren considerablemente, especialmente en lo relativo al género.
    - Adaptación a sinónimos o alias de los géneros manejados por el sistema, por ejemplo la búsqueda de *camisas chica* en lugar de *camisas mujer*.
- Aprovechamiento de la característica ya existente denominada *Perfect match* para la redirección directa a un resultado específico en caso de coincidencia exacta con una etiqueta, utilizada en otras áreas del sistema.
- Realización de *benchmarks* de rendimiento. Todas las pruebas y mediciones a realizar se harán en base a dos conjuntos de casos de prueba suministrados.
- Adicionalmente, debería crearse un *framework* de pruebas *JUnit* que permita comprobar la calidad de los resultados.

Estado del arte

1. Sugerencias

Como una primera y breve introducción al módulo de sugerencias a desarrollar, cabe decir que tiene como objetivo principal proveer una lista de resultados relacionados con la consulta introducida por el usuario en el cuadro de búsqueda situado en la cabecera del sitio web de forma que pueda elegir entre ellos para decidir el resultado que le resulte más adaptado a sus necesidades o simplemente proseguir la búsqueda sin seleccionar ninguno de ellos y que automáticamente le conducirá a la primera sugerencia, considerada la mejor de todas.

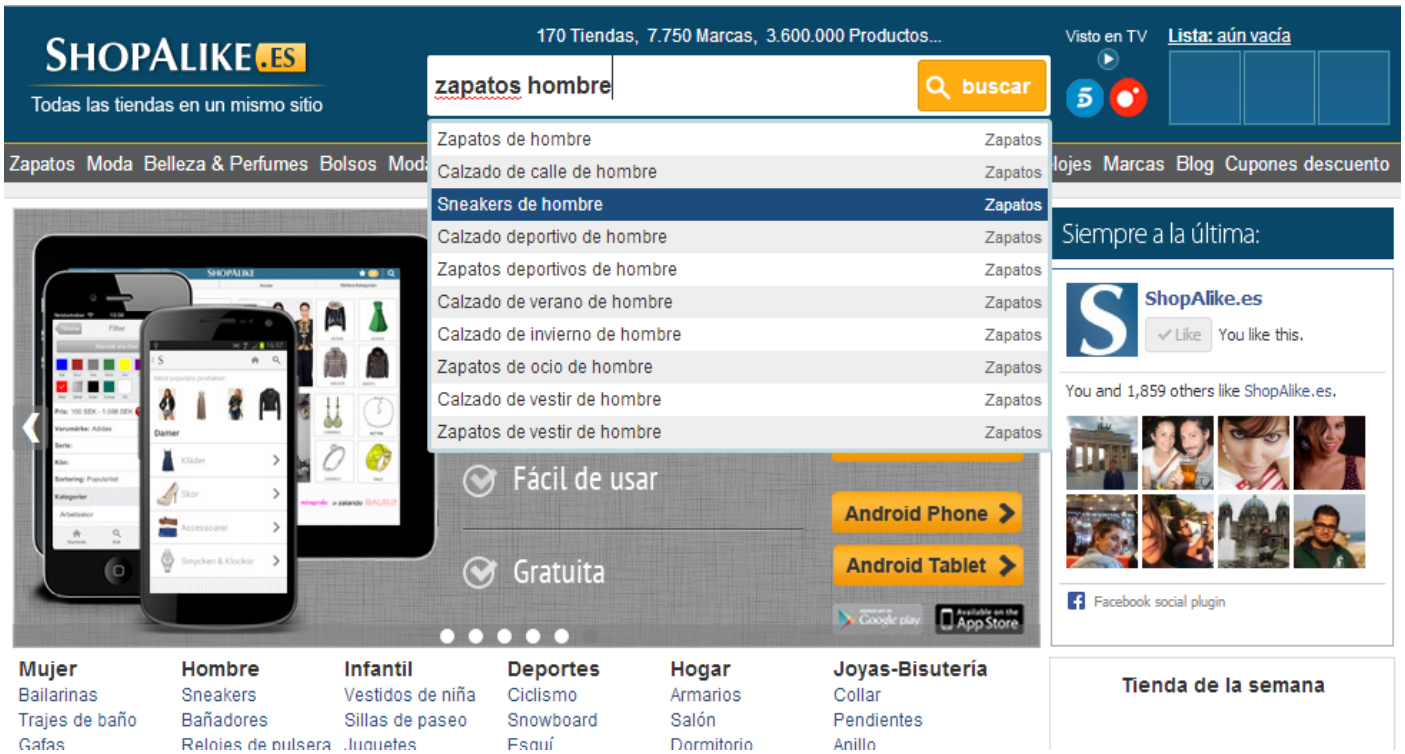


Ilustración 1 - Lista desplegable de sugerencias

2. Sistema de etiquetas

Para comprender el funcionamiento del sistema implementado, es necesario comenzar explicando el esquema básico sobre el que se sustenta todo el sistema.

Este esquema se basa en la organización en *tags* o etiquetas de todos los elementos manejados por el sistema, de forma que cada uno de ellos -desde un determinado modelo de camiseta, hasta la categoría genérica de zapatos, el color azul o el género masculino, por citar algunos ejemplos- se encuentra representado por una de estas etiquetas, las cuales contienen una serie de atributos descriptivos de los mismos.

De entre toda esta lista de atributos cabe destacar dos, de cara a facilitar la comprensión posterior de la presente documentación:

- Un identificador numérico único.
- Un tipo de etiqueta, tal como estilo, marca, género, color, talla... hasta un total de 52 tipos distintos actualmente.

El tipo de etiqueta permite realizar una organización jerárquica de las etiquetas en forma de árbol, el cual es iterado en la búsqueda de un determinado producto.

En realidad, se manejan varios árboles de forma independiente por temas de optimización del diseño, uno para cada una de aquellas categorías principales como puedan ser *Zapatos, Joyas, Muebles, Colores, Géneros*, etc... algunos de los cuales difieren según el país.

Los productos son descritos mediante un conjunto de una o más etiquetas correspondientes a cada uno de los nodos de dicho árbol jerárquico que se recorre en la búsqueda iterativa del producto.

Veamos un ejemplo ilustrativo de este sistema de etiquetas en el que se muestra cómo se obtiene el conjunto de etiquetas descriptores de unas *botas de fútbol con tacos Adidas Predator de color negro*, presentando de antemano los árboles iterados durante el mismo y una breve explicación a continuación:

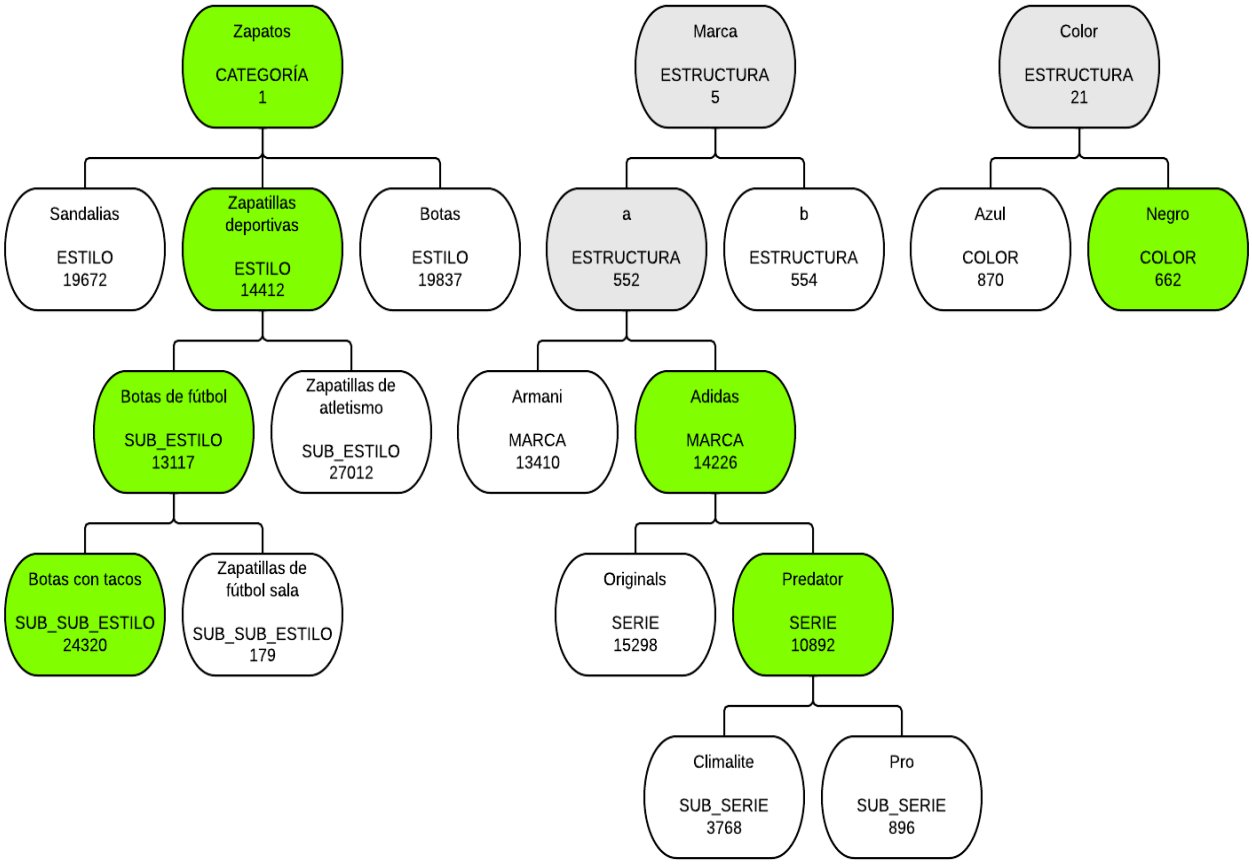


Ilustración 2 - Árbol jerárquico de etiquetas

Recorriendo el árbol de la categoría *Zapatos* desde su raíz, se van registrando las etiquetas de los nodos *Zapatos, Zapatillas deportivas,* y *Botas de fútbol* encontrados en niveles superiores hasta llegar al nodo buscado *Botas con tacos* así como su propia etiqueta. Con esto tendríamos el conjunto de etiquetas [1, 14412, 13117, 24320].

Sin embargo, aún no es completo ya que este conjunto describe todas aquellas *Botas de fútbol con tacos* pero se quiere concretar más y que sean solo las de modelo *Adidas Predator* y no también las de otras marcas.

Por tanto en el árbol de *Marcas*, se recorren los nodos hasta encontrar en este caso la serie *Predator* de la marca *Adidas*, añadiendo las nuevas etiquetas al conjunto que quedaría como [1, 14412, 13117, 24320, 14226, 10892].

Obsérvese aquí que existe un tipo de etiqueta denominado *Estructura* que no se registra en los conjuntos descriptores ya que solo se utiliza para facilitar la organización interna del árbol, pero no aporta nada para describir al producto.

Aún falta un último detalle, y es que no queremos obtener botas rojas o azules, si no tan solo las de color negro. Por tanto se habrá de iterar el árbol de *Colores* y repetir el proceso anterior.

Con ello, obtenemos finalmente el conjunto de etiquetas [1, 14412, 13117, 24320, 14226, 10892, 662] que describen el producto buscado, *botas de fútbol con tacos Adidas Predator de color negro*.

Esta jerarquía es clave en el sistema pues la forma de construir la *URL* de acceso a un producto se basa en ir añadiendo sucesivamente al *path* las *URL* de las categorías bajo las que se encuentra a medida que se recorre el árbol en busca del nodo que contenga el producto buscado.

La siguiente captura muestra de forma más clara este funcionamiento:

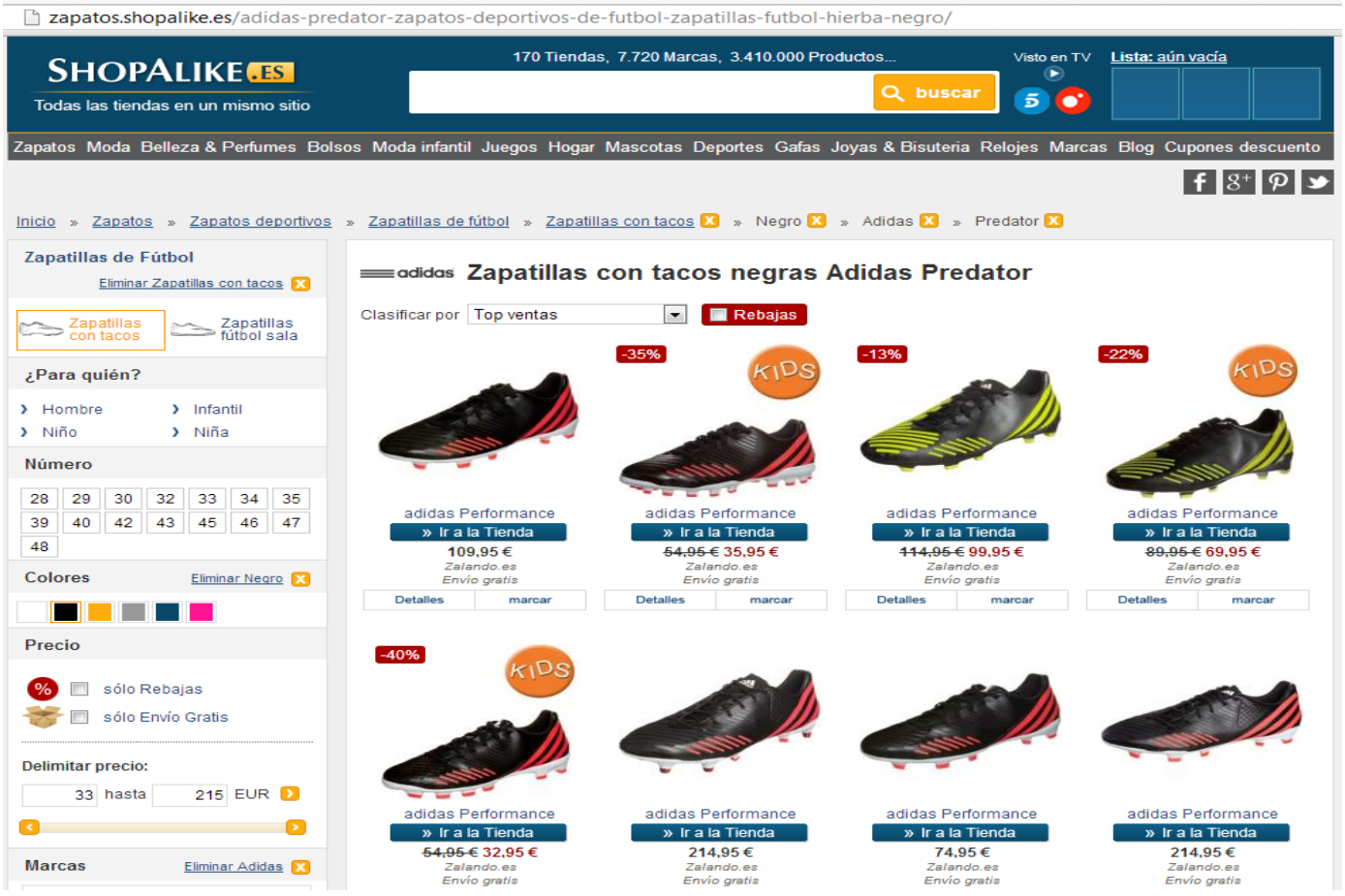


Ilustración 3 - Construcción del path URL

3. Sistema antiguo

El modelo utilizado actualmente en el sitio web para proporcionar sugerencias en cuanto a la entrada del usuario se basa en la estructura de datos denominada *Trie* que permite, tal y como se necesita en este caso, la recuperación de información.

La búsqueda en estas estructuras se realiza mediante la utilización de claves de forma similar a una estructura de tipo de diccionario, aunque su principal diferencia radica en que su estructura es de tipo árbol, almacenando las claves en sus hojas y usando los nodos internos como caminos guía en busca de la solución.

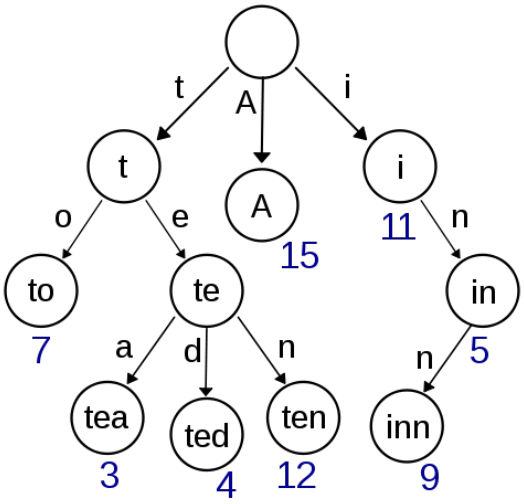


Ilustración 4 - Estructura de datos Trie

A grandes rasgos, ya que este sistema no forma parte del trabajo realizado, se puede describir su funcionamiento como una búsqueda de la entrada del usuario a lo largo de un árbol *Trie* cuyo contenido son las etiquetas de todos los productos existentes en la base de datos, respetando su jerarquía interna.

Este sistema ofrece resultados aceptables ante entradas similares a los nombres de las etiquetas, pero presenta muchos problemas ante cualquier error tipográfico por parte de los usuarios y en bastantes otras situaciones concretas detalladas más adelante.

Por ello, desde la dirección técnica de la compañía se decidió apostar por la implantación de un nuevo sistema que utilizase la ya existente tecnología de *Lucene* y aprovechase todas las ventajas que aporta su motor para mejorar la respuesta del sistema, acercándola lo máximo posible a las necesidades de los usuarios del mismo.

4. Valoración de alternativas

Pese a la existencia de una serie de alternativas posibles a valorar para un desarrollo como el planteado en este proyecto, como puedan ser tecnologías como *Apache Solr* o redes neuronales *WEBSOM* por citar algunas de las más importantes, desde la dirección de la empresa se instó directamente a continuar basando el desarrollo en la tecnología de *Apache Lucene* con el objetivo de aprovechar todo el trabajo ya anteriormente hecho en cuanto a indexación y búsqueda de documentos, sobre el que se basan además otras partes del sistema, pese a las posibles ventajas que pudiesen plantear otras alternativas.

5. Lucene

Lucene es una librería software de código abierto para recuperación de información de alto rendimiento y escalable, distribuida bajo licencia *Apache Software License*, escrita originalmente en *Java*, aunque luego portada para multitud de lenguajes de programación diferentes, que permite añadir capacidades de indexación y búsqueda a cualquier aplicación.

En concreto es capaz de indexar y realizar búsquedas sobre cualquier dato que pueda ser convertido a texto; desde páginas web hasta documentos de texto, pasando por archivos *PDF*.

Consta de dos procesos principales que forman el núcleo de su funcionamiento: la indexación y la búsqueda.

5.1 Indexación

El proceso de indexación consiste en analizar y extraer de entre toda la información disponible, la verdaderamente relevante. Posteriormente, con esa información se crea el índice a partir del cual se realizarán las búsquedas.

El índice es una estructura de datos que permite acceso rápido a la información, algo similar a lo que podría ser el índice de un libro, sin necesidad de escanear todos los documentos de nuevo cada vez que se quiera realizar una búsqueda.

En el proceso de creación del índice, *Lucene* analiza la información separándola en *tokens* y realizando operaciones sobre ellos, como por ejemplo eliminar los artículos o reducir palabras a su raíz.

Una vez analizada la información, ésta está lista para ser añadida al índice, el cual se basa en un índice inverso que permite búsquedas rápidas a partir de una clave (en este caso los *tokens*) con un uso eficiente del espacio en disco.

Veamos a continuación algunas de las principales clases de *Lucene* que intervienen en este proceso, fundamentales para comprender posteriormente la descripción del trabajo realizado, basado en gran medida en su funcionamiento:

5.1.1 IndexWriter

Es el componente central del proceso de indexación. Permite crear un nuevo índice o usar uno ya creado previamente sobre el que añadir nuevos documentos.

Sólo permite operaciones de escritura y borrado en el índice, no siendo posible utilizarlo para lectura o búsqueda. También permite la operación de actualización, aunque esta consta realmente de un borrado y una posterior adición del documento actualizado.

5.1.2 Analizador

El análisis es el proceso de convertir los campos de texto en las unidades de representación del índice. Estos términos son los que posteriormente determinarán qué documentos coinciden con una consulta en una búsqueda.

El analizador se encarga de extraer los *tokens* del texto que van a ser indexados y eliminar el resto, ejecutando operaciones tales como extracción de palabras, supresión de signos de puntuación, acentos o palabras comunes, conversión a minúsculas (normalización), reducción de palabras a la raíz y a formas básicas.

A la hora de implementar un analizador, hay que tener en cuenta numerosos factores como el lenguaje, el dominio o las abreviaturas, por tanto puede afirmarse, y más aún en una aplicación como la nuestra que involucra a multitud de países y lenguajes distintos, que implementar un analizador adecuado es crucial para obtener los resultados deseados.

Puede ser fácilmente extendido, añadiendo nuevos filtros que permitan adaptar la extracción de información a las necesidades de cada aplicación.

En concreto en nuestra aplicación, se complementa con al menos un filtro distinto por idioma manejado, que permiten afinar mucho el proceso de indexación y por ende, la consiguiente búsqueda posterior, ya que las palabras comunes como puedan ser las preposiciones o los sufijos a la hora de reducir una palabra a su raíz varían notablemente entre diferentes lenguajes.

5.1.3 Documento

Representa una colección de campos, que puede ser visto como un documento virtual que se quiere hacer recuperable.

5.1.4 Campo

Corresponden a porciones de información que van a ser requeridas por el índice durante la búsqueda, almacenadas como pares nombre-valor.

Almacenan toda la información del documento, por ejemplo autor, título o identificador, que son indexados y almacenados como campos separados de un documento.

Es posible asignar mayor peso tanto a un campo como a un documento en tiempo de indexación, favoreciendo que posteriormente se posicionen mejor en base a ciertos criterios de búsqueda.

5.2 Búsqueda

El proceso de búsqueda consiste en consultar el índice para obtener los documentos donde aparecen unas determinadas palabras o bien concuerdan con una determinada expresión de consulta.

5.2.1 Puntuación de documentos

Dada una consulta, los documentos obtenidos se clasifican mediante una puntuación otorgada por *Lucene* en base a la siguiente fórmula matemática:

$$puntuación(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{t\ en\ q} \left(tf(t,d) \cdot idf(t)^2 \cdot boost(t) \cdot lengthNorm(t,d)\right)$$

Como se puede apreciar, el cálculo matemático de la puntuación depende de diversos factores. Veámoslos un poco más en detalle:

5.2.1.1 Frecuencia del término buscado en el documento (tf)

Número de apariciones del término en el documento. Aquellos documentos que tienen más ocurrencias de un término dado reciben una puntuación más alta.

5.2.1.2 Frecuencia de documento inversa (idf)

Se corresponde con el inverso de la frecuencia de aparición del término en el índice, lo que significa que términos raros proporcionan una mayor contribución a la puntuación total.

5.2.1.3 Factor de coordenadas (coord)

Número de términos de la consulta que se encuentran en el documento.

5.2.1.4 Normalización en función de la extensión del campo (lengthNorm)

Mide la importancia de un término en base al número total de términos del campo.

5.2.1.5 Boost

Como ya se comentó en el apartado anterior, durante la indexación es posible asignar un mayor peso a ciertos campos o documentos, que por tanto debe influir a la hora de calcular la puntuación.

5.2.1.6 Normalización de la consulta (queryNorm)

No está en sí relacionado con la relevancia del documento, pero se utiliza su valor para permitir hacer comparables las puntuaciones de distintas *queries*.

Como resumen se podría decir que de cara a la puntuación que otorga *Lucene*:

- Los documentos tendrán mayor puntuación a mayor número de términos de búsqueda incluyan.
- Las coincidencias en palabras raras proporcionan mayor puntuación que en palabras comunes.
- Los documentos extensos no tienen tanta puntuación como los cortos a igualdad de términos encontrados.
- La puntuación aumenta a mayor número de menciones de los términos de búsqueda en los documentos.

Las principales clases que forman parte de este proceso son:

5.2.2 IndexSearcher

Permite realizar operaciones de lectura y búsqueda en un índice. De entre todos los métodos de búsqueda que facilita, el utilizado en nuestra aplicación se basa en la búsqueda de consultas o *queries* que producen una lista de coincidencias o hits.

5.2.3 Query

Representa la consulta a realizar sobre el índice, y a su vez se dividen en *Terms,* de tal forma que un *Term* representa a una única palabra.

Existen varios tipos de *query* distintos, aunque los principales son los siguientes:

5.2.3.1 TermQueries

Es el tipo de *query* más simple y representa la búsqueda de un *term* simple.

> Ejemplo: zapatos
>
> Esta *query* representa a la búsqueda definida por la frase *todos los zapatos*.

5.2.3.2 MultiTermQueries

Permite la búsqueda de varios *terms* en una misma *query*, combinando todos los resultados.

> Ejemplo: mujer zapatos
>
> En este caso la *query* representa a la frase *todos los productos para mujer y todos los zapatos*, y obviamente incluiría todos los zapatos de otros géneros como hombre, niño...

5.2.3.3 FuzzyQueries

Permite la búsqueda de términos similares a uno dado, marcándolo al final con el símbolo '∼' y pudiendo especificar la similitud requerida con un valor entre 0 y 1, de tal manera que cuanto más cercano sea este valor a 1 más similares deberán ser los términos.

Ejemplo: bota~0.8

La *query* anterior representa a la frase *todos los objetos cuyo nombre es muy similar a bolso* y será capaz de encontrar términos como bota, botas, bola, bata...

5.2.3.4 BooleanQueries

Lucene permite llevar a cabo búsquedas complejas separando los términos de búsqueda en las consultas mediante los operadores lógicos OR, AND y NOT, o sus símbolos equivalentes (+ para AND, - para NOT, y en caso de ausencia de operador lógico o simbólico la consideración como OR).

Ejemplo: (zapatillas OR botas) AND (mujer) NOT rojo

+(zapatillas botas) +mujer -rojo

Ambas queries son equivalentes, como ya se ha explicado sustituyendo los operadores lógicos por sus símbolos correspondientes, y representan la búsqueda correspondiente a la frase *todas las zapatillas o botas de mujer que no sean de color rojo*.

Además, se puede formar una BooleanQuery a partir de combinaciones de TermQueries y FuzzyQueries mediante operadores lógicos.

Ejemplo: +(zapatillas bota~0.8) +mujer -rojo

En este último caso, la *query* se corresponde con la frase *todas las zapatillas y objetos de nombre similar a bota de mujer que no sean de color rojo*, y entre los resultados posibles que podría encontrar además de zapatos verdes de mujer, podría haber tanto batas como botas de mujer amarillas.

Como cabe imaginar tras ver estos ejemplos, este amplio abanico de representaciones de *query* que ofrece *Lucene* proporciona un gran potencial en cuanto a la realización de búsquedas en el índice.

5.2.4 Hits

Representa la lista de coincidencias encontradas en el índice respecto a una determinada consulta realizada.

5.3 Arquitectura típica de sistemas de indexación y búsqueda basados en Lucene

La arquitectura típica de implementación del sistema de indexación y búsqueda basado en *Lucene* se correspondería con el siguiente diagrama:

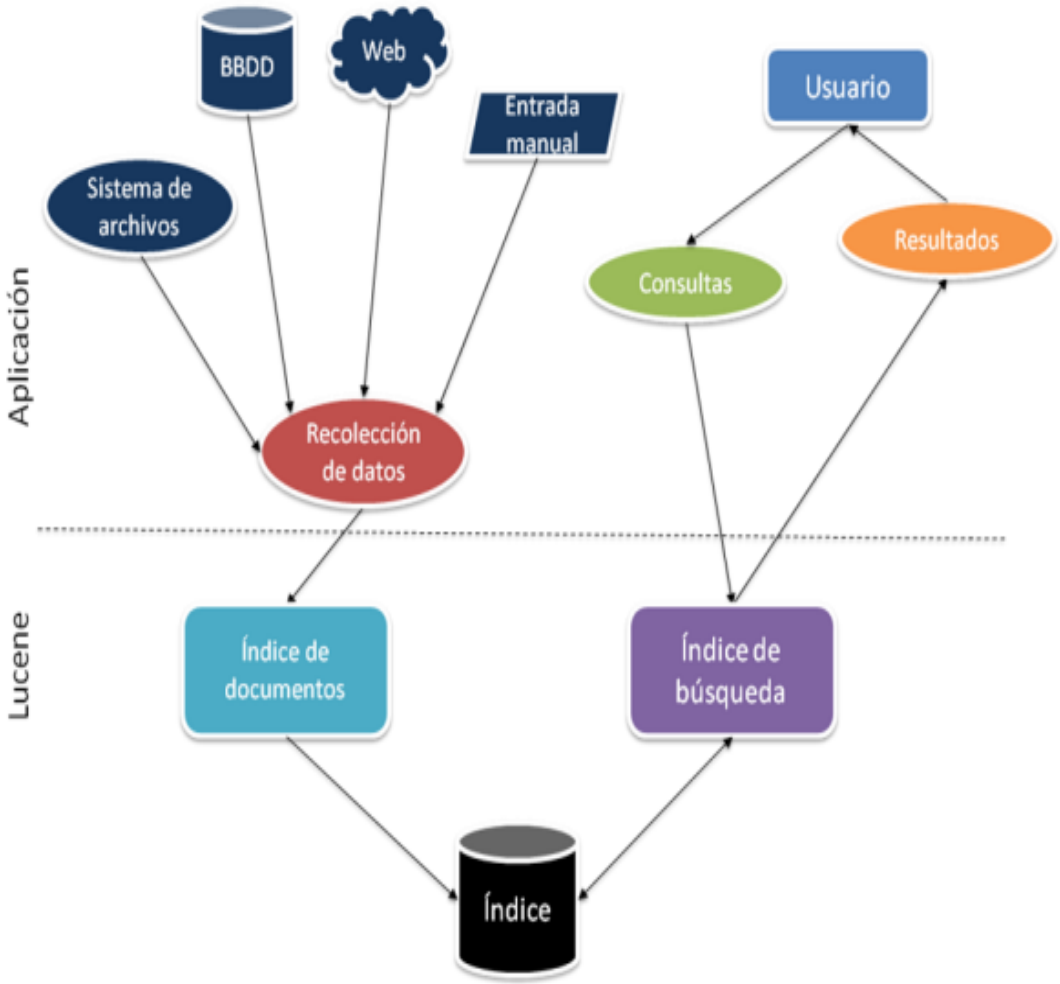


Ilustración 5 - Arquitectura típica de indexación y búsqueda

Como se puede observar a partir de la recolección de datos de diversas fuentes se crea un índice de documentos, sobre el que posteriormente pueden ejecutarse búsquedas de consultas requeridas por los usuarios que reciben los resultados correspondientes a ellas.

6. Estado del nuevo sistema al inicio del desarrollo del proyecto

Como ya se ha comentado, el nuevo sistema de sugerencias basado en *Lucene* se detuvo al poco tiempo de comenzar su desarrollo por temas de negocio, y en el momento en que comenzó el presente proyecto se encontraba en un estado prácticamente embrionario, en el que el esqueleto básico de realización de búsquedas y presentación de una lista de sugerencias se encontraba ya realizado pero no estaba adaptado para entradas de usuario mínimamente complejas, sirviendo de ejemplo su baja tasa de acierto, en torno al 30%.

Sin embargo, toda la arquitectura de indexación y búsqueda sobre la que añadir el módulo de sugerencias a desarrollar ya se encontraba completamente implementada.

6.1 Arquitectura del sistema de indexación y búsqueda

Centrándonos en el sistema concreto de los sitios web de la empresa, la implementación existente al inicio del proyecto se basa en un diseño adaptado al diseño general de todo el sistema, con una abstracción del software mediante la división del mismo en una parte de *backend*, encargada de la indexación y gestión de documentos, y una parte de *frontend* responsable de llevar a cabo las búsquedas de documentos.

La parte *backend* consta de un controlador que obtiene todos los documentos indexables almacenados en el servidor de caché, los cuales son registrados en el índice por un proceso indexador.

Por su parte, la parte *frontend* está formada por dos componentes que implementan un analizador de los datos recogidos por un interfaz de usuario a partir de una determinada entrada de usuario, y un *parser* que realiza la búsqueda en el índice de documentos anteriormente creado en busca de los documentos solicitados.

Esta arquitectura sobre la cual partió el trabajo llevado a cabo en este proyecto se corresponde con el siguiente diagrama:

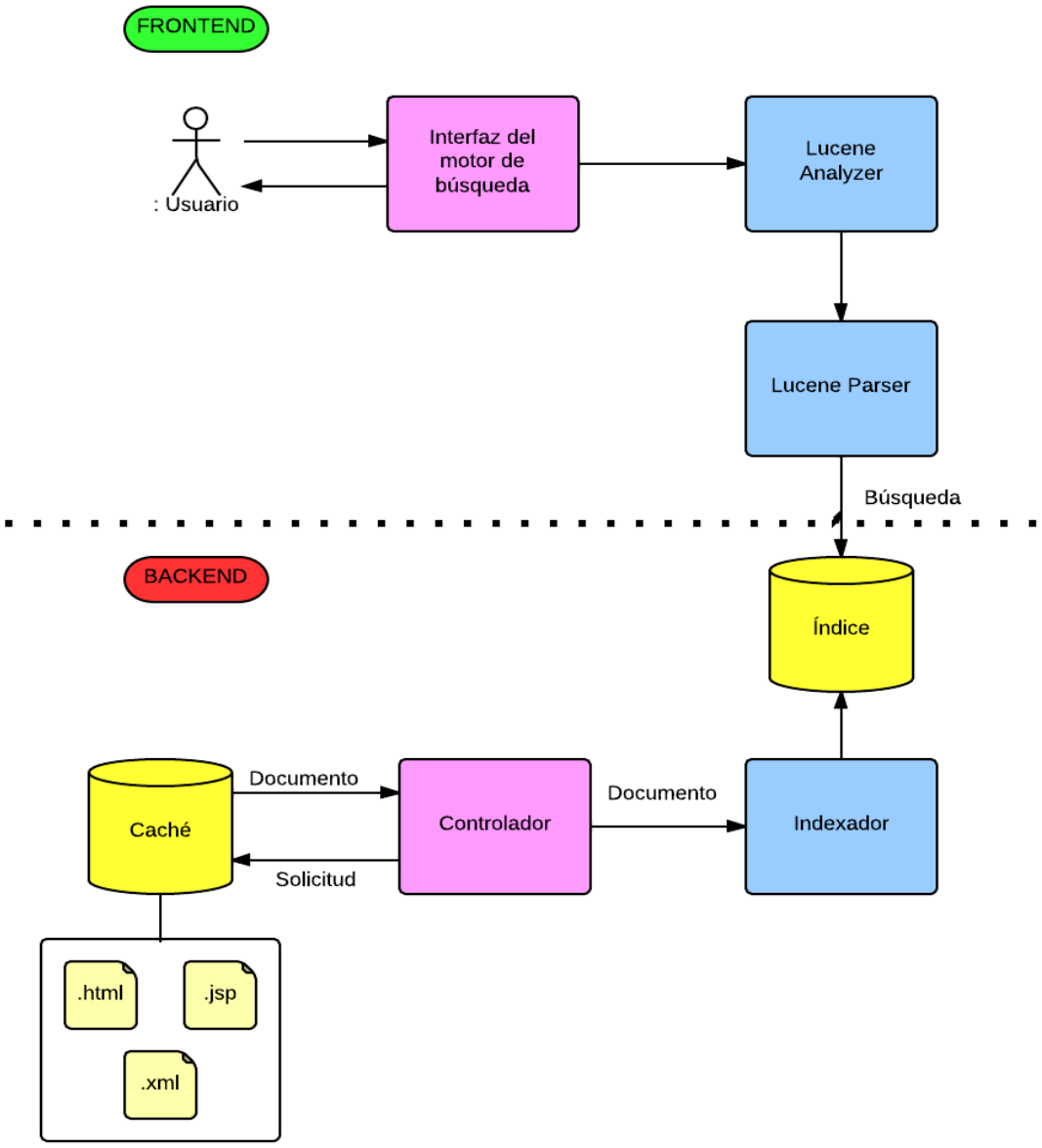


Ilustración 6 - Arquitectura de indexación y búsqueda

Propuesta

1. Arquitectura

El nuevo sistema se implementó como una nueva capa sobre *Lucene* que calcula y retorna una lista de sugerencias en base a las consultas de los usuarios.

Para ello se partió de la ya existente y previamente vista arquitectura de indexación y búsqueda de *Lucene*, añadiendo capacidades de análisis sobre los documentos del índice de productos:

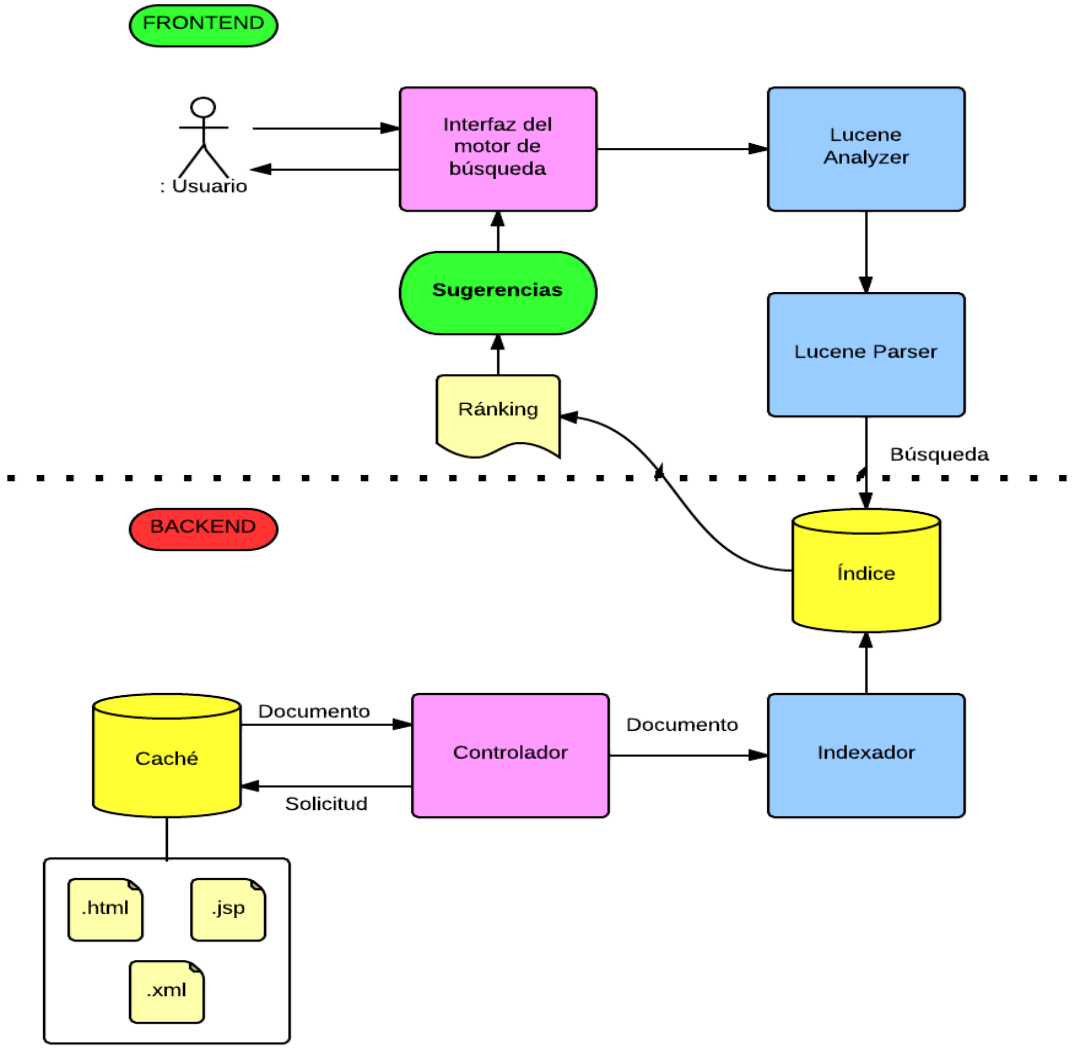


Ilustración 7 - Arquitectura del sistema implementado

2. Medidas tomadas

En este capítulo se listan las medidas tomadas para la implementación de la propuesta realizada de desarrollo del sistema de sugerencias, estructurada en los siguientes pasos:

- Reorganización
- Ajuste del sistema de puntuación de documentos de Lucene
- Promoción de documentos
- Perfect match
- Solución de errores en Lucene

2.1 Reorganización

Tras un período inicial de análisis del estado en que se encontraba el sistema, se observó que para un porcentaje grande de ejemplos de los conjuntos de prueba, éste encontraba el documento esperado, aunque no lo consideraba la mejor sugerencia en función de la entrada introducida, si no que otorgaba mayor puntuación a otros documentos, relacionados o no con el objeto buscado. Esta situación era especialmente frecuente en las entradas de usuario no regulares

Debido a ello, como primer paso a tomar se planteó llevar a cabo una reorganización del ranking de documentos calculado por *Lucene*, recalculando la puntuación de los documentos en función de otras variables del sistema, fuera del ámbito de acción del motor de *Lucene*.

Buscando un buen balance entre tiempo de respuesta y tasa de acierto, se optó por no aplicar la reorganización a todos los documentos encontrados por *Lucene*, si no que se fijó un umbral igual al número de sugerencias mostrado en el menú desplegable de la caja de búsqueda del sitio web, que en la actualidad es de 10 sugerencias.

Se asumió por tanto, que en aquellos casos en que la sugerencia óptima no se encontrase con este proceso, la causa vendría de otra parte del sistema por lo que no tenía sentido lastrar más el tiempo de respuesta del sistema de sugerencias en su búsqueda a base de ajustar más la reorganización, pasando a buscar nuevas soluciones y ajustes para dichos casos.

Posteriormente se llevó a cabo un nuevo reajuste, permitiendo la ampliación de este umbral en aquellos casos en que documentos posteriores al que marcaba el umbral, tuviesen la misma puntuación otorgada por *Lucene* que éste último, permitiendo encontrar la sugerencia óptima en algunos casos concretos en los que en caso contrario no se hubiera podido hallar sin esta ampliación.

Sin más dilación, veamos a continuación las consideraciones tomadas para llevar a cabo esta reorganización, estructuradas de la siguiente forma:

- Criterio del SortSum
- Criterio de similitud
- Permutación recursiva de elementos sin repeticiones
- Ordenación vectorial

2.1.1 Criterio del SortSum

En muchos de estos resultados anómalos, ciertos productos de marcas comerciales conocidas e importantes, pese a encontrarse entre las primeras sugerencias, quedaban por detrás de otros de marcas poco relevantes.

A raíz de esta situación, la primera medida tomada para reorganizar las sugerencias se basó en calcular la nueva puntuación de los documentos combinando la puntuación obtenida por *Lucene* con el valor de *SortSum* del conjunto de etiquetas descriptivas de los documentos.

SortSum es un valor calculado y actualizado de forma automática con periodicidad semanal por el sistema, que asigna un valor numérico a cada conjunto de etiquetas descriptor de una *URL* (formado pues por no más de una etiqueta de cada tipo), en función del número de visitas, el número de clics de ratón o el número de enlaces a ella dentro de los sitios web, entre otros múltiples factores.

Este valor se utiliza para varios fines distintos dentro del sistema, por ejemplo para la ordenación de los objetos mostrados como resultado de una búsqueda, pues define cuantitativamente la importancia de los productos.

Por tanto, su uso para alterar la puntuación de los documentos sugeridos permitiría situar primero a aquellos documentos de entre los sugeridos por *Lucene* que fuesen considerados más importantes en el momento de la búsqueda, lo cual encajaba directamente con el comportamiento esperado en un buen número de casos.

Una vez decidido que el *SortSum* debería ser tenido en cuenta en el cálculo de puntuaciones, quedaba otro aspecto clave por definir, que no era otro que en qué medida debería hacerlo.

Debido a que el valor de *SortSum* no se encuentra contenido en ningún rango fijo de valores, como paso previo a ello se realiza una normalización.

Tras realizar diversas pruebas con diferentes combinaciones, se observó que los mejores resultados calculando la nueva puntuación en base a la siguiente fórmula:

$$puntuación_{modificada} = puntuación_{Lucene} \cdot SortSum_{normalizado}$$

$$con\ SortSum_{normalizado}\ \epsilon\ [1,2]$$

Esta primera alteración de la puntuación de los documentos sugeridos, permitió aumentar notablemente la tasa de acierto entre el conjunto de prueba de ejemplos *fuzzy*.

2.1.2 Criterio de similitud

A pesar de que la solución recién expuesta mejoró significativamente los aciertos, en ciertos casos aún no era suficiente, ya que no siempre deberían pesar más siempre las marcas comerciales más importantes o los productos más adquiridos en el reordenamiento de las sugerencias, sino que también se debería ponderar cierta similitud a la entrada del usuario, pues podría darse la situación de que la sugerencia esperada no coincidiese con el producto más importante.

Veámoslo con un ejemplo real concreto:

Un determinado usuario, desea buscar productos de la marca de gafas *Goggle*, y confundido por el nombre de un conocido buscador teclea *google* en la caja de búsqueda. Si bien *goggle* es un término inglés para referirse a gafas, con muy poca probabilidad se utilizaría en una búsqueda genérica de gafas, en detrimento del término mucho más utilizado globalmente de *glasses*, con lo que cabe pensar que con casi toda seguridad el usuario querrá buscar la marca comercial así denominada.

El sistema de sugerencias detecta que es una búsqueda *fuzzy*, pues no tiene equivalencia directa con una etiqueta y busca en el índice términos similares a *google* como son *goggle* (la marca que se considera sugerencia óptima) y *goggles* (gafas genéricamente).

En base a ello *Lucene* devuelve una lista ordenada de resultados que incluye entre los primeros puestos dos documentos relativos a la sugerencia esperada *Goggle* y a otra marca de gafas, en este caso líder en el sector y por tanto muy importante, como es *Oakley*, clasificados ambos con la misma puntuación.

El recién descrito método de reordenación en base al *SortSum* colocaría por delante como primera sugerencia a *Oakley*, seguido de *Goggle* pues aún normalizando los valores la diferencia entre ambos es considerablemente grande, resultado inverso al esperado.

Para solucionar esto, se realizó un nuevo ajuste sobre la fórmula de cálculo de la puntuación, teniendo en cuenta la similitud del nombre de las etiquetas de los documentos con la entrada del usuario.

De entre la gran variedad de algoritmos de cálculo de similitud o distancia entre cadenas de texto, se decidió utilizar la distancia de *Levenshtein* debido a que diversos estudios [*] prueban que es la que mejor comportamiento presenta ante errores ortográficos y tipográficos. Además permitiría mantener cierta cohesión dentro del sistema ya que el propio motor de *Lucene* implementa una versión propia de esta distancia durante la búsqueda de documentos a partir de una *query* de tipo *fuzzy*.

2.1.2.1 Distancia de Levenshtein

La distancia de *Levenshtein* o distancia de edición se define como el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Estas operaciones pueden ser la inserción, el borrado o la sustitución de un carácter por otro.

Así por ejemplo la distancia de *Levenshtein* entre la entrada de usuario y las dos mejores sugerencias obtenidas en el ejemplo anterior serían:

$$distancia_{Levenshtein}(google, oakley) = 6$$

(Sustitución de los 6 caracteres de la palabra)

$$distancia_{Levenshtein}(google, goggle) = 1$$

(Sustitución de la 'o' en tercera posición por 'g')

Al igual que en el caso anterior, se realizaron diversas pruebas en busca de la máxima optimización de la fórmula utilizada para recalcular las puntuaciones, hasta fijarla en:

$$puntuación_{modificada} = \frac{puntuación_{Lucene} \cdot SortSum_{normalizado}}{distancia_{Levenshtein}(entrada, sugerencia)}$$

$$con\ SortSum_{normalizado} \in [1,2]$$

2.1.2.2 Stopwords

En computación, *stopwords* o palabras de parada son palabras que son filtradas del resto en el momento del procesamiento textos.

Cualquier grupo de palabras puede ser elegido como *stopwords* para un cierto propósito. Algunas palabras, tales como artículos o preposiciones, son comunes a casi todos los sistemas de búsqueda, y algunos además eliminan palabras como verbos de las consultas en búsqueda de un mayor rendimiento.

La lista de palabras a excluir debe ser analizada detalladamente pues puede influir mucho en la búsqueda de resultados y debería ser realizada por un experto en lingüística o como mínimo por una persona cuya lengua materna sea la analizada, en este caso el alemán.

En el sistema ya existía una entrada en la base de datos con una lista de *stopwords*, utilizada durante la creación del índice.

Sin embargo, en ningún momento era utilizada para filtrar las entradas de los usuarios, lo cual podría parecer lógico en un principio, e incluso podría haberse solucionado de otra forma distinta, si no fuera porque con el nuevo diseño que utiliza la similitud se convierte en indispensable ya que carecía de sentido comparar una cadena que contiene *stopwords* (entrada de usuario) con otra en la que estas palabras se han eliminado (nombre del documento indexado).

2.1.2.3 Normalización de caracteres

En el idioma alemán se utilizan ciertos caracteres especiales que deben ser normalizados:

- Beta (*ß*), pese a que su uso está decayendo y oficialmente ya se acepta su interpretación como una *s* doble (*ss*).
- *Umlauts* o diéresis, sobre algunas vocales (*ä*, *ö*, *ü*), en cuyo caso muchos usuarios optan, al igual que sucede con las tildes en castellano, por su omisión, sustituyéndolos por la representación fonética (*ae*, *oe*, *ue*).

Para garantizar una mayor cohesión entre los datos indexados y las posibles entradas de usuario omitiendo la escritura de dichos símbolos, se optó por sustituirlos por sus representaciones alternativas tanto en fase de creación del índice como en la de comparativa de similitud de la entrada de usuario.

2.1.3 Permutación recursiva de elementos sin repeticiones

La medición de distancia entre dos cadenas tal y como acaba de describirse presenta un problema, imaginemos que el usuario realiza la búsqueda de *camisetas adidas*, y pese a que en el sistema se encuentra las etiquetas de *Camisetas* y *Adidas*, al unir sus nombres en la cadena a comparar con la actual esta resulta siendo *adidas camisetas* debido al orden de iteración sobre las etiquetas.

En este ejemplo, la distancia entre ambas cadenas calculada con el método recién descrito sería 14 y así mismo consideraría el término *camisas* mucho más similar, a distancia 4, asignándoles mucho más peso.

Ante este problema y la imposibilidad de prevenir y alterar el orden de alteración de las etiquetas de un conjunto a la hora de crear un nombre descriptor del mismo, la solución más óptima pasó por implementar un método que dado un conjunto de etiquetas, crease una lista con todas las posibles permutaciones de las etiquetas sin repeticiones.

Dicho método es de *O(n)* para n elementos pero se consideró asumible debido a que los posibles conjuntos de etiquetas manejados en el sistema no superarían en ningún caso la decena de elementos por lo que realmente no repercutirían demasiado en el rendimiento del sistema.

La distancia entre la consulta realizada y el conjunto de etiquetas será por tanto la menor de las distancias entre el término buscado en la consulta y cada una de las posibles permutaciones de las etiquetas descriptoras del conjunto.

2.1.4 Ordenación vectorial

Una vez concluidos todos los cálculos de ajuste de las puntuaciones de los documentos sugeridos, el paso restante consiste en ordenarlos nuevamente en base a dichos valores de forma que aparezcan en el orden correcto en el menú desplegable mostrado al usuario en el sitio web.

Dado el bajo número de sugerencias que por el momento se analizan a posteriori de entre las encontradas por *Lucene*, el tiempo de reordenación de las mismas no sería un aspecto crítico, pero en una decisión de diseño de cara a facilitar la escalabilidad en posibles futuras modificaciones del proyecto, en que quizás se decidiese aumentar el número de sugerencias a analizar, se optó por implementar un algoritmo de ordenación vectorial lo más eficiente y rápido posible que evite pérdidas de rendimiento en esa hipotética situación.

El elegido, por tanto, fue el conocido algoritmo de ordenación *Quicksort*, basado en la técnica de *divide y vencerás*, y que permite la ordenación, en promedio, en orden *O(n·log n).*

Su funcionamiento se basa en la elección de un elemento del vector a ordenar, denominado pivote, en base al cual:

- Se resitúan el resto de elementos del vector a su lado, de manera que a un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- El vector se divide en dos subvectores formados por los valores a izquierda y derecha del pivote respectivamente.
- El proceso se repite recursivamente para cada subvector mientras éstos contengan más de un elemento. Una vez terminado el proceso todos los elementos estarán ordenados.

Veamos el funcionamiento del algoritmo con un vector de ejemplo:

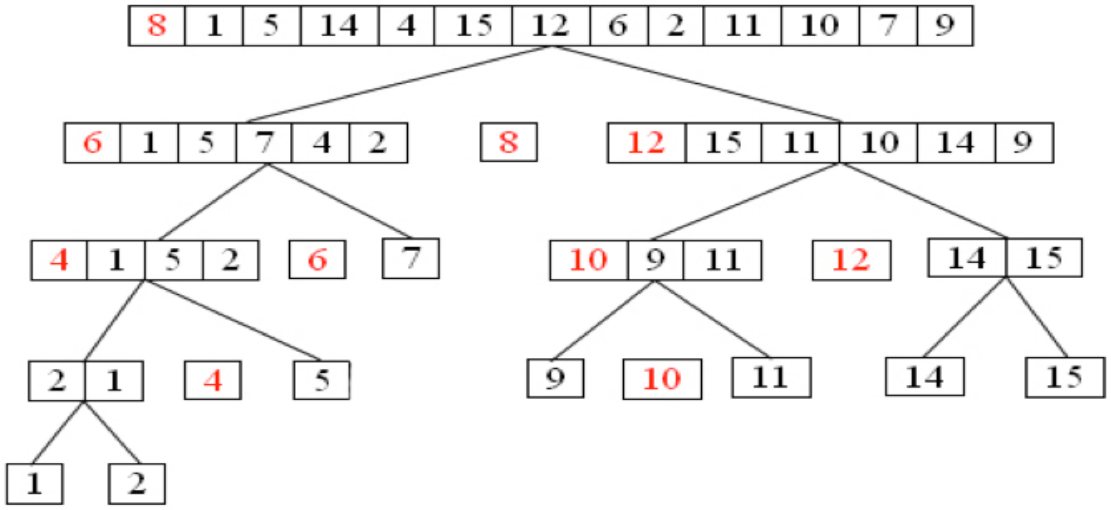


Ilustración 8 - Algoritmo de ordenación vectorial

Su rendimiento puede ser todavía mejorado en caso de necesidad futura con técnicas de reposicionamiento del pivote sobre el que se basa pero no se consideró primordial en el momento en que se implementó en el sistema dado el volumen de datos manejado.

2.2 Ajuste del sistema de puntuación de documentos de Lucene

Como ya se ha comentado previamente, *Lucene* crea el ranking de documentos en base a la siguiente fórmula matemática:

$$puntuación(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{t\ en\ q} \left(tf(t,d) \cdot idf(t)^2 \cdot boost(t) \cdot lengthNorm(t,d) \right)$$

Esta implementación por defecto suele funcionar bien en la mayoría de sistemas y casos, también en el presente, pero pese a ello, en vista del tipo de documentos manejados, se consideró necesario realizar un ajuste en algunos de los componentes que la definen, optimizando el cálculo de la puntuación y el consiguiente ranking de documentos posterior más acorde a las necesidades del sistema.

En concreto se optó por anular dos factores: la frecuencia del término buscado (*tf*) y la frecuencia de documento inversa (*idf*).

Por un lado, la frecuencia del término buscado (*tf*), que a modo de recordatorio, otorga más puntuación a aquellos documentos que mencionan más veces el término buscado, lo cual no es importante en nuestro sistema ya que además de que no todos las etiquetas están descritas con igual cantidad de detalle, sólo es importante saber si incluye el término buscado o no para contabilizarlo como posible candidato a sugerir.

Y por otro lado, tampoco es relevante para el sistema la frecuencia del término buscado en el documento (*tf*), que otorga más puntuación a aquellos documentos que contienen términos raros en el índice frente a los que contienen términos comunes por las mismas razones aportadas para el parámetro anterior.

La anulación de estos dos factores se implementó gracias a la posibilidad de configuración que brinda *Lucene* a los desarrolladores a través de la extensión de la clase que implementa la similitud.

Antes de comentar más en detalle cómo se lleva a cabo esta extensión, veamos un diagrama de clases que incluye la clase añadida en nuestra implementación:

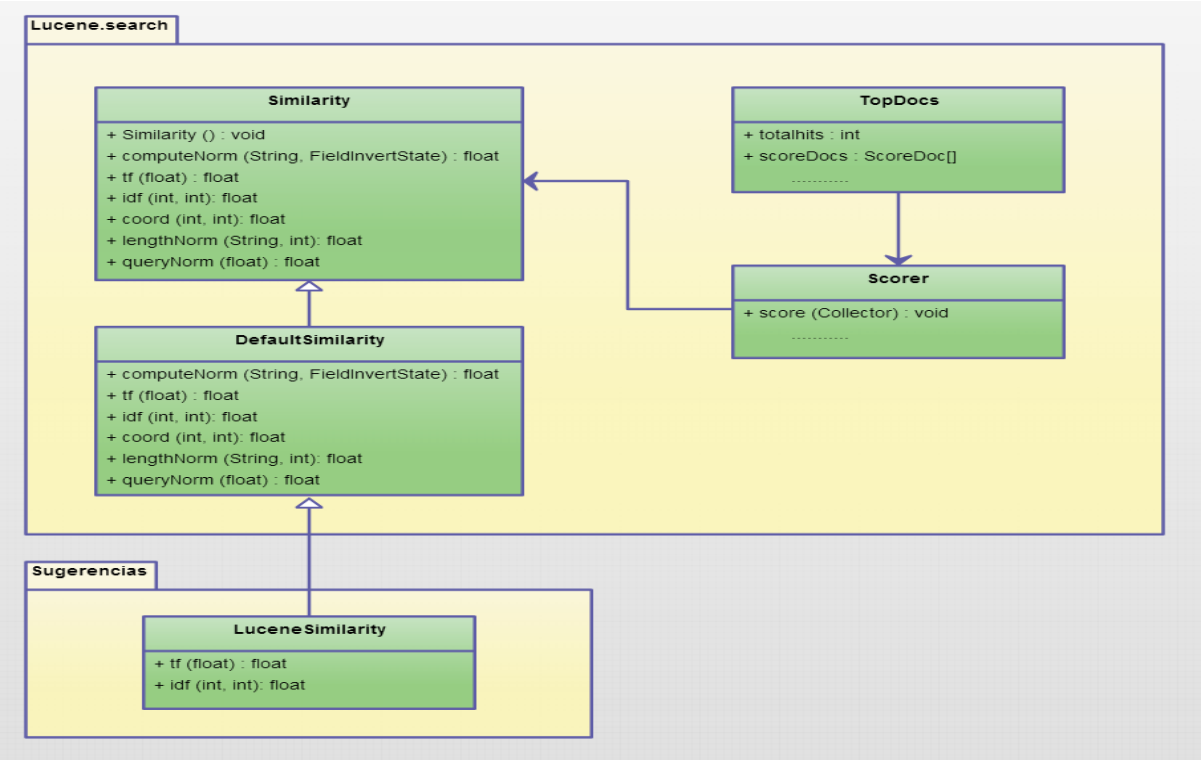


Ilustración 9 - Diagrama de clases de cálculo de puntuación

Lucene contiene una clase abstracta denominada *Similarity* que define los métodos que necesita el componente de cálculo de puntuaciones *Scorer* para el cálculo de la similitud o puntuación de documentos, utilizado a su vez para generar y ordenar el ranking de documentos.

Por defecto, *Lucene* implementa una versión de dicha clase denominada *DefaultSimilarity* que se corresponde con la fórmula anteriormente presentada.

Sin embargo, ofrece la posibilidad de implementar modificaciones de esta fórmula mediante la extensión de la clase anterior, acción llevada a cabo en este caso con la clase *LuceneSimilarity*, que extiende los métodos correspondientes a los factores *tf* e *idf* que se decidieron anular, lo cual se consiguió fijando su valor de retorno a 1.

La nueva fórmula implementada, que aumentó la tasa de acierto sensiblemente, se correspondería con la siguiente expresión:

$$puntuación(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{t\ en\ q} \left(boost(t) \cdot lengthNorm(t,d)\right)$$

2.3 Promoción de documentos

Lucene permite alterar los resultados de búsqueda mediante el *boosting* o promoción de varios niveles como son los documentos y los campos durante la indexación, o las consultas durante la búsqueda.

La promoción de campos y consultas ya se encontraba correctamente implementada en el sistema, priorizando el nombre de los documentos respecto a la descripción o reduciendo la de los términos *fuzzy* de una consulta respectivamente, por citar algunos ejemplos.

Sin embargo, la implementación de la promoción de documentos era bastante pobre, con los consiguientes resultados, ya que se basaba en el número de etiquetas del documento, sin importar su tipo.

Esta implementación motivaba resultados anómalos, de entre los cuales destacaba la imposibilidad de encontrar etiquetas de productos genéricas y muy importantes para un sistema de ventas de artículos como son por ejemplo chaquetas, bolsos, botas o colchones.

Se decidió cambiar el sistema, optando por una implementación que tuviera en cuenta no sólo el número de etiquetas, si no el tipo y la jerarquía de las mismas, asignando prioridades en función de cuán importantes fuesen para el sistema.

Así pues por ejemplo el documento correspondiente a la categoría *Zapatos* debería promocionarse más que el correspondiente a *Botas* dentro de la categoría *Zapatos* y éste a su vez más que el correspondiente a *Botas de tacón* bajo las categorías *Botas* y *Zapatos*, por orden de importancia.

La fórmula utilizada para calcular el factor de promoción de documentos tiene en cuenta además de la prioridad por jerarquía, el posterior uso que el motor de *Lucene* hace de él, dividiendo su valor entre 100 para normalizarlo en el intervalo [0,1]:

$$boosting(d) = \frac{100}{\sqrt{\sum peso_{tipo}}}$$

Así pues, en base a la anterior fórmula, los pesos de los tipos de etiqueta con mayor importancia deberán tener un valor menor y viceversa.

Tras realizar diversas pruebas, los pesos finalmente asignados fueron los siguientes:

- (1) Categoría Total:
  - (1) Género..... 2
    - (2) Estilo..... 4
      - (2) Sub_estilo..... 6
        - (2) Sub_sub_estilo..... 8
          - …

    (5) Marca Total:
    (1) Serie..... 6
    (1) Sub_serie..... 7
    - …

    (8) Resto

Cabe tener en cuenta que no es posible que un subtipo se encuentre solo por lo que pese a que en ciertos casos puedan tener el mismo valor que su tipo padre, a la hora del cálculo se sumarán los de ambos.

Esto se observa mejor con un ejemplo: el documento correspondiente a la etiqueta *Muebles* tendrá un peso 1, y el documento correspondiente a la etiqueta *Camas* dentro de *Muebles* tendrá un peso total de 2 por la suma de los pesos de ambas etiquetas, no pudiendo nunca existir un documento que incluya la etiqueta *Camas* sin incluir *Muebles*.

Aquellos documentos en que se combinen tipos de etiquetas de diversas jerarquías como pueda ser *Camisetas de Adidas* sumarán los pesos de la parte correspondiente al estilo y a la marca.

2.3.1 Problema de estructuración del árbol de etiquetas

Si bien el sistema recién descrito mejoró los resultados desde su implementación, tras diversas pruebas se detectó un error crítico que se explica detalladamente en este apartado.

El sitio web alemán difiere en multitud de aspectos del resto de sitios web internacionales, y en muchas situaciones y probablemente debido a que fue el pionero, arrastra ciertas características especiales que difieren del diseño óptimo y que no se terminan de solventar pues acarrearía un rediseño demasiado grande.

Entre ellas destaca, con relación a este proyecto, una estructuración ilógica en el árbol de etiquetas, que afecta directamente a la eficiencia del sistema de sugerencias.

En concreto, la moda infantil no se encuentra al mismo nivel que la moda de hombre y de mujer, si no un nivel por encima ya que las dos anteriores se encuentran englobadas dentro de la categoría *Moda*, inexistente en la organización del resto de países. De forma gráfica el problema es el siguiente:

Moda (Categoría)

Moda de mujer (Género)

Chaquetas (Estilo)

Moda de hombre (Género)

Chaquetas (Estilo)

Moda infantil (Categoría)

Chaquetas (Estilo)

Como se puede observar, ante la búsqueda genérica de chaquetas sin especificar un género concreto, aún pese a que en el sistema la importancia de este tipo de productos en el género femenino es mayor, y en base a ello reflejado en un mayor valor de *SortSum* debería aparecer como primera sugerencia, el peso de los documentos relativos a la moda infantil utilizando el nuevo sistema de promoción será mayor, puesto que se encuentran un nivel por encima de los equivalentes para hombre y mujer, haciendo que se consideren una mejor sugerencia.

Dada la dificultad para modificar la estructura del árbol de etiquetas debido a la consiguiente cantidad de código fuente del sistema que sería necesario readaptar, y pese a que los responsables de la gestión del mismo son conscientes del grave problema que supone, la única solución viable pasa por intervenir en tiempo de indexación y alterar la puntuación de los documentos relativos a productos de moda de hombre y de mujer.

En concreto, cuando se procesa uno de estos documentos, se calcula el peso del mismo considerando que las etiquetas de tipos *Estilo* y *Sub_estilos* se encontrasen un nivel por encima, o lo que es equivalente, actuando como si la categoría *Moda* no existiera.

2.4 Perfect match

Antes incluso de la existencia del actual sistema de sugerencias, el departamento de *Quality Management* de la empresa creó una extensa lista de consultas relativas a productos con un gran porcentaje de ventas en la página web, que incluía los identificadores de etiquetas relacionadas de forma que ante una búsqueda determinada se garantizase que el buscador redireccionase directamente a ellos, conduciéndoles al resultado que más interesaba que vieran.

Este sistema también es tenido en cuenta en la implementación actual del sistema de sugerencias, y por ello se incluyó en la lista de requisitos del presente proyecto.

La implementación llevada a cabo consiste en la detección de este tipo de consultas a través de una búsqueda de la entrada de usuario en la tabla correspondiente de la base de datos, y considerando el valor indicado como sugerencia perfecta en caso de que exista dicha entrada en la tabla.

La consideración de sugerencia perfecta implica que se mostrará en la primera posición de la lista desplegable de sugerencias mostrada al usuario.

Sin embargo, el hallazgo de esta sugerencia perfecta no garantiza con toda seguridad que sea lo buscado por el usuario, con lo que conviene presentarle sugerencias alternativas, calculadas de la misma forma que anteriormente aunque siempre presentadas como segunda y sucesivas mejores sugerencias.

Por último, obviamente, dada la alta probabilidad en muchos casos de que la sugerencia perfecta fuese encontrada a través del motor de *Lucene*, se añadió un módulo de verificación que evitase presentar la misma sugerencia por duplicado.

2.5 Solución de errores en Lucene

Además de las grandes modificaciones anteriores, a lo largo del desarrollo del proyecto se detectaron multitud de errores tanto en el motor de *Lucene* como en las descripciones de las etiquetas en la base de datos. Algunos de los más notorios respecto a *Lucene* se listan a continuación en este apartado.

2.5.1 Variación de los tipos de documentos indexados

El índice no incluía documentos relativos a etiquetas de tipo *Tienda, Look y Talla*, posiblemente porque cuando se llevó a cabo su desarrollo no se consideraban suficientemente importantes.

En la actualidad, especialmente en lo relativo a tiendas (el sitio web alemán cuenta con más de 500 tiendas asociadas) es fundamental permitir búsquedas tan comunes como puedan ser productos de una determinada tienda, por lo que se consideró añadir este tipo de etiquetas al índice.

Respecto a las etiquetas de tipo *Look* y *Talla*, si bien no son tan comunes en la búsqueda, no suponen un gran aumento de tamaño ni de pérdida de procesamiento del índice por lo que se decidió aprovechar para indexarlos también y hacer viables posibles búsquedas al respecto.

2.5.2 Descomposición de aliases

Una de las características más propias del idioma alemán es la formación de palabras a partir de la unión de dos o más de ellas.

Debido a que no existe forma directa de reproducir este concepto al castellano, para facilitar la comprensión a no germanoparlantes comenzaremos exponiendo un ejemplo sencillo de forma que se pueda tener en mente a lo largo del resto de la explicación y ayude a asimilar el problema presentado:

Partiendo de las palabras *Herren* (hombre) y *Schuhe* (zapatos), el equivalente a la expresión *Zapatos de hombre* en idioma alemán correspondería a la unión de los dos términos anteriores en uno solo, en este caso *Herrenschuhe*.

El anterior ejemplo es suficiente para comprender el resto de la explicación, pero es importante puntualizar que podrían añadirse más términos añadiendo más significado y por supuesto más longitud a la palabra anterior, y uniones de 3 o 4 términos de esa forma son ampliamente utilizados en el sistema.

Volviendo al asunto que nos ocupa, el sistema de etiquetas está preparado para utilizar esta forma de creación de palabras sin que afecte a su funcionamiento, añadiendo entre los alias de la etiqueta la palabra con un separador representado por un guión '-' entre cada una de las palabras que la forman.

En nuestro ejemplo, entre los alias de la etiqueta con nombre *Herrenschuhe* se encontraría *herren-schuhe*, cuyo significado literal sería que *herrenschuhe* es equivalente a *herren-schuhe* y a *herren schuhe*, y por tanto los tres términos conducirían en una búsqueda a esa etiqueta.

Sin embargo, esto no era tenido en cuenta por la implementación de *Lucene*, y en el momento de creación de los documentos, al ir recorriendo y almacenando todos los campos que describen una etiqueta, o lo que es lo mismo, los que permiten encontrarla en la búsqueda, en este caso sólo se almacenaba *herrenschuhe* al extraerse del nombre de la etiqueta y *herren-schuhe* extraído del alias.

Esta situación era aún peor con otros alias si diferían del nombre de la etiqueta, ya que entonces solo se almacenaba una de las tres combinaciones, equivalente en realidad a reducir a un tercio las posibilidades de encontrar la etiqueta en una búsqueda.

Esta anomalía, que puede no parecer muy importante, provocaba que búsquedas como *herrenschuhe* tuvieran resultados satisfactorios y en cambio otras como *herren schuhe* obtuvieran resultados realmente dispares y carentes de sentido.

De cara a solucionarlo se plantearon dos posibilidades, realizar la división en tiempo de indexación de documentos o en tiempo de búsqueda.

Claramente, y más teniendo en cuenta que la indexación se hace semanalmente, la primera opción es mucho más óptima ya que para un sistema como el manejado es preferible sacrificar espacio en disco (al extenderse los campos descriptivos de los documentos el índice crece considerablemente) a tiempo de respuesta de cara al usuario.

2.5.3 Filtro por género

En determinadas situaciones, pese a que la consulta indicase explícitamente el género, *Lucene* incluía entre los resultados, aunque nunca en los primeros puestos, sugerencias del resto de géneros.

Ante esta situación, y considerando lógica la presunción de que si por ejemplo un usuario solicita la búsqueda de *Camisas de hombre* no desea obtener camisas de mujer o de niño, como sí podría suceder si buscase por un término más genérico como *Camisas*, se optó por filtrar y eliminar estas sugerencias de entre los resultados.

En la primera versión implementada se optó por eliminarlos directamente de la lista de sugerencias calculada por *Lucene*, pero tras la realización de pruebas posteriores, pruebas que serán expuestas más adelante en este documento, se comprobó que el tiempo de búsqueda generaba un cuello de botella en el tiempo de respuesta con lo que parecía razonable tratar de reducirlo, además de que a su vez también mejoraría el tiempo de análisis o post-procesamiento evitando realizar el filtro en él.

Por tanto, la versión finalmente implementada consiste en que una vez identificado que el usuario especificó un género concreto en la consulta, se modifica la consulta a realizar a *Lucene* indicando que el resto de géneros deben ser excluidos de los resultados, lo cual se puede hacer utilizando el operador '-' como ya vimos en la descripción inicial de *Lucene*.

2.5.4 Lista negra

Casi al final del desarrollo del proyecto se requirió por parte de la empresa la implementación de un sistema de lista negra que ante determinadas búsquedas excluyese ciertos resultados no deseados.

Esta solicitud final surgió a raíz de la asunción de problemas en el funcionamiento del sistema de normalización de *Lucene*, el cual será detallado más en profundidad en las conclusiones del proyecto, pero del que se puede adelantar que provoca que términos con un significado muy dispar pero con una misma raíz léxica sean interpretados como iguales por *Lucene*, apareciendo entre sus resultados.

Evidentemente, esta forma de actuar, a posteriori, sólo funciona para casos conocidos, en los que se sepa qué resultados incorrectos aparecen ante determinadas consultas, y es una solución temporal hasta la implementación de mejoras en el sistema de normalización de *Lucene*, sobre el que volveremos más adelante.

La implementación en sí es muy sencilla y consta de un conjunto de pares formado por la cadena que representa a la consulta y el identificador de las etiquetas a evitar cuando ésta sea buscada por algún usuario.

En el momento del análisis de sugerencias de *Lucene*, en caso de coincidir la entrada de usuario con algún valor almacenado en dicho conjunto, se descartan aquellas sugerencias que contengas los identificadores de etiquetas a evitar.

3. Tecnologías utilizadas

El desarrollo del proyecto se realizó en el lenguaje de programación *Java*, utilizando el motor de indexación de documentos de *Apache Lucene*.

En cuanto al entorno de desarrollo, se optó por una máquina con sistema operativo *Windows 7* y el *IDE Eclipse* en su versión *Indigo*.

En lo relativo a la base de datos, se utilizó el sistema de gestión de bases de datos *MySQL*, aunque en la gran mayoría de situaciones el acceso a la misma se realizó a través del sistema de *backoffice* de la empresa.

Por último, se utilizó la herramienta *Luke*, de gran utilidad en muchas fases del desarrollo, que permite analizar y realizar búsquedas en índices ya existentes creados mediante *Lucene*, a partir de una interfaz gráfica y permitiendo configurar multitud de parámetros, facilitando la realización de todo tipo de pruebas y simulaciones de nuevas funcionalidades a añadir al sistema. Sirva la siguiente captura de pantalla como ejemplo de simulación de una búsqueda:

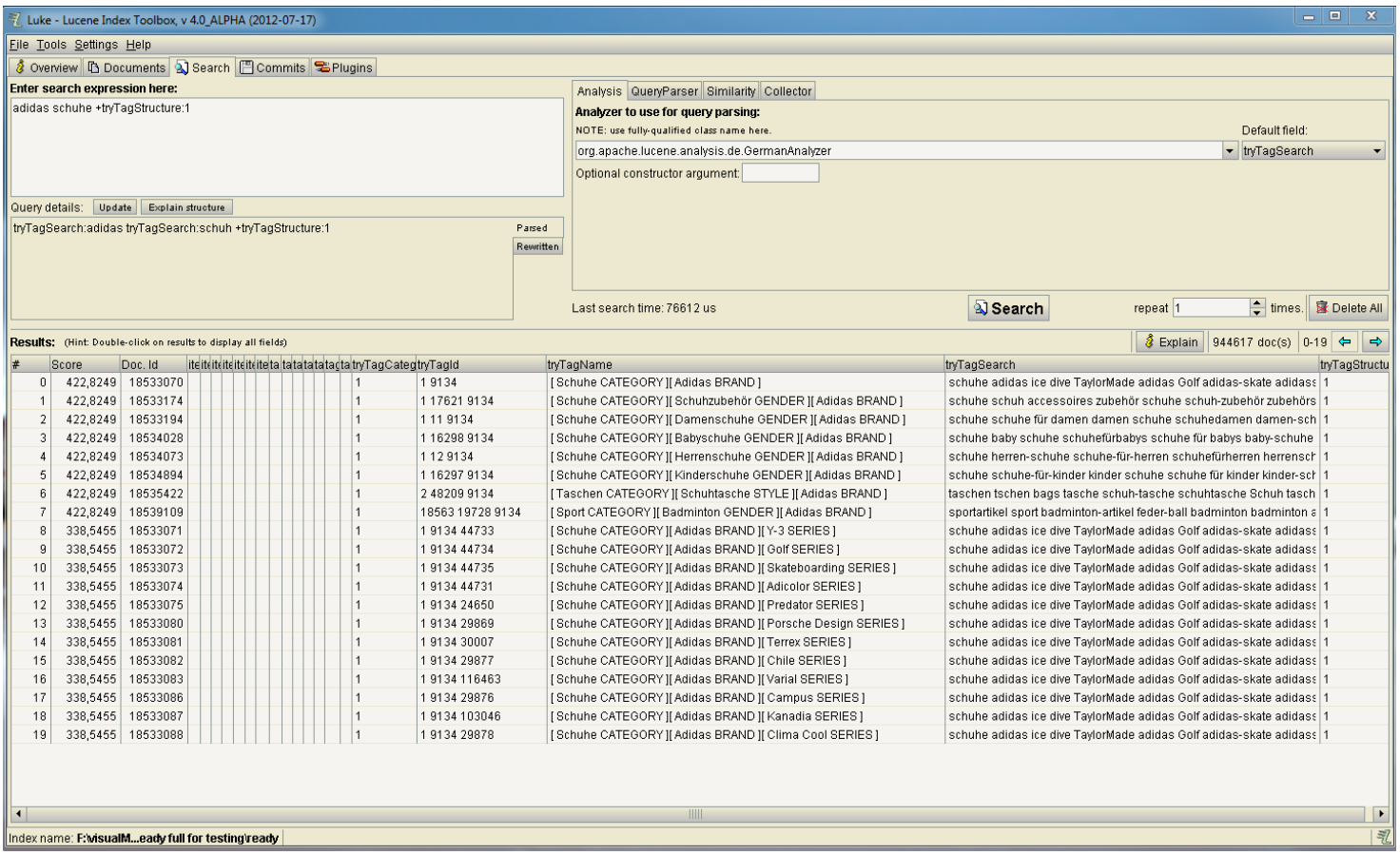


Ilustración 10 - Pantalla de ejemplo simulador Luke

Resultados experimentales

A lo largo del desarrollo del proyecto se fueron llevando a cabo pruebas para medir la evolución del sistema implementado comparándolo con el ya existente.

En concreto se realizaron con los dos conjuntos de casos de prueba facilitados al inicio del proyecto con ejemplos reales, formados por pares de entrada de usuario e identificadores de etiquetas esperados como sugerencia:

- C1 - Consultas: compuesto por 140 entradas de usuario de múltiples tipos.
- C2 - Consultas extendidas: compuesto por 17 entradas de usuario no regulares, comprendiendo ejemplos de los tipos *fuzzy*, orden, género y alias.

De aquí en adelante toda referencia a dichos conjuntos de prueba utilizará la nomenclatura C1 y C2 para facilitar la lectura.

Cada uno de los conjuntos consiste en un *array* de objetos de tipo *Query*, representativos de las entradas de usuario, y que por su parte incluyen dos campos: el texto de la consulta y los identificadores de etiqueta que se esperan como salida correcta, y con los que se compararán los resultados obtenidos.

Veamos un ejemplo:

```
public static Query[] C1 = new Query[]{
        new Query( "rote adidas schuhe", "15115,9134,1" ),
        new Query( "lego star wars", "48020,19346,18276" ),
        new Query( "mädchen shirts von desigual", "42618,42614,17063" ),
        ...
};
```

Estos conjuntos serán ejecutados en un programa de pruebas *JUnit* que mostrará los resultados obtenidos una vez ejecutados por completo.

Las comparativas se realizaron en tres puntos temporales del desarrollo: el estado en el que se encontraba antes de comenzar, tras la implementación de la primera gran mejora consistente en la reorganización, y el estado final tras la implementación de mejoras en el núcleo de *Lucene*.

1. Tasa de acierto

1.1 Estado inicial

Al inicio del proyecto, como ya se ha comentado la tasa de acierto del sistema en el conjunto C1 era muy baja, exactamente del 32%, mientras que en el conjunto C2 se podría considerar prácticamente nula ya que era del 5%.

1.2 Estado posterior a la implementación de la reorganización

C1	Sistema antiguo	Nuevo sistema
Tasa de acierto (%)	74.29	76.43
Sugerencias encontradas	951	1332
Resultados únicos encontrados	15	20

Tabla 1 - Tasa de acierto C1 - Tras reorganización

C2	Sistema antiguo	Nuevo sistema
Tasa de acierto (%)	47.06	83.33
Sugerencias encontradas	71	165
Resultados únicos encontrados	1	7

Tabla 2 - Tasa de acierto C2 - Tras reorganización

Interpretación de los resultados

En estas tablas se observa que tras la primera gran mejora, el rendimiento del nuevo sistema ante consultas regulares como las del conjunto C1 se equiparó, e incluso superó ligeramente al del sistema antiguo.

Pero más importante si cabe, ya que fue el aspecto para el cual con más interés se planeó la mejora, es la diferencia de rendimiento, a favor del nuevo sistema, en cuanto a la búsqueda de consultas no regulares como podemos ver en la segunda tabla, correspondiente al conjunto C2.

1.3 Estado posterior a la implementación de mejoras en el núcleo de Lucene

Antes de comenzar, cabe destacar que se observará una mayor tasa de acierto en el sistema basado en *Trie* con respecto al estado anterior, ya que a la par del desarrollo de este proyecto se llevaron a cabo mejoras en él durante el tiempo transcurrido entre ambos estados, buena parte de las cuales derivadas del descubrimiento de errores en la estructura del árbol de etiquetas y en la descripción de éstas en el presente proyecto, por lo que se puede afirmar que el presente trabajó también colaboró a mejorar los resultados del sistema de sugerencias antiguo, presente aún a día de hoy en el sitio web.

También se puede apreciar que el número de sugerencias encontradas por el nuevo sistema disminuye respecto al apartado anterior, lo cual es debido a un ajuste más fino a la hora de aceptar resultados como válidos respecto a la consulta.

C1	Sistema antiguo	Nuevo sistema
Tasa de acierto (%)	88.57	95.71
Sugerencias encontradas	1069	1257
Resultados únicos encontrados	4	11

Tabla 3 - Tasa de acierto C1 - Tras mejoras núcleo de Lucene

C2	Sistema antiguo	Nuevo sistema
Tasa de acierto (%)	64.71	94.44
Sugerencias encontradas	85	148
Resultados únicos encontrados	0	5

Tabla 4 - Tasa de acierto C2 - Tras mejoras núcleo de Lucene

En este caso, y como única excepción en el período de pruebas, se analizó el rendimiento sobre un tercer conjunto de 8 consultas, consistente en consultas genéricas equivalentes a los nombres de las etiquetas de tipo *Categoría*, como son *taschen* (bolsos), *kleider* (vestidos), *hosen* (pantalones), *jacken* (chaquetas), *blusen* (camisetas) y *hemden* (camisas); así como un par de tiendas (*tchibo.de* y *mogani.de*) para probar las nuevas capacidades de indexación añadidas en esta fase del desarrollo y compararlas con el sistema antiguo.

Consultas extra	Sistema antiguo	Nuevo sistema
Tasa de acierto (%)	62.50	100
Sugerencias encontradas	50	62
Resultados únicos encontrados	0	3

Tabla 5 - Tasa de acierto consultas extra - Tras mejoras núcleo de Lucene

Interpretación de los resultados

Nuevamente vemos que pese a la notable mejora del sistema antiguo durante esta fase, las nuevas mejoras implementadas sobre el sistema desarrollado mantienen la tasa de acierto de éste último por encima.

También es muy importante destacar la obtención de un mayor número de sugerencias en el sistema nuevo, ya que ofrecen más posibilidades al usuario que le permitan afinar aún más la búsqueda a sus necesidades en el caso de que la mejor sugerencia calculada no coincida con su voluntad.

Obviamente también es importante que éstas estén relacionadas con la búsqueda, motivo por el cual se decidió afinar más la lista de resultados, motivando, como ya se ha comentado, una reducción del número de sugerencias totales encontradas respecto a la fase anterior.

2. Tiempo de ejecución

2.1 Estado inicial

Respecto al tiempo de ejecución no se consideró muy relevante el comportamiento en el estado inicial ya que el funcionamiento distaba bastante de ser el esperado por lo cual no representa un buen factor comparativo.

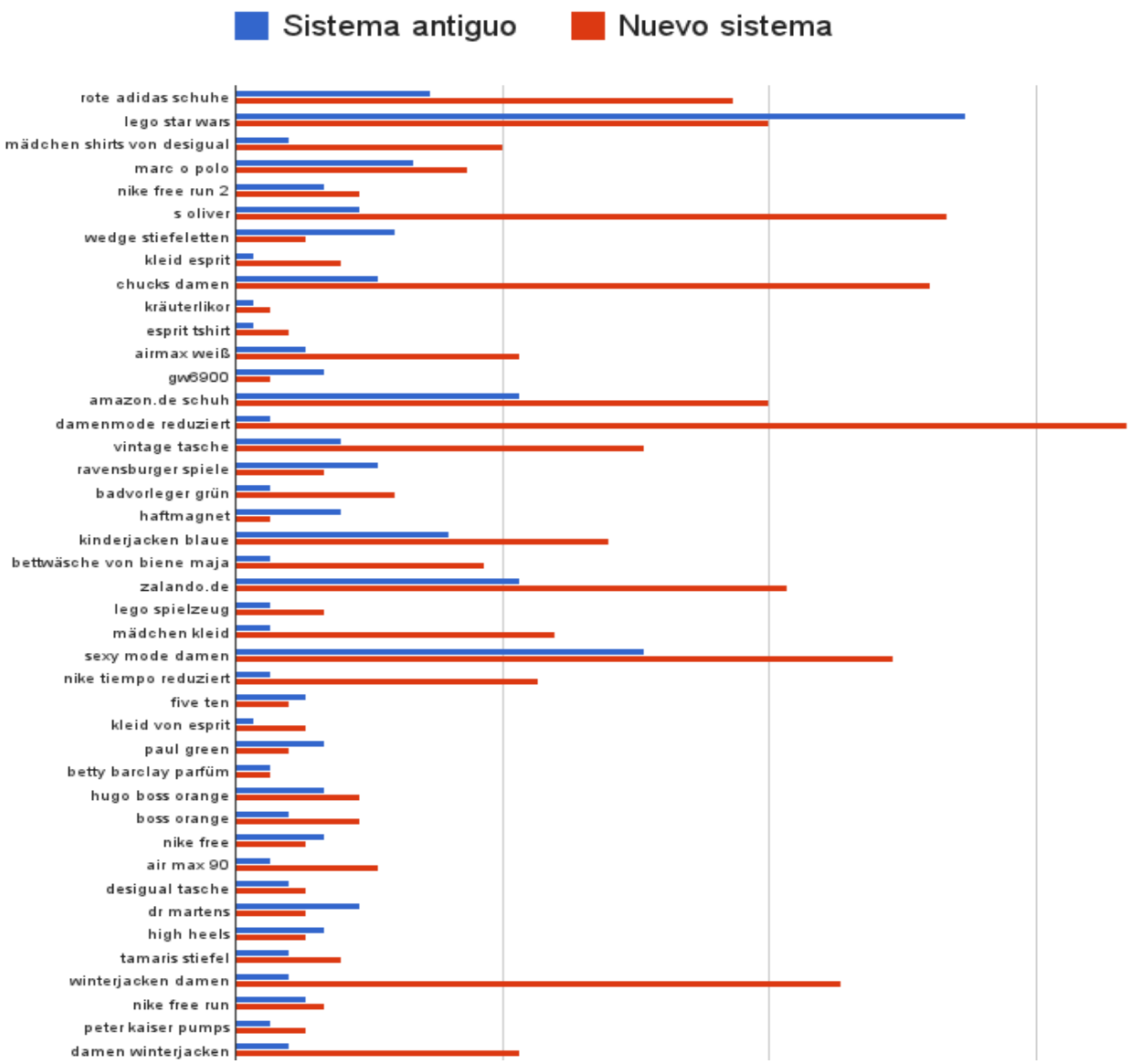
2.2 Estado posterior a la implementación de la reorganización

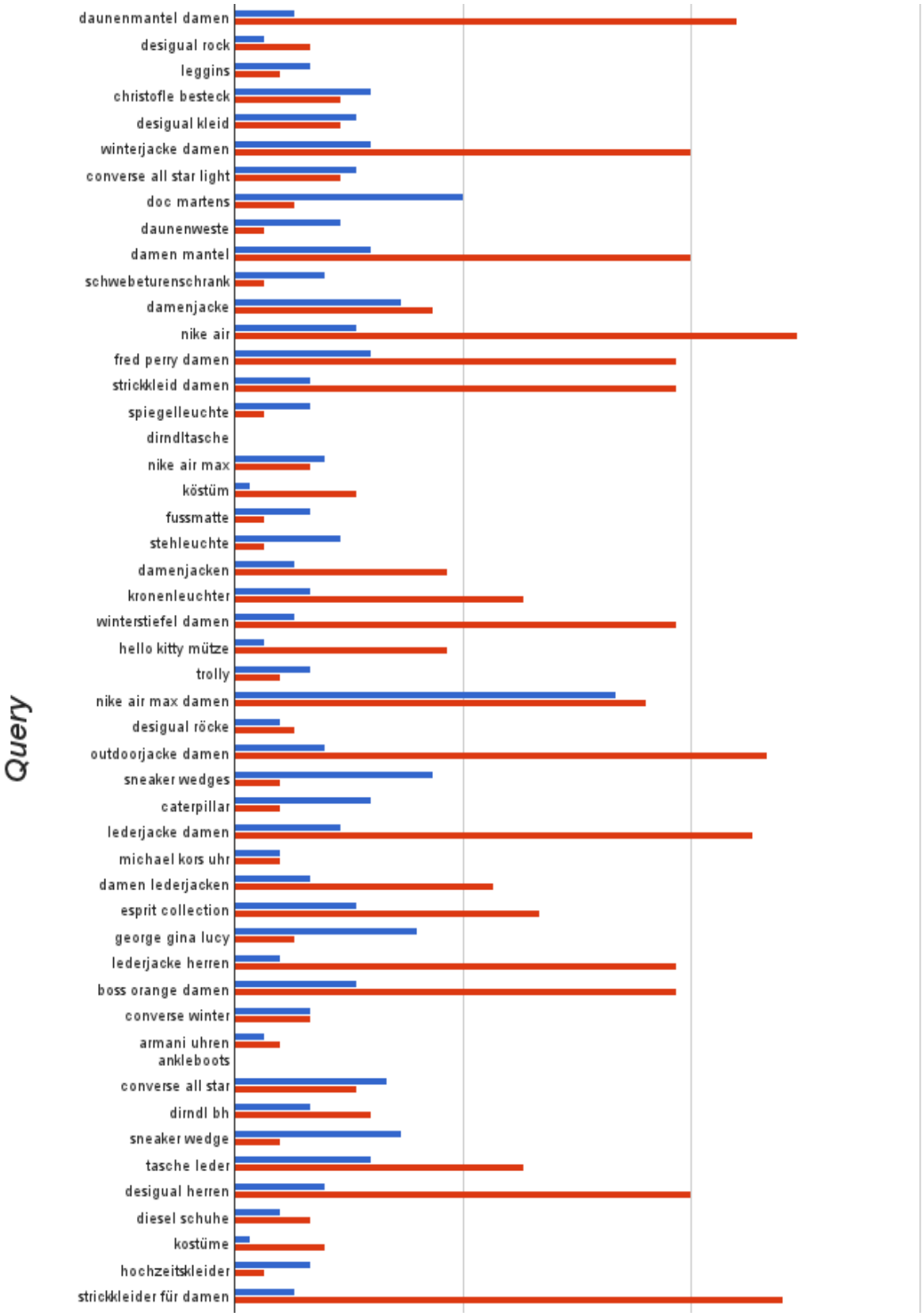
Análogamente al apartado anterior, se muestran en las siguientes tablas valores que resumen el comportamiento de ambos sistemas respecto a los conjuntos de prueba.

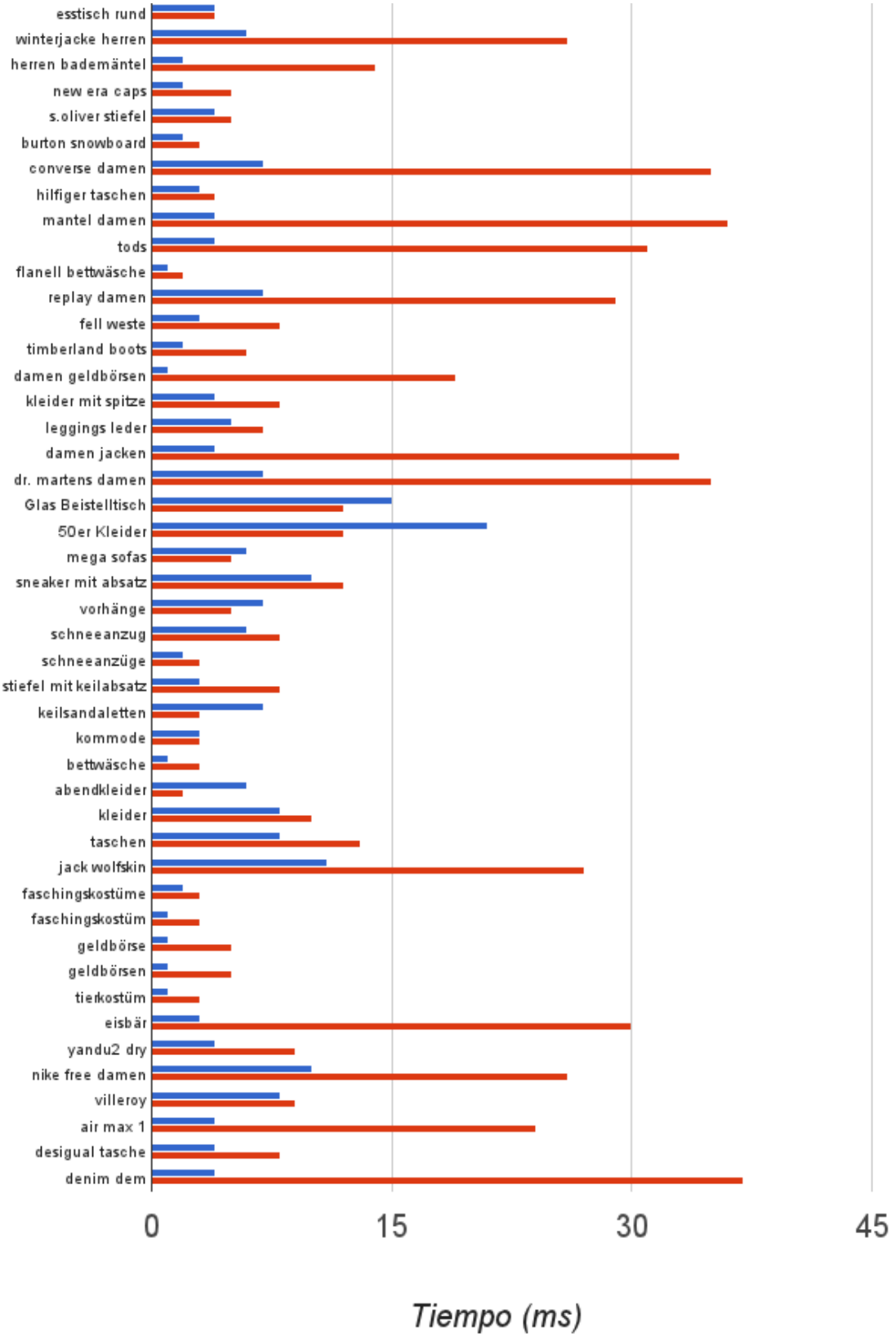
Además, en este caso se muestra también una representación gráfica de los tiempos de ejecución de cada uno de los ejemplos de los conjuntos, para permitir una mejor valoración de los resultados.

C1	Sistema antiguo	Nuevo sistema
Tiempo medio (ms)	5.893	10.935
Número de búsquedas más lentas que la media	4	38
Tiempo medio excluyendo búsquedas lentas	5.257	6.568

Tabla 6 - Tiempo de ejecución C1 - Tras reorganización




Query
daunenmantel damen
desigual rock
leggins
christofle besteck
desigual kleid
winterjacke damen
converse all star light
doc martens
daunenweste
damen mantel
schwebeturenschrank
damenjacke
nike air
fred perry damen
strickkleid damen
spiegelleuchte
dirndltasche
nike air max
köstüm
fussmatte
stehleuchte
damenjacken
kronenleuchter
winterstiefel damen
hello kitty mütze
trolly
nike air max damen
desigual röcke
outdoorjacke damen
sneaker wedges
caterpillar
lederjacke damen
michael kors uhr
damen lederjacken
esprit collection
george gina lucy
lederjacke herren
boss orange damen
converse winter
armani uhren
ankleboots
converse all star
dirndl bh
sneaker wedge
tasche leder
desigual herren
diesel schuhe
kostüme
hochzeitskleider
strickkleider für damen



Gráfica 1 - Tiempo de ejecución - Tras reorganización

C2	Sistema antiguo	Nuevo sistema
Tiempo medio (ms)	8.606	13.2
Número de búsquedas más lentas que la media	2	7
Tiempo medio excluyendo búsquedas lentas	6.23	5.6

Tabla 7 - Tiempo de ejecución C2 - Tras reorganización

Gráfica 2 - Tiempo de ejecución C2 - Tras reorganización

Interpretación de los resultados

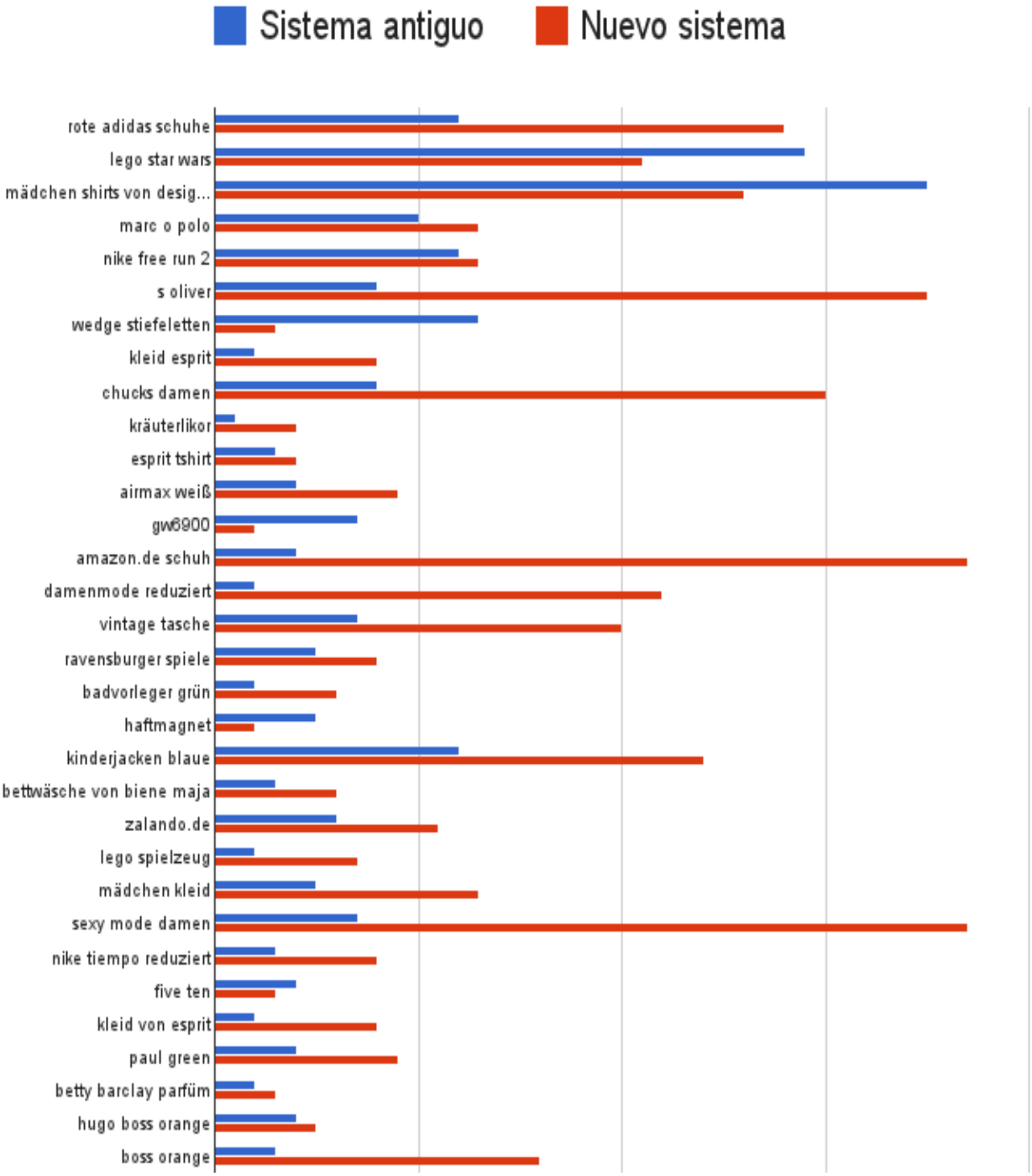
Como principal dato a destacar se observa que pese a que para un alto porcentaje de *queries* el tiempo de ejecución es aceptable y similar al sistema antiguo, existen una serie de *queries* para las cuales la ejecución se prolonga demasiado y que condicionan el tiempo medio del conjunto, pues cuando las excluimos el rendimiento medio es más o menos parejo.

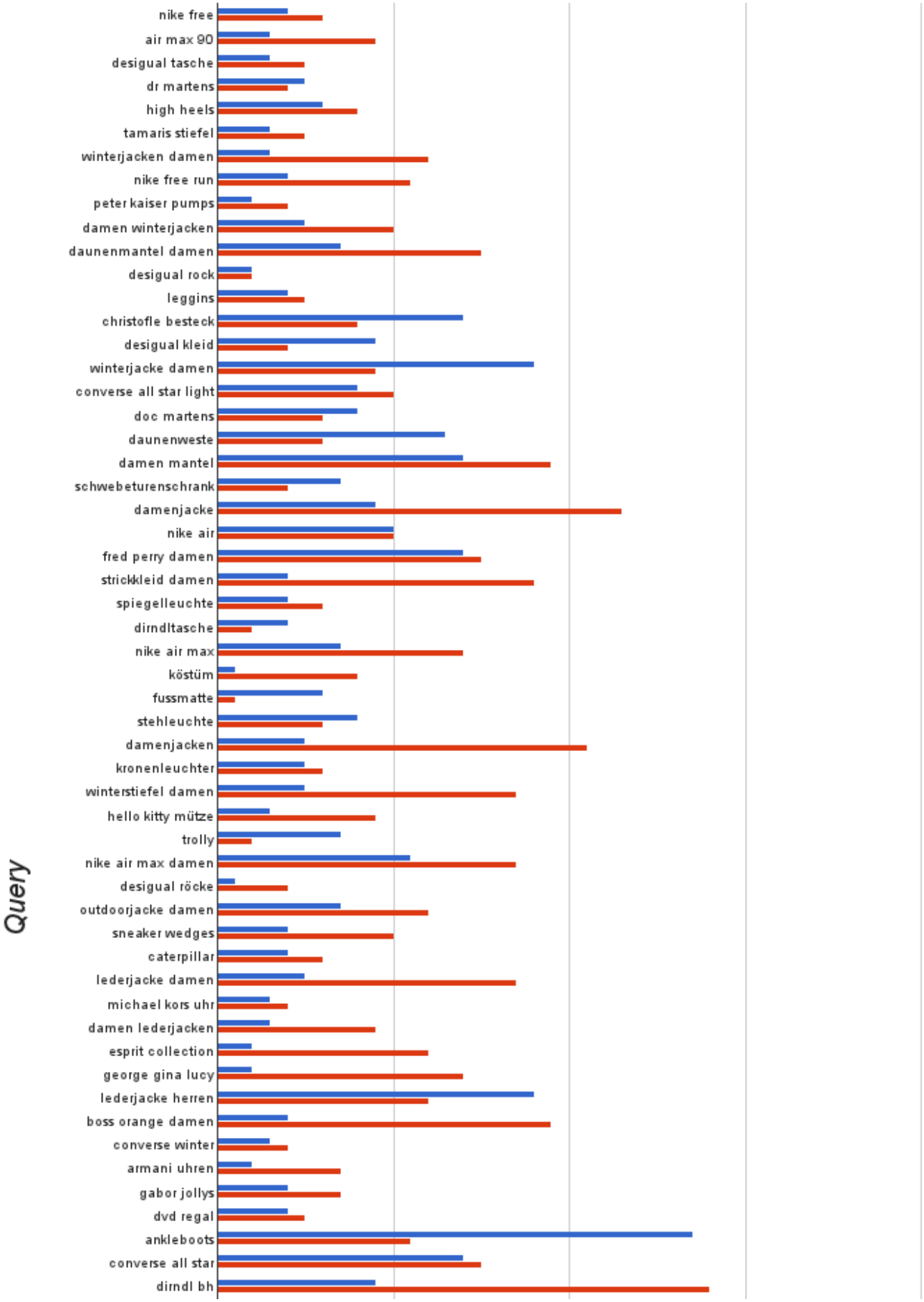
Con el fin de acelerar estas búsquedas lentas se llevaron a cabo las mejoras en el núcleo de *Lucene*, cuyo rendimiento analizamos a continuación.

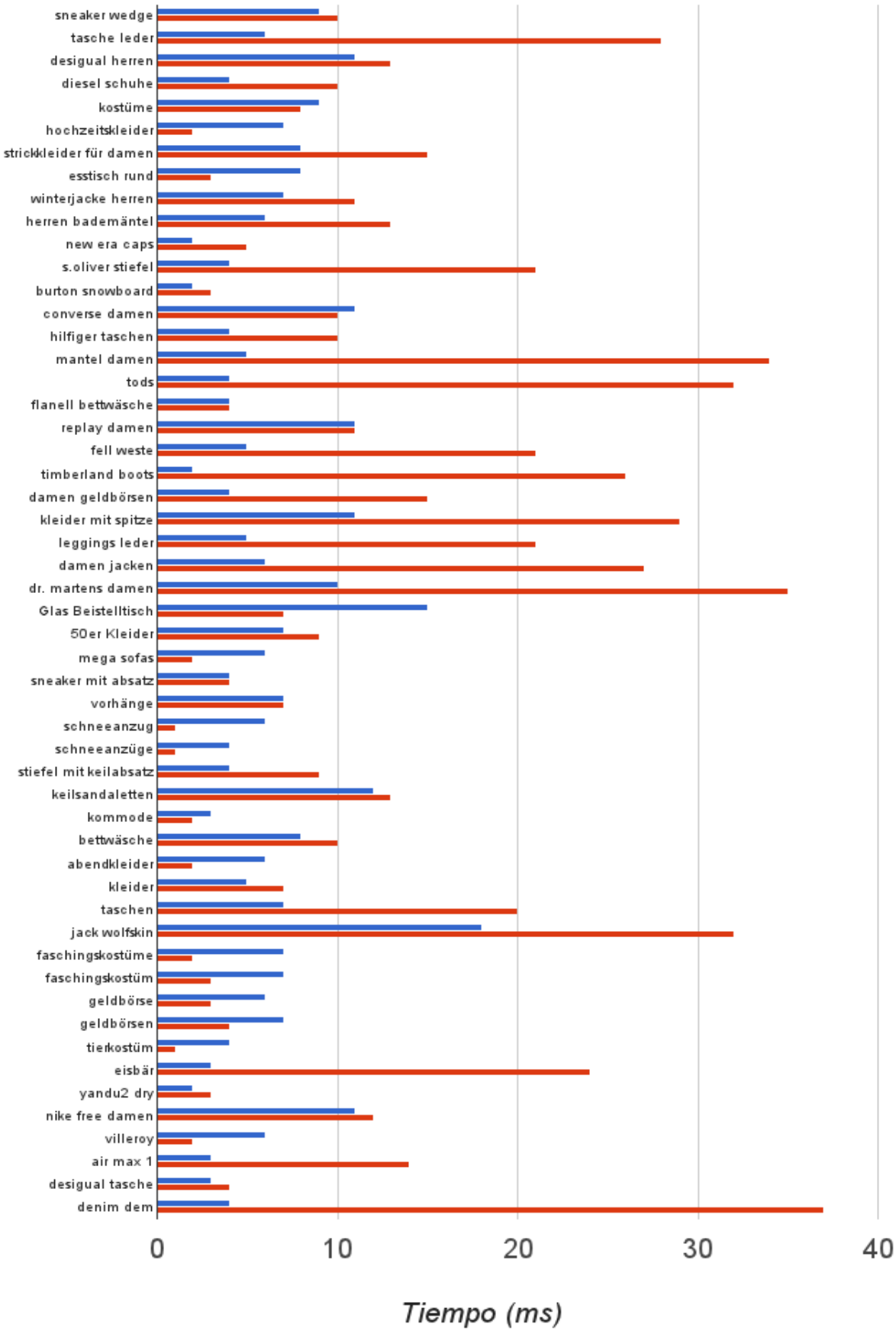
2.3 Estado posterior a la implementación de mejoras en el núcleo de Lucene

C1	Sistema antiguo	Nuevo sistema
Tiempo medio (ms)	6.621	11.614
Número de búsquedas más lentas que la media	3	27
Tiempo medio excluyendo búsquedas lentas	6.102	7.849

Tabla 8 - Tiempo de ejecución C1 - Tras mejoras núcleo de Lucene


Sistema antiguo
Nuevo sistema
rote adidas schuhe
lego star wars
mädchen shirts von desig...
marc o polo
nike free run 2
s oliver
wedge stiefeletten
kleid esprit
chucks damen
kräuterlikor
esprit tshirt
airmax weiß
gw6900
amazon.de schuh
damenmode reduziert
vintage tasche
ravensburger spiele
badvorleger grün
haftmagnet
kinderjacken blaue
bettwäsche von biene maja
zalando.de
lego spielzeug
mädchen kleid
sexy mode damen
nike tiempo reduziert
five ten
kleid von esprit
paul green
betty barclay parfüm
hugo boss orange
boss orange

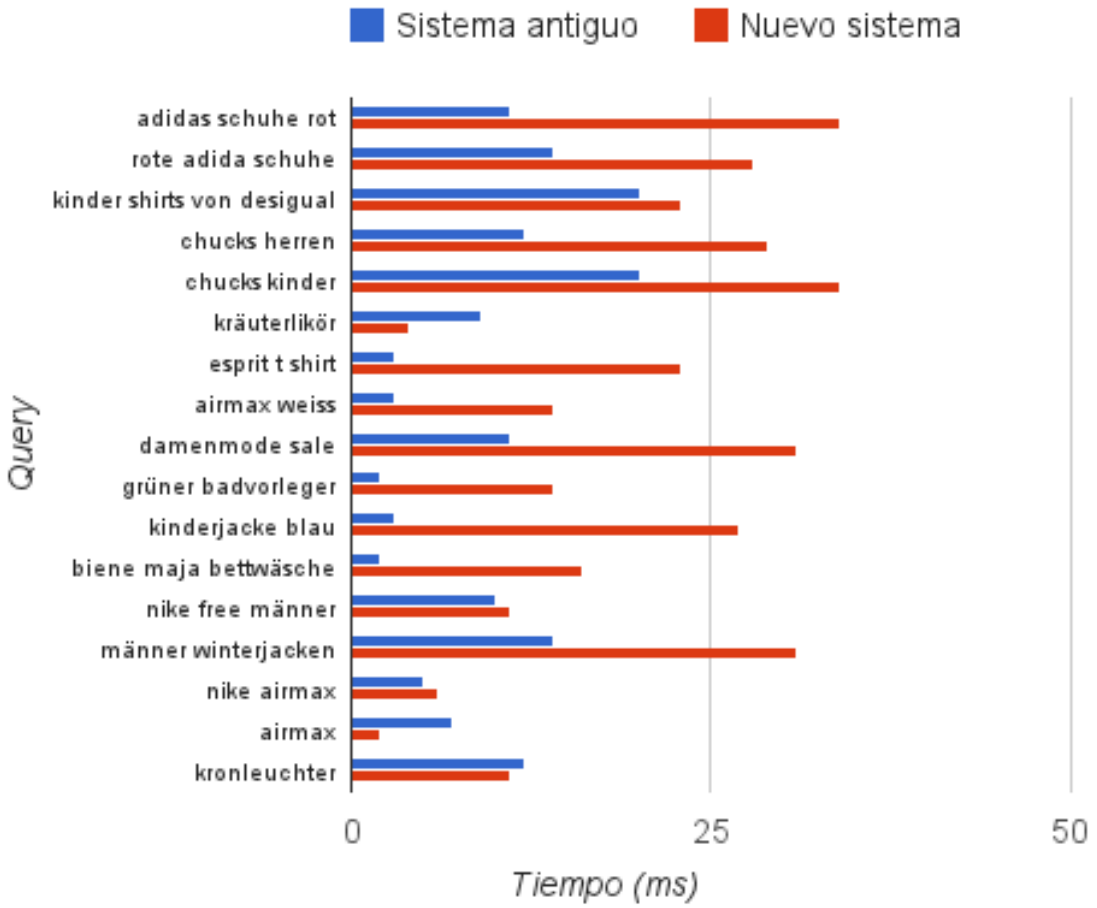

Query
nike free
air max 90
desigual tasche
dr martens
high heels
tamaris stiefel
winterjacken damen
nike free run
peter kaiser pumps
damen winterjacken
daunenmantel damen
desigual rock
leggins
christofle besteck
desigual kleid
winterjacke damen
converse all star light
doc martens
daunenweste
damen mantel
schwebeturenschrank
damenjacke
nike air
fred perry damen
strickkleid damen
spiegelleuchte
dirndltasche
nike air max
köstüm
fussmatte
stehleuchte
damenjacken
kronenleuchter
winterstiefel damen
hello kitty mütze
trolly
nike air max damen
desigual röcke
outdoorjacke damen
sneaker wedges
caterpillar
lederjacke damen
michael kors uhr
damen lederjacken
esprit collection
george gina lucy
lederjacke herren
boss orange damen
converse winter
armani uhren
gabor jollys
dvd regal
ankleboots
converse all star
dirndl bh



Gráfica 3 - Tiempo de ejecución C1 - Tras mejoras núcleo de Lucene

C2	Sistema antiguo	Nuevo sistema
Tiempo medio (ms)	9.294	15.9
Número de búsquedas más lentas que la media	2	9
Tiempo medio excluyendo búsquedas lentas	7.866	9.75

Tabla 9 - Tiempo de ejecución C2 - Tras mejoras núcleo de Lucene



Gráfica 4 - Tiempo de ejecución C2 - Tras mejoras núcleo de Lucene

Interpretación de los resultados

Tras la realización de esta segunda fase de mejoras, se han reducido tanto el tiempo medio de ejecución como el número de búsquedas lentas en ambos conjuntos de prueba, con respecto a la fase anterior.

Sin embargo, pese a la mejora, los valores obtenidos distan aún del rendimiento producido por el sistema antiguo, ya que si bien en la mayoría de casos el tiempo de ejecución, aún siendo ligeramente mayor, es asumible, para ciertas consultas concretas supera el tiempo esperado para ajustarse a un sistema de tiempo real, produciendo una demora en el tiempo de respuesta al usuario.

Con el fin de analizar más en profundidad la causa de este incremento en el tiempo de ejecución, se realizaron las pruebas mostradas a continuación, analizando la ejecución en dos fases separadas: la búsqueda y el análisis posterior.

3. Tiempo de búsqueda frente a tiempo de análisis

3.1 Estado inicial

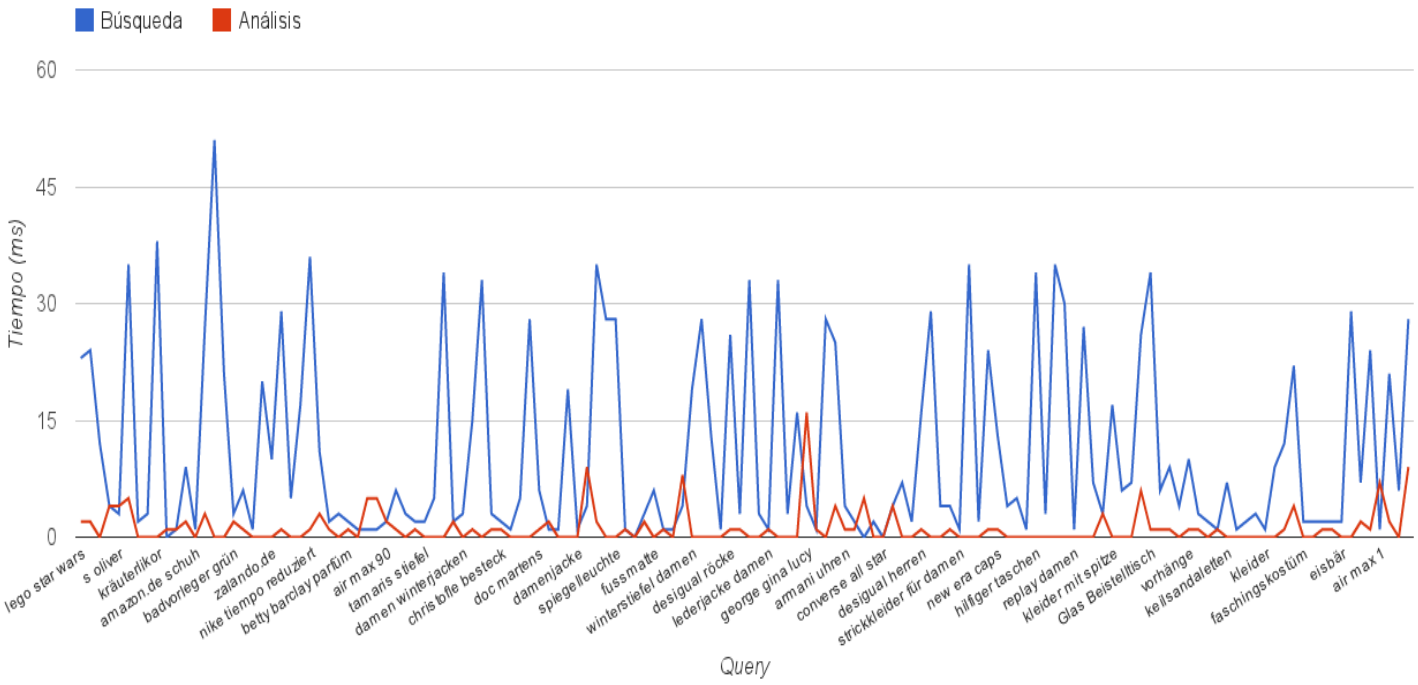
Durante la medición de tiempos de respuesta para el *benchmark* anterior, se observó que el tiempo de respuesta del módulo encargado tanto de preparar la consulta a buscar como de realizar la propia búsqueda, cuyo desarrollo previo se daba por finalizado y optimizado al inicio del proyecto, era ya superior en gran parte de los ejemplos de los conjuntos de prueba al tiempo total de respuesta del sistema de sugerencias antiguo.

Por tanto, tras observar un peor rendimiento en el tiempo de respuesta tras la realización de los *benchmarks* anteriores, se decidió medir por separado el tiempo de la implementación llevada a cabo y el tiempo del módulo de búsqueda, para verificar si los tiempos obtenidos se veían únicamente lastrados por este cuello de botella o debería intentar incrementarse la velocidad del sistema desarrollado.

3.2 Estado posterior a la implementación de la reorganización

C1	Sistema antiguo	Nuevo sistema	
		Búsqueda	Análisis
Tiempo medio (ms)	5.893	10.936	1.15

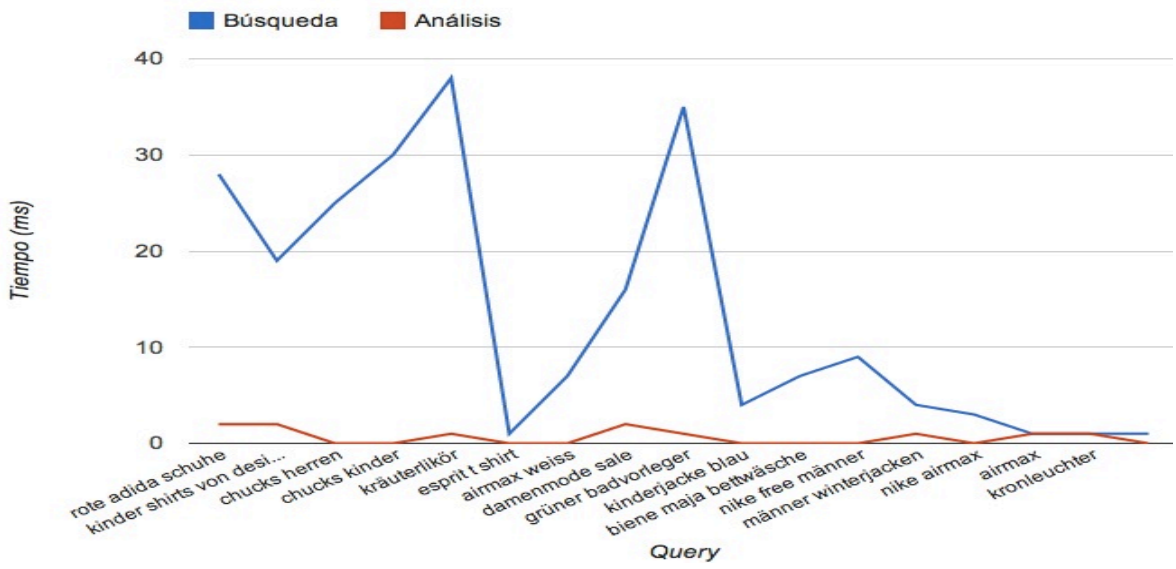
Tabla 10 - Tiempo de búsqueda frente a tiempo de análisis C1 - Tras reorganización



Gráfica 5 - Tiempo de búsqueda frente a tiempo de análisis C1 - Tras reorganización

C2	Sistema antiguo	Nuevo sistema	
		Búsqueda	Análisis
Tiempo medio (ms)	8.606	12.722	0.611

Tabla 11 - Tiempo de búsqueda frente a tiempo de análisis C2 - Tras reorganización



Gráfica 6 - Tiempo de búsqueda frente a tiempo de análisis C2 - Tras reorganización

Interpretación de los resultados

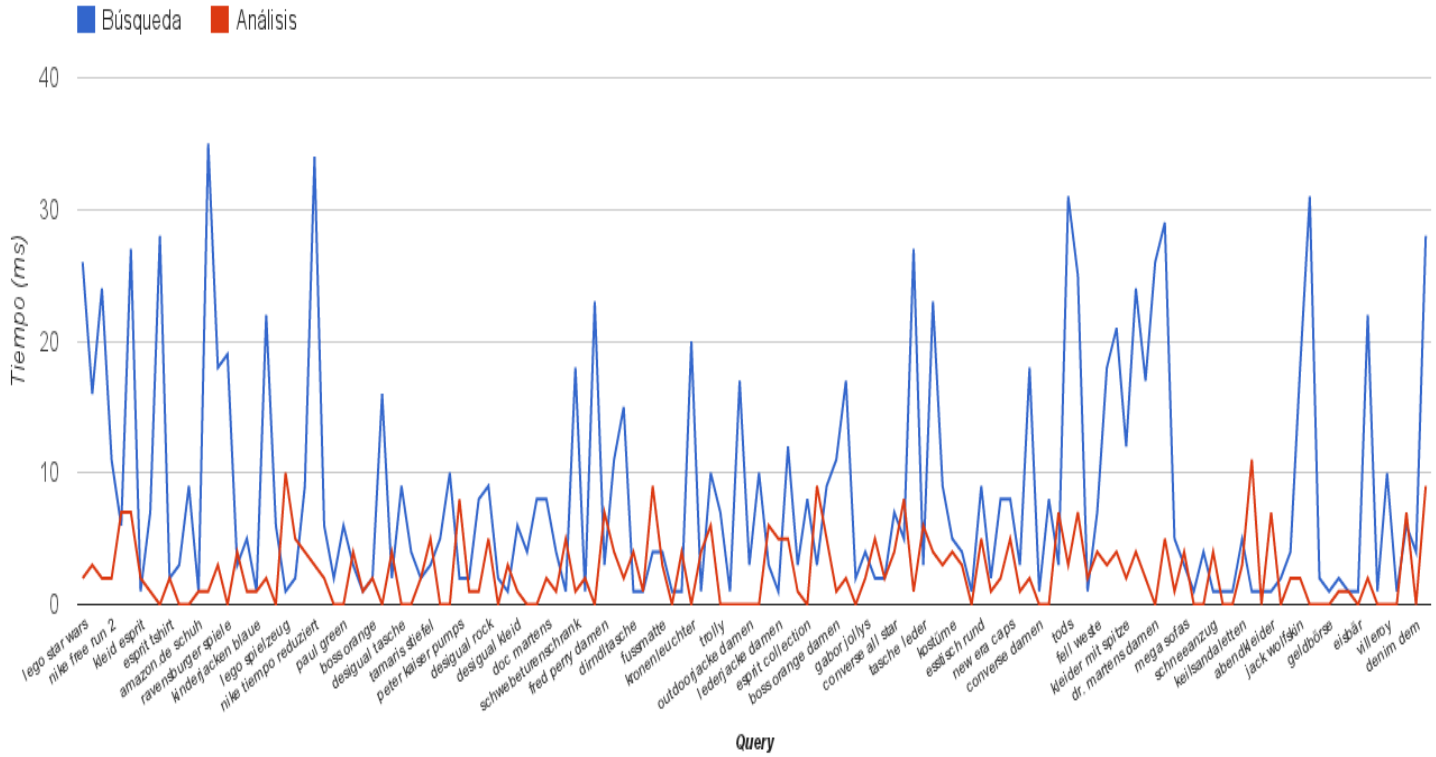
Como se preveía en el estado inicial, las pruebas realizadas en esta fase ponen de manifiesto que la mayor parte del tiempo de ejecución se debe a la fase de búsqueda en el índice.

Pese a que lógicamente, tal y como era esperado de antemano esta fase necesita un mayor tiempo de ejecución al tener una mayor carga de datos a manejar, su lentitud en ejecución de las consultas consideradas lentas en la fase de pruebas anterior condiciona el tiempo total de ejecución, ya que la fase de análisis se mantiene en unos valores similares a los del resto de consultas.

3.3 Estado posterior a la implementación de mejoras en el núcleo de Lucene

C1	Sistema antiguo	Nuevo sistema	
		Búsqueda	Análisis
Tiempo medio (ms)	6.621	8.457	2.471

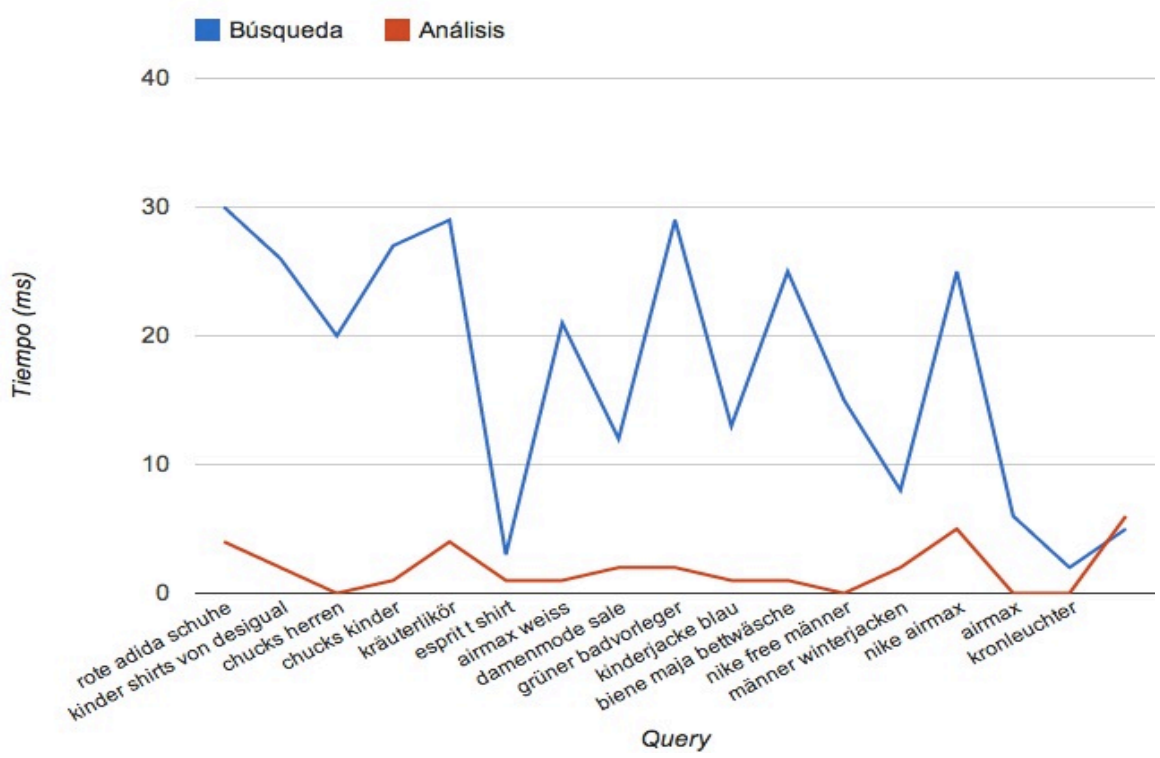
Tabla 12 - Tiempo de búsqueda frente a tiempo de análisis C1 - Tras mejoras núcleo de Lucene



Gráfica 7 - Tiempo de búsqueda frente a tiempo de análisis C1 - Tras mejoras núcleo de Lucene

C2	Sistema antiguo	Nuevo sistema	
		Búsqueda	Análisis
Tiempo medio (ms)	9.294	14.444	1.777

Tabla 13 - Tiempo de búsqueda frente a tiempo de análisis C2 - Tras mejoras núcleo de Lucene



Gráfica 8 - Tiempo de búsqueda frente a tiempo de análisis C2 - Tras mejoras núcleo de Lucene

Interpretación de los resultados

Nuevamente, de forma análoga a la fase anterior, queda de manifiesto el cuello de botella sobre el tiempo de ejecución provocado en la fase de búsqueda en el índice.

4. Conclusiones derivadas de las pruebas

Tras la finalización de las pruebas realizadas, las cuales han sido ampliamente detalladas en el presente capítulo se pueden extraer una serie de conclusiones generales que se listan a continuación.

Por un lado, respecto a la tasa de aciertos obtenida en los conjuntos C1 y C2, el nuevo sistema es claramente superior al sistema antiguo, ya que presenta un mayor porcentaje de aciertos ante todo tipo de *queries* de entrada.

Esta tasa con valores cercanos al 95% de acierto cumple totalmente con los requisitos iniciales, e incluso supera las expectativas previstas, alcanzando un valor muy difícil de superar pues el 100% es prácticamente inalcanzable debido a que es imposible controlar todas las posibles ambigüedades o errores tipográficos que pueden surgir.

Por otro lado, el punto negativo del nuevo sistema en comparación al antiguo radica como ya hemos visto en el elevado tiempo de ejecución para determinadas *queries* concretas, debido al cuello de botella originado por el módulo de búsqueda de documentos en el índice; que pese a que en una mayoría de los casos de prueba no ocurre así, condiciona el rendimiento global al alcanzar valores inasumibles en un sistema que precisa de interacción con el usuario en tiempo real.

Planificación y presupuesto

1. Planificación

1.1 Tareas

La planificación del proyecto se subdivide en las tareas mostradas a continuación:

1. Estudio del estado del arte del sistema de indexación y búsqueda de documentos existente.
2. Estudio de literatura actualizada sobre la tecnología Apache Lucene.
3. Análisis: identificación de problemas a resolver con el nuevo sistema.
4. Diseño: propuesta del nuevo sistema de sugerencias y medidas a tomar.
5. Implementación: codificación del diseño propuesto.
6. Pruebas y depuración: realización de pruebas unitarias y depuración de los errores encontrados.
7. Experimentación: realización de pruebas sobre el sistema a partir de los conjuntos de prueba suministrados.
8. Documentación: registro escrito de la evolución de la investigación y de los aspectos que la definen.

1.2 Recursos

Damián García García, autor de este proyecto de investigación, realizó labores de analista programador. Se considera una jornada laboral de 40 horas semanales.

1.3 Diagrama de Gantt

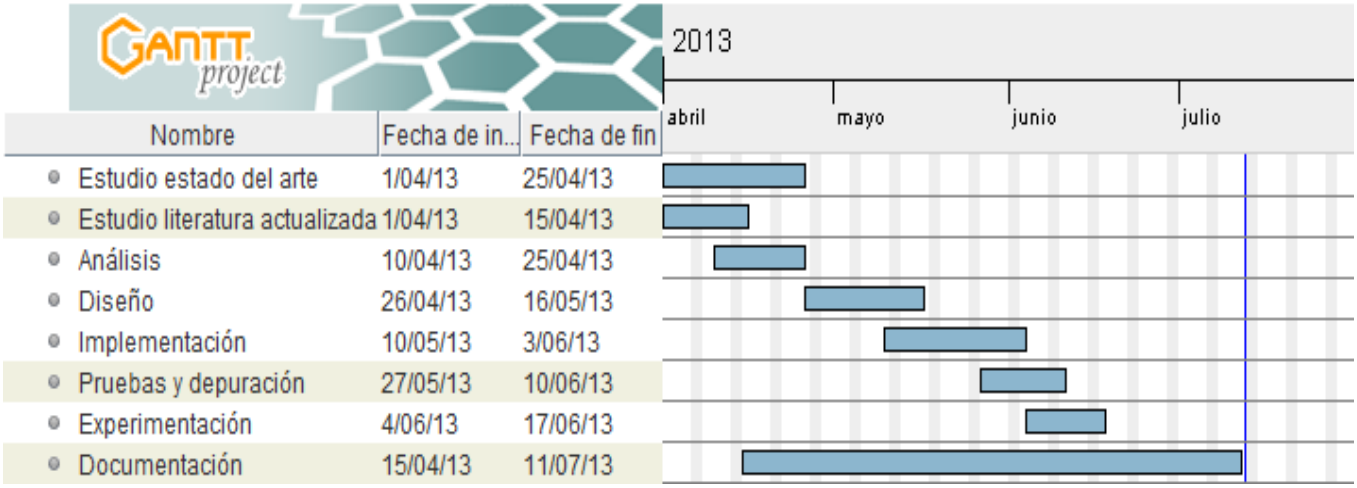


Ilustración 11 - Diagrama de Gantt

2. Presupuesto

En este apartado se presenta una estimación de costes económicos de este proyecto de investigación desglosado en tres bloques:

1. Costes de recursos humanos
2. Costes de recursos hardware
3. Costes de recursos software

Los costes de personal se estiman de acuerdo al convenio Entgeltgruppe 10 del año 2013 que rige en el estado federal de Berlín, y el IVA aplicado es el que rige en Alemania actualmente al 19%.

2.1 Costes de recursos humanos

Engloba todos los costes derivados de la recopilación de información, estudio del estado del arte, análisis, diseño e implementación del sistema desarrollado, las pruebas y depuración de errores del mismo, y la experimentación y documentación a cabo.

PRESUPUESTO DE RECURSOS HUMANOS				
Tarea	**Recurso**	**Unidades (h)**	**Coste unitario bruto (€)**	**Coste total bruto (€)**
Estudio estado del arte	Analista programador	58	18.75	1087.50
Estudio información actualizada	Analista programador	34	18.75	637.50
Análisis	Analista programador	69	18.75	1293.75
Diseño	Analista programador	80	18.75	1500
Implementación	Analista programador	103	18.75	1931.25
Pruebas y depuración	Analista programador	41	18.75	768.75
Experimentación	Analista programador	52	18.75	975
Documentación	Analista programador	115	18.75	2156.25
			TOTAL	**10350**

Tabla 14 - Presupuesto de recursos humanos

2.2 Costes de recursos hardware

Costes relacionados con las necesidades hardware para la ejecución del sistema implementado.

PRESUPUESTO DE RECURSOS HARDWARE			
Descripción	**Unidades**	**Coste unitario (€)**	**Coste total (€)**
Equipo experimentación: Intel Pentium IV 4096MB	1	285.45	285.45
		IVA (19%)	54.23
		TOTAL	339.68

Tabla 15 - Presupuesto de recursos hardware

2.3 Costes de recursos software

El proyecto puede tanto desarrollarse como ejecutarse utilizando únicamente aplicaciones de software libre gratuitas, exceptuando el coste de la licencia del sistema operativo.

PRESUPUESTO DE RECURSOS SOFTWARE			
Descripción	**Unidades**	**Coste unitario (€)**	**Coste total (€)**
Licencia corporativa de Windows 7	1	309	309
		IVA (19%)	58.71
		TOTAL	367.71

Tabla 16 - Presupuesto de recursos software

2.4 Presupuesto final

PRESUPUESTO FINAL			
Descripción	**Unidades**	**Coste unitario (€)**	**Coste total (€)**
Recursos humanos	1	10350	10350
Recursos hardware	1	339.68	339.68
Recursos software	1	1	367.71
		TOTAL	**11057.39**

Tabla 17 - Presupuesto final

El coste total del proyecto asciende a ONCE MIL CINCUENTA Y SIETE EUROS CON TREINTA Y NUEVE CÉNTIMOS DE EURO.

Conclusiones

La principal conclusión al final del período concedido por la empresa para la realización del proyecto, y reflejada así mismo en la reunión final de presentación de resultados ante los jefes ejecutivos de la empresa, los *project manager* y el jefe de desarrollo *frontend*, fue la obtención de resultados satisfactorios y de un sistema que si bien no se encontraba todavía en un estado suficientemente consolidado, sí serviría como base para la implementación final del nuevo sistema de sugerencias, una vez resueltos problemas pendientes.

De entre dichos problemas pendientes, el principal, como ya ha sido visto en el apartado de pruebas comparativas radica en el elevado tiempo de ejecución, que tiene su origen en el cuello de botella originado por el módulo de búsqueda de documentos relacionados con las *queries* en el índice, mayormente implementado antes del desarrollo del presente proyecto y del cual se esperaba un rendimiento mayor.

Tras analizar las causas de este cuello de botella, se descubrió que el problema se debía principalmente al módulo de normalización de términos, el cual precisaba una implementación propia en lugar de seguir utilizando el código por defecto para ello de *Lucene*, ya que la reducción de términos a su raíz provocaba multitud de ambigüedades con términos que no tendrían relación alguna con una determinada consulta pero que eran incluidos por ello entre los resultados de la búsqueda, incrementando considerablemente su número y por ende el tiempo de iteración sobre los mismos.

Este nuevo sistema de normalización requería la aplicación de ciertos criterios semánticos y sólo posibles de realizar por una persona cuya lengua materna fuese el alemán, por lo que la continuación del desarrollo por mi parte carecía de sentido y pasó a ser responsabilidad de un compañero alemán al cual yo asesoraría a la par que realizaba otro tipo de tareas en un proyecto distinto.

Por último, y ya desde un punto de vista personal, el presente proyecto me sirvió para descubrir y trabajar con un interesante sistema como *Lucene* del que no tenía conocimientos al inicio, afianzar mis habilidades con *Java* y adquirir nuevos conocimientos sobre un tema tan fundamental para un mundo que evoluciona hacia una cada vez mayor cantidad de datos a gestionar como es la indexación de documentos.

Dificultades encontradas

Durante el desarrollo del proyecto se encontraron ciertas dificultades de diversa índole, buena parte de ellas derivadas de la falta de nivel de conocimiento del idioma alemán, aspecto bastante recurrente teniendo en cuenta la necesidad de tratar cadenas de texto.

Aparte de los problemas lingüísticos, surgieron otra serie de problemas técnicos que hicieron que en determinadas fases del proyecto el desarrollo fuese algo más lento debido a la necesidad de documentarse o de realizar adaptaciones del sistema.

Respecto a este segundo aspecto, los problemas provinieron por un lado del hallazgo de diversos errores en el motor de Lucene sobre el que se sustenta todo el sistema de sugerencias, el cual se daba por probado y finalizado, y por otro de la necesidad de adaptación al desarrollo de cambios y nuevas funcionalidades desplegadas por otros compañeros en el sistema durante el propio desarrollo del mismo.

Además el desconocimiento personal acerca de *Lucene* implicó un gran número de horas de documentación e investigación al respecto previas al inicio del proyecto.

Por último, la complejidad del sistema de estructuración del árbol de etiquetas unida a la carencia de documentación sobre él provocó varias correcciones en el código fuente a lo largo del proyecto.

Bibliografía

Documentación oficial del proyecto Apache Lucene (http://lucene.apache.org/)

Gospodnetic, Otis; Hatcher, Erik; McCandless, Michael (2009). *Lucene in action*. Manning publications.

Konchady, Manu (2008). *Building search applications: Lucene, LingPipe and Gate*. Mustru publishing.

Amón, Iván; Jiménez, Claudia (2010). *Funciones de similitud sobre cadenas de texto: una comparación basada en la naturaleza de los datos*. Universidad Pontificia Bolivariana; Universidad Nacional de Colombia.

Fang, Hui; Zhai, ChengXiang (2009). *Evaluation of the default similarity function in Lucene.*

Naber, Daniel (2007). *Apache Lucene: searching the web and everything else.* International conference on Java technology Jazoon07.

Schreiner, Eric (2008). *High speed searching with Lucene: a practical introduction to the programming of Apache's Lucene framework*. Colorado software summit.