MetaSearch – Final Report

Patrick Moorehouse

Student Number: Omitted

Email: Omitted

Introduction

A meta-search is a type of search tool, which is used to find information on the subjects a user submits. What makes a meta-search different to other search engines like Google or Bing, is that it sends user requests to several other search engines and aggregates the results into a single list, or it can also display them according to their source

Why perform a search in this way? The reason a meta-search engine is effective is because the internet is so vast that no single engine can possibly produce perfect results all the time. They may miss some pages that are relevant to your search, and also may return some that are not relevant. By combining results from multiple search engines, we increase the chances of getting more relevant documents. The results are also ranked according to how many search engines returned a page, and what ranking each of those engines gave it, so that the most relevant pages are displayed first.

This premise forms the motivation for this project. A meta-search engine would be built that uses three search engines, Google, Blekko and Bing, to produce better results than any of those three on their own could return. A web interface would be built that will allow users to enter their queries, which would then be forwarded to the three mention search engines's API's (Application Programming Interface), and the results gathered through these API's would then be combined and ranked so that the best results return would be displayed first, and low ranked pages returned by fewer engines would be lower down in the list.

Further aims included incorporating query expansion, using a thesaurus API to return alternative search terms to a user based on the query submitted. Complex searching, using Boolean operators, such as AND, NOT and OR, to improve the accuracy of results and give the user greater control. A user experience review, with a survey page built into the website that would connect to a mysql database in order to save user submissions, which would then be viewed on the website via another webpage which connects to and loads data from the mysql database. An about page would be included which would give information on some of the terms used throughout the site, such as aggregation and query expansion, to aide users and make using the website easier for them.

The design and look of the website was intended to be kept simple and clean, taking inspiration from Google, whose interface is very simple, and sets a standard for usability. By keeping the design very simple and clean looking, and easy to use, it was intended that the search engine would appeal to a wide array of users, but perhaps primarily to those who would value higher accuracy and precision in their results.

Functional Requirements

There were a number of requirements set for the completion of the web application. A number of these were optional extras, which would aid in achieving higher grades. The basic objectives were mandatory however. These included;

Mandatory Features:

- Build a basic functioning version of a meta-search engine which aggregates search results from 3 different search engines.
- Complex Search ability: Allow users to use Boolean search operators such as AND, NOT and OR, accounting for the different ways different search engines handle these kinds of queries.
- Web based user interface: A website that can take in user queries and displays the query results back to them. Should aim for ease of use in performing searches, reading and navigating results, and applying search options
    - Allow users to choose between two methods of displaying results.
        - Aggregated: Should implement a form of aggregation of your choice, which should then produce and display a list of aggregated results, combining the results from all 3 search engines into a single list.
        - Non-aggregated: Should allow the user the option to display results as separate lists for each engine. This would mean a list of results from Google, a second list for Bing, and a third list displaying Blekko's results.
- Evaluation: Report on the performance of the search engine, using a variety of Information retrieval evaluation metrics. 50 queries were supplied which were to be used as search terms while evaluating the engine. A list of golden results for these 50 queries was also supplied, against which the search engines results should be compared.
    - The top 100 results should be compared against the 100 results for that query in the golden results.
    - Report on the following metrics: Precision, Recall, F-Measure, Precision@10, Average Precision, and MAP.
    - Evaluation based on both display methods should be carried out. This means evaluating the performance metrics of the aggregated results, as well as the results for each of the three separate underlying search engines. This means four sets of evaluation data should be acquired.
    - Perform some statistical analysis on the results obtained to demonstrate whether the difference in evaluation scores is significant or not.

Additional Features:

These are optional extra features that should be included to attain a higher grade.

- Extend the display engine to include clustering of results. Each cluster should represent a different facet of the query. An example of this is the search for "Jaguar". Clustered results should display results pertaining to the car manufacturer, Jaguar, in one cluster, while another cluster might contain results relevant to jaguars, the cat.

- Add query re-write functionality. This functionality should expand the original query to improve search results. This feature should be evaluated to ascertain whether the results were improved by use of query expansion or not.
- Run a user evaluation, with approximately 20 subjects (users). This evaluation should assess user satisfaction of the overall meta-search system, from their experience with the interface; to their rating of the quality of results they obtained using the search engine. The two main aspects are the sites appearance and usability and their experience using it, and the functionality, regarding how the found the quality of results.

Outline of Mandatory Features

Build a basic functioning version of a meta-search engine.

To build a basic functioning meta-search engine which gathers results from three search engines to produce an aggregated list, the first step would involve building a web page with a search form. This form would need a text box where a user enters their query, and then there would need to be buttons or toggles to switch between the display options (aggregated or non-aggregated) or turn certain features on or off (query expansion). Once this form is submitted, the query would need to be processed, and then forwarded to each of the search engines APIs

Query processing would involve modifying the query to support complex searches, or sending to a thesaurus API to retrieve synonyms and similar words. To prepare a query for complex search support, the query would need to be modified to adhere to the varying rules for each engine. For example, both Google and Bing support the keyword NOT, but Google does not support the keyword OR, as it uses the symbol | instead. These factors need to be considered so that each engines receives a query with complex search operators it understands.

Once the query has been processed, it is then ready to be forwarded to the search engine APIs. This is done by using the cURL function. The APIs send back the results for the query in a form that would need to be decoded. They mostly offer different encoding options, such as XML or JSON, but JSON is the format that would be used for this project. The JSON object would need to be decoded so the information could be accessed.

Once this information has been decoded, it must be stored in memory for efficient processing. This would need to be done by looping through the JSON data, and storing each element of the data into an array. There are different types of array to choose from, numerical or associative, 2D or multidimensional. One would need to be chosen that best suits the requirements for the application.

Once the data for one search engine is loaded into an array, the other two engines would need to be stored in memory also. One all of the three engines results are loaded, they can then be processed. To make this easier and more accurate, some form of processing on the data would need to be performed. Since different search engines may return varying versions of the same site address, for example, http://example.com or www.example.com, the URLs would need to be normalised. The best way to do this would simply be to clean the URLs as the JSON data was being loaded into arrays. To perform aggregation, the web pages also need some kind of ranking or score. There are many methods of scoring, so one must be chosen, and ideally applied to pages as they are being loaded into the array.

Once the JSON data is loaded into memory, and the pages have had a rank applied, they are then ready to be aggregated. Aggregation requires comparing the pages from one search engines results against the results of another engine, finding where there are duplicates, and summing the score, such that each site only appears in the final data set once, but the score it was assigned based on its rank for each of the engines that returned it should be summed.

When a list of websites has been produced, with no duplicate entries, and each page has had its score across multiple engines summed, the array can then be re-ordered according to the scores. This means that the sites returned by the most engines and that are the most highly ranked in each of those engines appear at the top of the list. Pages that are returned by only one engine and are not highly ranked will appear much further down the list. This is the desired outcome of the aggregation process.

Complex Searching

As stated, each of the underlying engines has their own rules regarding the application of complex search operators. Therefore the input query would need to be processed, and each engine API should receive its own version of the query. The processing would involve replacing a key word, such as NOT, with whatever symbols or keywords an engine supports.

Web based interface: Website

A website would need to be built, that takes in a user query, processes it, sends it out to the underlying search engine APIs, retrieve the information, and display it back to the user. This would mean a main page with a search form and various control objects such as radio buttons for selecting search parameters. The site would need a number of webpages, to compliment the main search page. These would include a page for user feedback, a page for evaluation information, and possibly a page that is capable of running designated queries and comparing results against the gold standard in order to produce a set of evaluation metric scores. While not specifically stated that an about page is required, it may be beneficial to include one with some information to aide a user's understanding of the principals involved in a meta search and the meaning behind some of the terms that appear on the site, in order to improve their user experience and make the site easier to use and understand.

The website would need a design style that lends itself to ease of use and clarity of presentation. The user should be able to select a display option, such as aggregated or non-aggregated. This would mean some mechanism such as radio buttons or checkboxes, to toggle or select display options.

Aggregated display should show the combined list of results from all three engines. Non-aggregated should display all three engines separately. A choice would need to be made on how these datasets would be displayed, such as in three columns, or in a list with one following another. It should be easy for the user to find and view the search results they are looking for.

Evaluation

To evaluate the performance of the system, a list of 50 pre-determined queries would need to be passed through the aggregation system, and scores calculated based on the required IR metrics. This would mean storing a file with the 50 queries. The file would then need to be opened, and loaded

into an array. The golden results would also need to be loaded from a file and stored in an array. After the query list and golden results have been put into arrays, a loop can be run which iterates over the queries array, and for each query, the usual aggregation process performed. Once the aggregation process has been run for a query, a loop would iterate over the search results, and compares each entry against the golden results list to see if it is one of the relevant documents. At this stage, and for each element in the results array, the average precision must be summed up. At the 10^{th} element of the results, the precision @10 can be calculated. Once all the results in the array have been assessed, the Precision, Recall and F-measure can be calculated.

This process is repeated for each query in the queries array. At the end of each query processing, the Average precision should be summed, as well as the average F-measure, so that the MAP (Mean Average Precision) and average F-Measure across the 50 queries can calculated.

This evaluation should be carried out for both display types, aggregated and non-aggregated. Non-aggregated evaluation should include separate performance tests for each of the three engines, resulting in four sets of data, a set for the MetaSearch engine, and sets for Google, Bing and Blekko. Since each of the types of display options both require 100 results per query from each of the three underlying engines, this means that to carry out both evaluations, each of the three engines would need to run through the 50 queries twice. Google only allows 10 results per request, so this means for each query, 10 requests must be sent to Google. Bing allows 50 results per request, so 2 requests must be sent for each query to Bing. Blekko allows the user to set the amount of results, so only one request need be sent to Blekko. This means that for 50 queries, there will be 650 requests sent to external APIs, with 15000 results retrieved for processing. Since this has to be performed twice, once for aggregated and once for non-aggregated, this means to run the evaluation once for each display option, a total of 1300 API requests must be sent, and 30000 results retrieved and processed. These figure do not account for other API requests, such as to a thesaurus, or and of the data sent or received during general building and testing, or during user experience reviews.

The performance metrics alone should not be considered conclusive. Some statistical analysis would need to be performed to ascertain whether the findings of the evaluation are statistically significant, or whether they may have occurred by random chance. The suggested method of analysis was a Student T-Test. Students were advised that the Wicoxen Signed Rank Test was not suitable for statistical analysis on information retrieval evaluations.

Outline of Additional Features

Clustering

Clustering involves extending the display options to allow results to be grouped together in clusters, each of which represents a different facet of the query. This is a complex task, requiring that the snippets from each site be analysed, and for clusters to be generated based on that analysis.

Snippets would need to be processed. The first step of this process is stop-word removal. Since the analysis seeks to find terms of relevance, words like “the”, “it” and “that” should be removed, as they are highly frequent in any text and do not help to categorise the text it came from. The snippets are analysed, and an array is created to hold information on the words that appear in snippets and their frequency of appearance. This information acts as vector, which each term in the document

representing a dimension. This is used to express the sentiments behind documents, and to group them according to these sentiments. For example, searching for Jaguar, one cluster might contain information on the car manufacturer, while another on the wild cat.

Query Rewrite/Expansion

This task involves some form of processing on the query that a user inputs, which aims to improve the recall of the engine. There are a number of alternative routes which can be taken to accomplish this. The query can be checked for spelling, or it can be stemmed, or synonyms and similar words could be retrieved and added to the query, or displayed for the user to choose whether or not to add them.

The query would be taken in via the interface, then sent to an appropriate API, and the returned data would then need to be decoded from JSON and processed or displayed as necessary, whether it would then be added into the search term or displayed for the user to choose from.

User Evaluation

A user experience evaluation should be conducted. This would require approximately 20 subjects to visit the website and test out the usage of the system. This involves searching a number of queries, and trying out the different search options and display methods.

A survey would then need to be given so that users can answer questions regarding their experieces using the search engine. The survey should evaluate their experience in terms of appeal towards the style and design of the site, as well as their feelings about the effectiveness and functionality of the engine.

The method by which users would be given a survey was not specified. The brief stated that a form would need to be devised to assess user satisfaction in various aspects of the website, however, it does not state whether the form should be built into the website, or if a third party survey provider would suffice. This leaves the method of survey delivery open to choice.

Implementation

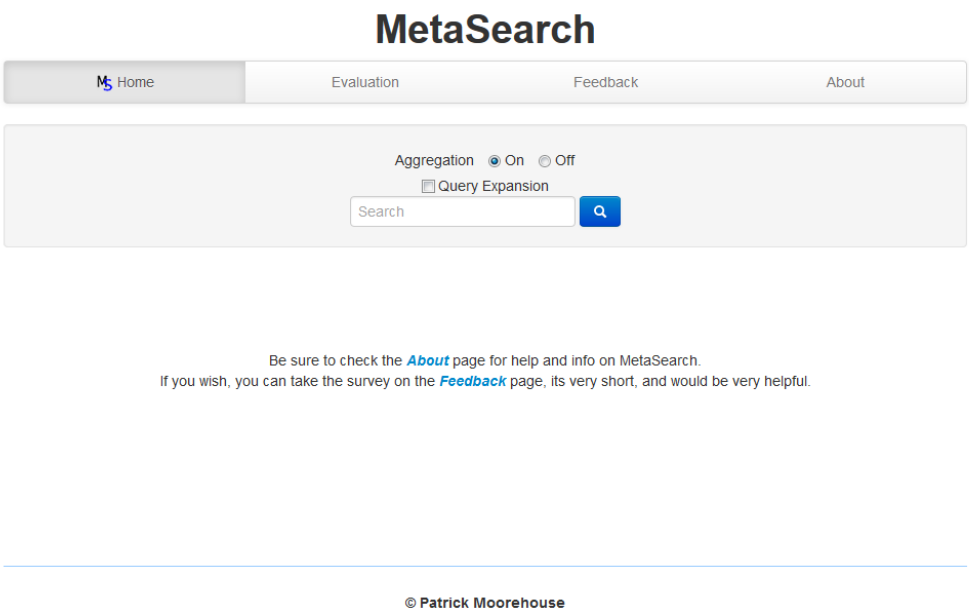
This section outlines the methods that were used to implement the required and optional features. I will begin with the general look, design and layout of the site. I will then discuss the aggregation technique, followed by the way in which the evaluation was handled.

Website Design

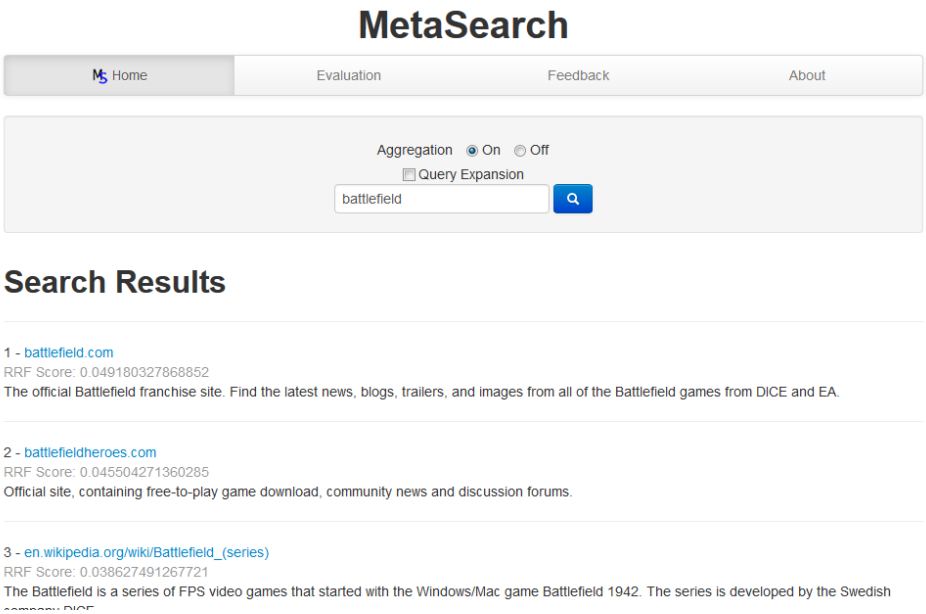
The design was intended to be as simplistic and clean looking as possible, which also adds to the ease of use. Few people would want a search engine that is obnoxious and in your face, so to speak. Therefore, the design of this site emphasised clarity, ease of navigation and control, and a pleasant way of displaying results.

I decided to use Twitter Bootstrap to help increase the pace of User Interface development. Twitter Bootstrap enables clean and effective designs to be used simply by specifying classes for your html tags.

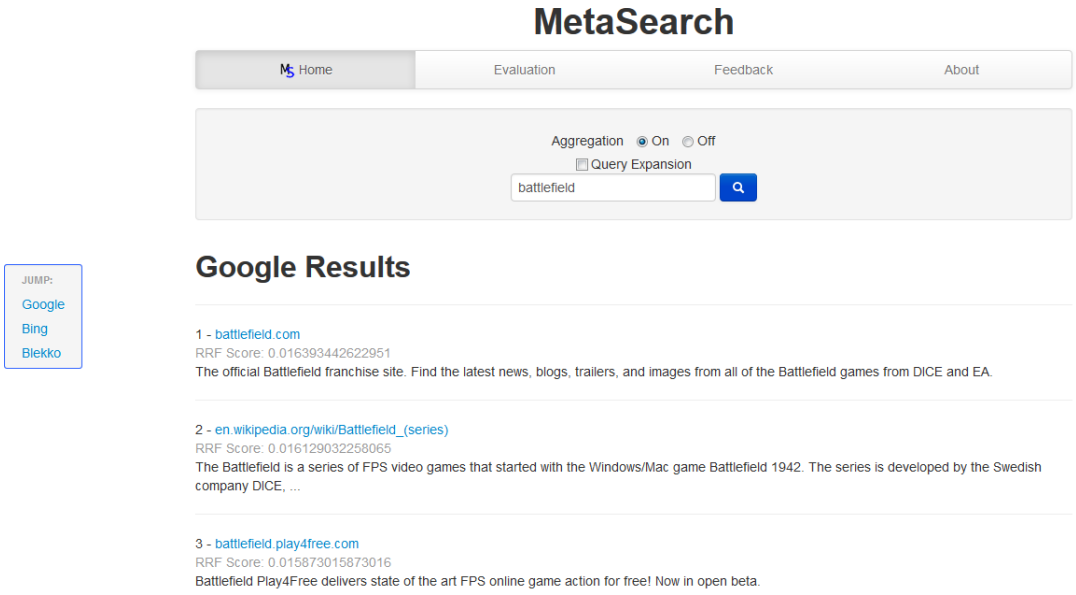
Below we see the design of the main page. The Design is simple and minimalistic. It simply consists of a header with the site name, MetaSearch, a navbar, a form for searching, and a footer. The form offers the choice to turn aggregation on or off via radio buttons, and a checkbox allows query expansion to be turned on or off. I added links on the homepage to the About page, so that users can go there to learn some more about the site and its terminology, and to the feedback page, where they can submit a survey if they wish. The design features mainly white and grey with small hints of blue in the colour scheme.



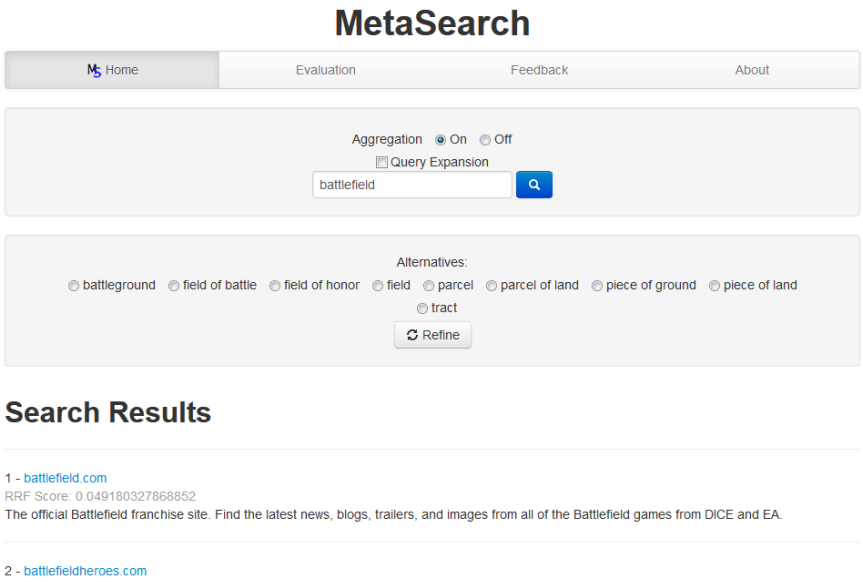
When the user searches for something, the navbar and search form remain in place. They are simply presented with the search results below the search form, so they can browse the results, or enter a new query. The display of the results is simple and uncluttered, the rank and the page URL are shown, and the URL can be clicked to navigate to the site. Below is the RRF score, but this is in a muted colour, so as not to distract users too much. I could have taken this out of the displayed results, but I felt it was useful to display the score that determined where pages were ranked, and I made it a link to the about page so curious users could investigate further. Below the RRF score is the snippet, or, description, of the website.



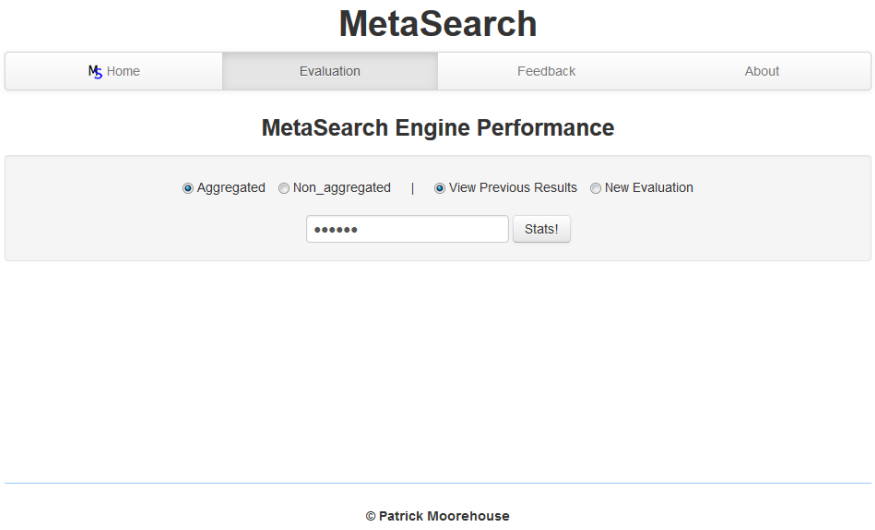
Next we see the search page when the non-aggregated view is chosen. There is a single column which shows each of the three engines results, one after the other. On the left can be seen a navigation tab that is shown only when the non-aggregated results are shown. This helps users jump to the desired set of results quickly and easily.



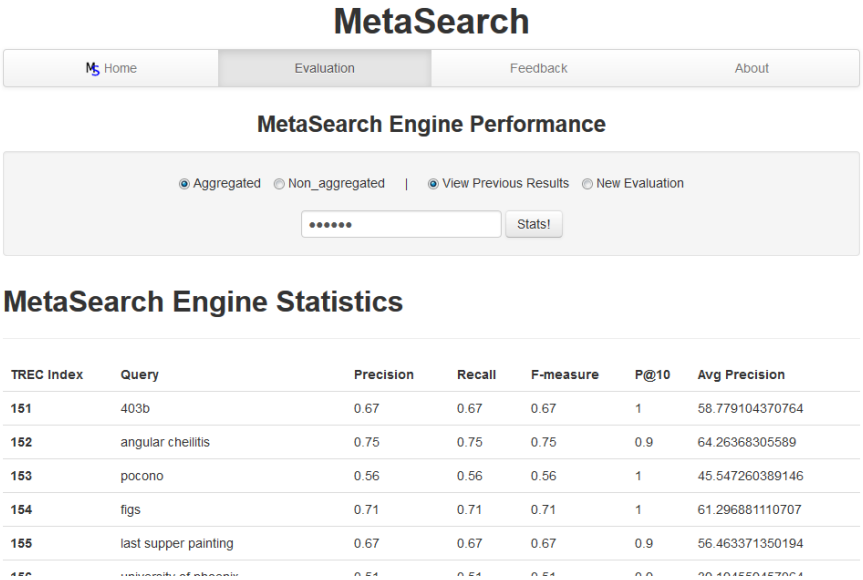
Next we see the search page when query expansion is turned on. The page loads with results, and below the search form a second form is displayed which shows a list of alternative synonyms and similar words or phrases. The user can choose one and click refine to reload the page with new results from the newly expanded query.



Next we see the evaluation page. This page has a very simple form that offers the user to choose between Aggregated or Non-Aggregated results evaluation. To save on the amount of queries being sent to the search engine APIs, I included the option to either run a new evaluation, or to view previous results. New evaluations send a lot of requests, 650 in fact, and retrieve 15,000 results and then analyses them. For this reason, it would be impossible to allow any user to run evaluations as they please. So a password form was added so that only those who have been given the password (such as lecturers etc) can run new evaluations. The previous results is simple the html file that was saved when a previous evaluation was completed, which is included in the evaluation page with a php include, to display past results. This allows users to quickly see the performance scores of the various engines instantly, even if they don't have permission to run new evaluations.



This is an example of how the performance test results look when an evaluation is complete.

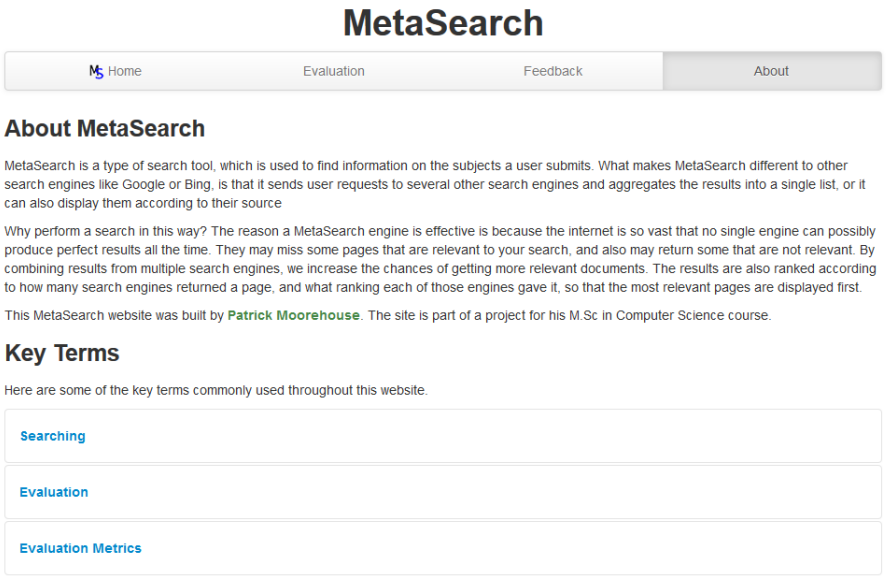


The following shows the simple form for user feedback. It asks a few questions on their experience using the site. When complete, the user can click submit and their answers are stored in the database.

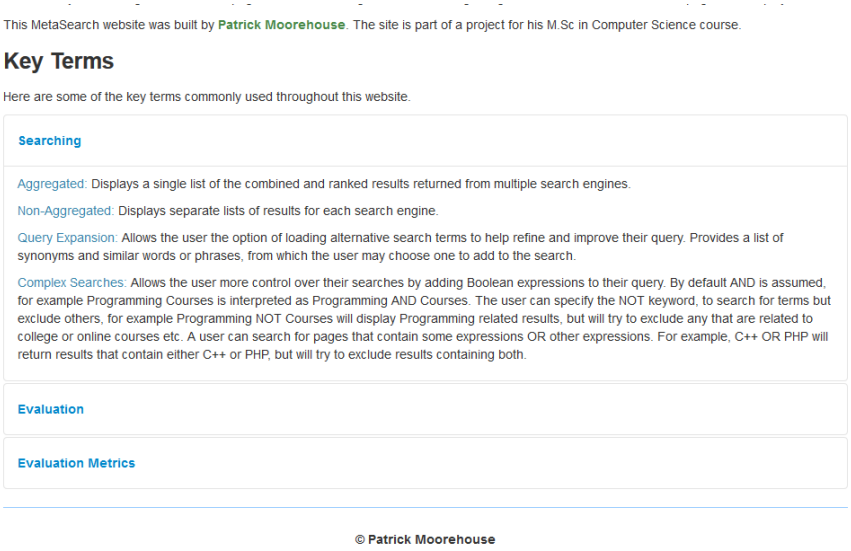
When the user clicks submit, they get a brief thank you message before being redirected to the homepage.

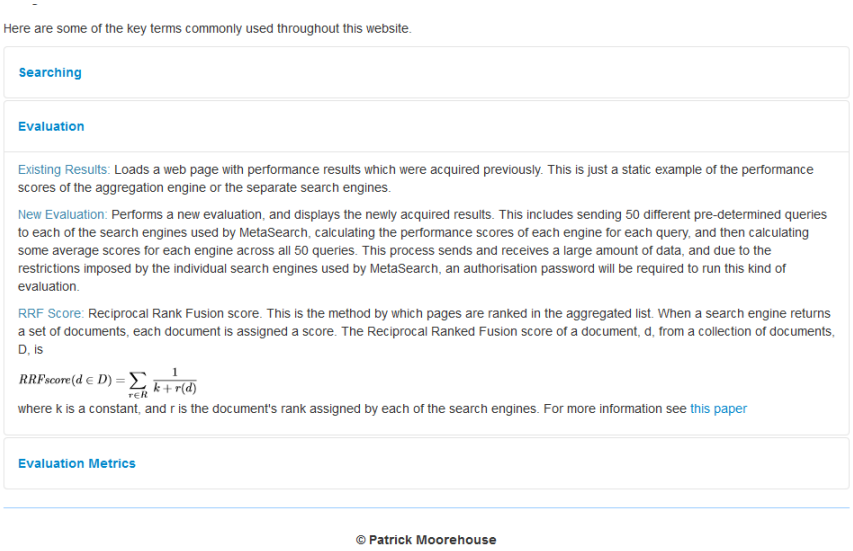
Thanks For The Feedback!

The About page offers a brief explanation of what the site is about, then offers some extra information on the key terms used throughout the site. The extra information and key terms is hidden behind collapsible regions. This was done to add as much information to the page while minimising clutter. It allows the users to expand these menus so they can read the information if they choose.



Here we see the collapsible menus.





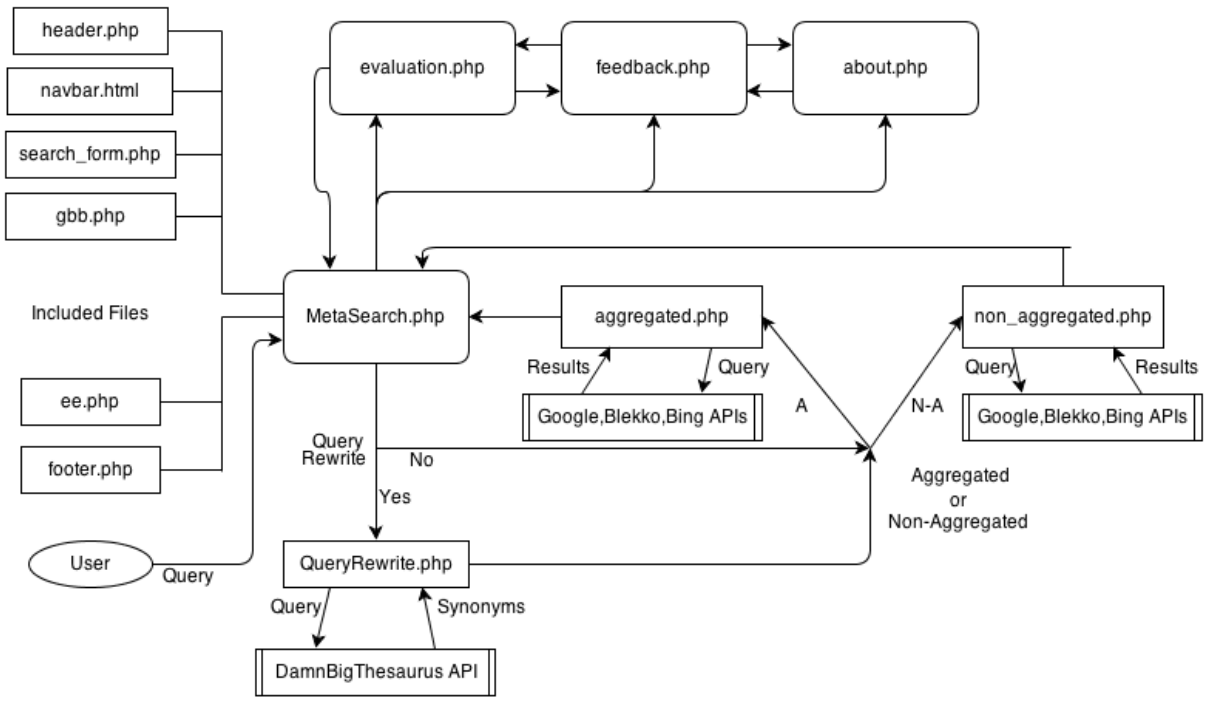
Site Structure

Here I will show a number of site architecture diagrams and explain how the site is navigated, and what pages are loaded when a user performs an action.

There are four main pages, MetaSearch.php which is the main page, evaluation.php, feedback.php, and about.php. These are the main pages and any one can be reached from any of the others by clicking its link in the navbar.

The following diagrams show the site architecture, focusing on one of the main pages at a time. The other pages are shown, and their link noted, but their only the components of the page in focus are shown. This is to save clutter and confusion in understanding the structure.

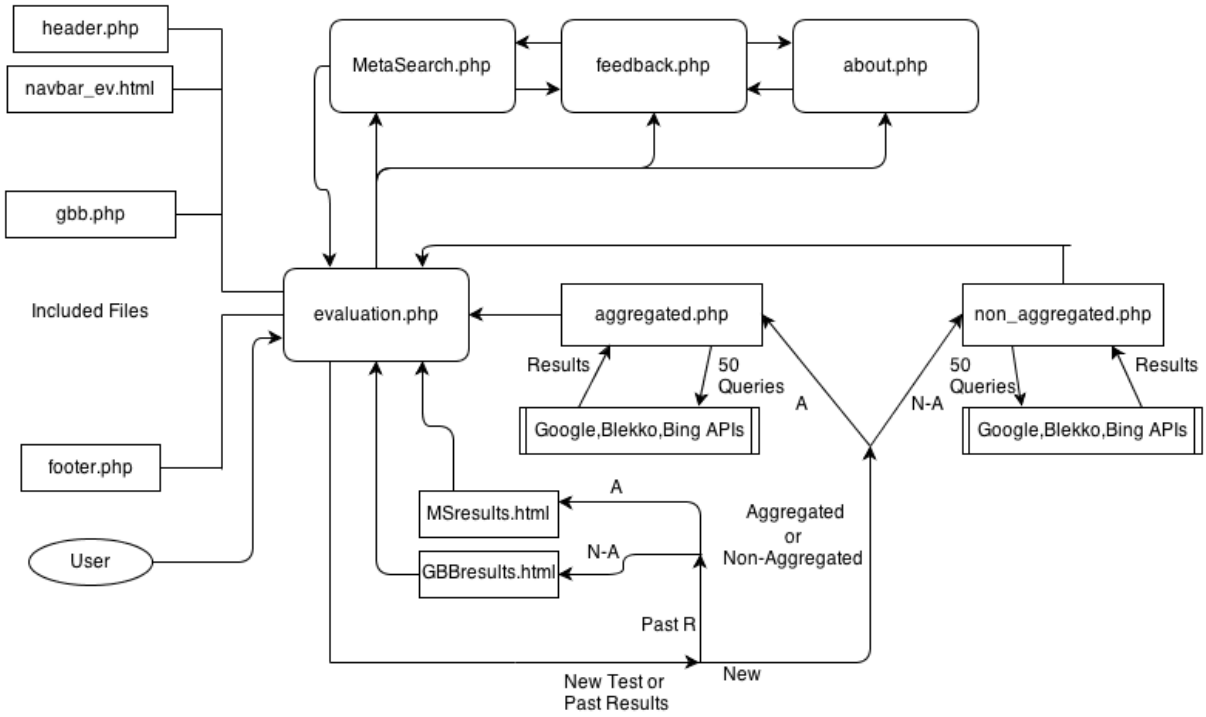
MetaSearch.php (main page)



The MetaSearch.php is loaded, and all the files it requires, header.php, navbar.php, search_form.php, gbb.php (a php file containing the search API keys), ee.php (a small php where easter eggs related to the query submitted can be entered), and footer.php, are included with a php include statement.

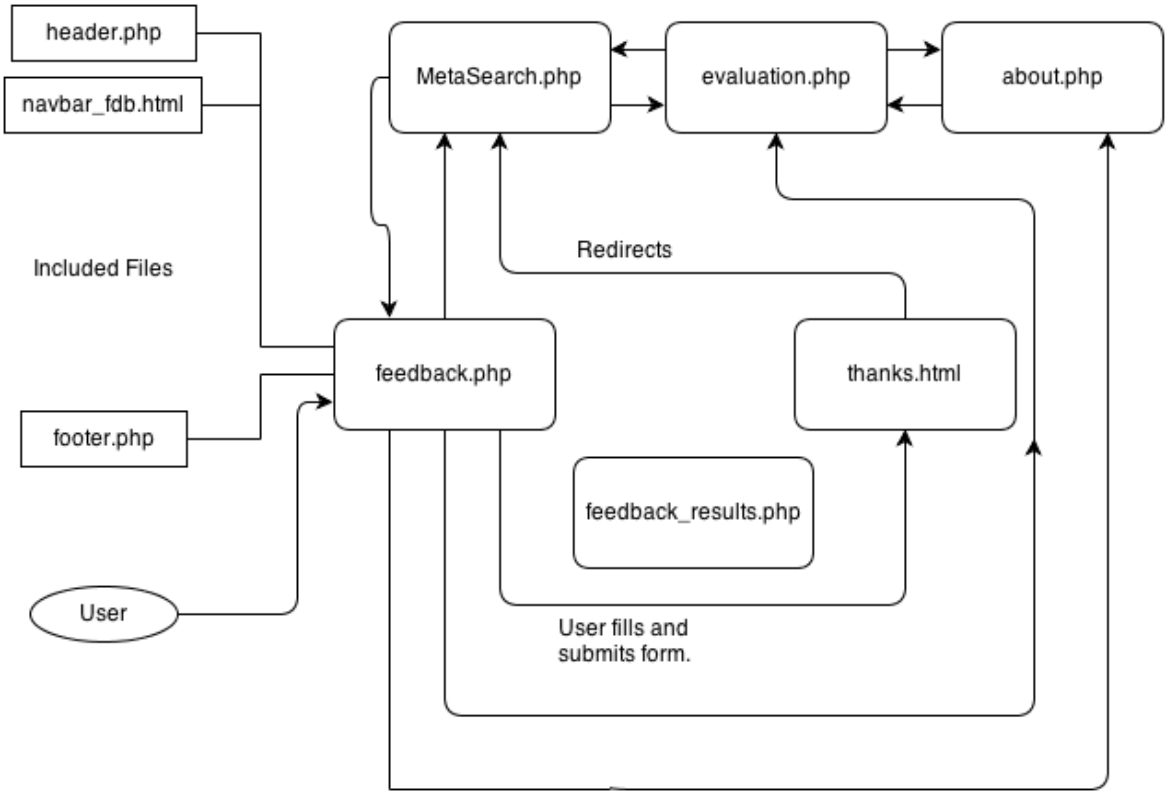
The user enters their query in the search box and submits. First, on page load, the page checks if there is any POST information. POST information indicates that a query has been submitted. Then, it is checked whether the user has Query Expansion/Rewrite turned on. If it is turned on, the QueryRewrite.php file is inserted, which then sends the query to the DamnBigLabs thesaurus API. A list of synonyms and similar words to the query are then returned. These are then listed in a form, as seen above. The page then checks if the user chose aggregated or non-aggregated display. Whichever the user chose, the appropriate file is loaded, and the query is sent to search APIs, and the results are returned. These results are then processed and displayed to the user.

evaluation.php



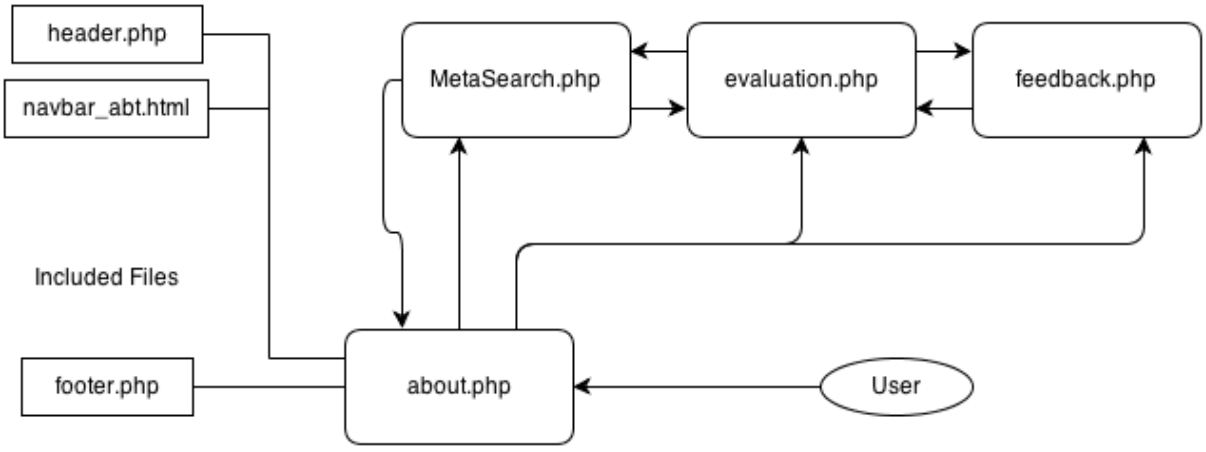
When the evaluation page is loaded, the user chooses whether they want a new test or to view previous test results. If the user chooses previous results, the page then checks if they chose aggregated or non-aggregated results. It then selects the appropriate html file that was contains the previous test results and this is then included into the evaluation page so it can be displayed to the user. If the user chose to do a new evaluation, it checks for password, if password is not set or is incorrect, the user receives a message to say that they did not supply the correct password. If the password is correct, it then tests whether to run the aggregated or non-aggregated results and the appropriate file is loaded. This file contains the code that runs the evaluation. This is explained in the Implementation section.

feedback.php



The feedback page simply displays a form where the user can answer a series of questions about their experience using the site. When they submit the form, the results are stored in the MySQL database, and they are then greeted with a thank you message, which then directs them back to the homepage. The feedback_results.php page is where the survey results are viewed. It is not linked to directly on the feedback page as it is not intended for public viewing. I was not deemed necessary to password protect the page however.

about.php



The about.php page does not run and functions or scripts, it simply displays information to the user.

Aggregation Technique

The aggregation technique used was Reciprocal Ranked Fusion. I had originally chosen to use combMNZ as the aggregation ranking method, however, upon further review, I discovered how simple it was to compute the RRF score, and it was also reported to produce better results. It also scored based on the rank of a page, but not on the number of results returned. This meant a page could be stored in an array as the JSON data from the API was being read, and its score could be assigned at the same time, regardless of how many results there were.

Reciprocal Ranked Fusion score is given by the equation:

$$RRFscore(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)}$$

where k is a constant, and r(d) is the rank of the document d. k is a constant that was found by Cormack, Clarke and Buttcher to work best at a value of around 60.

The algorithm works on the premise that documents that are not highly ranked, while rightfully attaining lower ranking scores, should not be given zero importance, like some algorithms do, in particular exponential algorithms. The presence of the constant k serves to minimise the impact of outlier engines giving specific pages a very high rank in comparison to other engines.

When a query was submitted in the search form, first the display method would be checked and the appropriate php loaded to perform the action. In the case of non-aggregated results, the query would be sent to the first API using the cURL function. The API key and parameters would be loaded from the gbb.php file, then the request url would be constructed with these parameters, for example;

```
$google_url =
"$rootUri?key=$g_key[$key_array_index]&cx=$cx[$key_array_index]&q=$query&alt=json&start=$st
art_num";
```

Next cURL would be initialised, the URL would be set with CURLOPT_URL option, cURL would be executed to create a variable with the JSON data stored in it;

```
$ch_g = curl_init();
curl_setopt($ch_g, CURLOPT_URL, $google_url);
curl_setopt($ch_g, CURLOPT_RETURNTRANSFER, 1);
$data_g = curl_exec($ch_g);
curl_close($ch_g);
```

The $data variable now holds the JSON data, and is decoded with the function;

```
$js_google = json_decode($data_g);
```

A foreach loop is then run for each element of the decoded JSON object, and the result information is appended to an output string, to be outout at the end of the script with an php echo command.

For aggregated results, the data was not simply printed. First the URLs were cleaned, removing any prefixes, and deleting the trailing / if one was present. Then the data was entered into an associative

array with the cleaned URL as the key value, and the snippet/description and the calculated RRF score as values associated with the URL.
This was repeated for each result, and this was done for all three engines. When data was to be entered into the results array, it was first checked if the entry already existed, with a simple test:

```
if(!isset($ResultArray[$goog_url]))
```

If the item did not already exist in the array, it was added, along with the snippet info and RRF score, if it already existed, then the existing score simply had the new score added to it.
This use of associative arrays with the cleaned webpage URLs as key values meant that duplicates were virtually non-existent; as they were eliminated as they were found, saving the need to remove them later.
When an array was produced containing the combined results, it was then sorted according to the RRF scores with the uasort function, and a custom comparison function.

```
function sortByScore($a, $b)
{
        if($a['score']==$b['score']) return 0;
   return $a['score'] < $b['score']?1:-1;
}

uasort($ResultArray, 'sortByScore');
```

Note the Ternary operator, ?, used here to return 1 for true and -1 for false, as the uasort uses integers, and the RRF scores are not integers, so a conversion is carried out.

Once the array has been sorted according to score, a loop is carried out over the array and the values printed to display the aggregated list of results.

Evaluation

The evaluation was performed by means of a php page that was written to run all 50 queries, through each of the 3 search engines, and comparing the results against the gold standard.

The first part of this process was to read the list of queries from a file, and store them in an array. This array would be what the main loop for the evaluation process would be based on. Then the golden results were also loaded into an array, read from a text file. The file contained a list, with the TREC index number, a space, and then the URLs of the results. Reading the file one line at a time, the php explode function was used to load both the TREC index and the URL into a file as separate elements of an array. The TREC was later used to identify which of the entries in the results file were linked to which query.

Once these two arrays were loaded into memory, the evaluation would begin. A loop from 0 to 49 would start, and would take the query from the queries array whose element array index matched the counter. For each loop of the array, and therefore, each query, the query would be processed for complex searching operators, and then sent to one of the search engine APIs using the cURL function.

Depending on the engine this sometimes needed another loop. For Blekko, no loop was needed as 100 results would be returned per request. The request would be sent once, results returned as JSON data. The JSON data would then be decoded, and a loop would begin, repeating for each element of the decoded JSON data. Each element of the JSON data is a result, and thus contains information about a website. The url of the site would be cleaned, removing any instances of www. http:// https:// and any variant of these, so that a site would be matched with its result in the golden set regardless of the different display methods of the search engine.

For non-aggregated evaluation, this is repeated for each entry in the JSON results, and metrics such as precision, average precision, recall and f-measure are calculated. For google this step would have to be repeated 10 times, with a parameter in the URL stating where in the results the starting point should be. This parameter was incremented by 10 for each iteration of the loop, so the results returned were pages 1-10, 11-20, 21-30 and so on. Bing had to be repeated twice, as Bing returns 50 results per query. The performance metrics that were calculated were then printed to screen. This was repeated for each of the 50 queries, so that a table was produced listing all of the scores for each query. This was repeated for each of the engines.

In the case of aggregated results, the scores were not calculated immediately. Instead, the results from the three engines were loaded into an array and aggregated and ranked first (see aggregation section for details) and then the process of described above would begin, loading the queries from file, loading the golden results, then looping for all queries and comparing the aggregated results against the gold standard, calculating metrics along the way.

Technologies Used

There were a number of technologies used, from the search APIs to the software used to build the web application. The following are the technologies that were used. According to market research

Search Engines and other APIs

Google

Google is a search engine created by Larry Page and Sergey Brin as Ph.D. students, which has grown to become the world's most widely used search engine. Google indexes so many web pages,

Bing

Bing is a search engine created by Microsoft. It has gone through a number of previous iterations, such as Live Search, Windows Live Search and MSN Search. Bing was launched in 2009, and is marketed by Microsoft as a "Decision Engine".

Blekko

Blekko is a web search engine which aims to provide better search results than Google Search by returning results from a set of 3 billion trusted webpages, and excluding sites which they call "content farms", which are sites that produce a large volume of textual content that is often taken from other sources and written with the goal of fulfilling search engine ranking algorithms, so that

their pages get more views and generate ad revenue. Blekko also offers a feature known as slashtags, which are used to improve the relevance of results.

Big Huge Thesaurus

Big Huge Thesaurus is an online thesaurus service provided by Big Huge Labs. It allows users to enter words and returns synonyms and similar words. Big Huge Thesaurus has an API that allows the retrieval of synonyms in your online applications.

Programming Languages

HTML

HyperText Markup Language is the primary markup language for writing web pages. In a html document, content on the page is wrapped in html tags, which dictate how the content behaves, and how it is displayed. An example of this is a heading tag, which increases the size of enclosed text, but also adds weight to the contents meaning. When a search engine indexes the page, it takes content within a heading tag to have special importance within the page. There is an opening tag and a closing tag, and the first heading tag is h1. An example: <h1>Heading</h1>

CSS

Cascading Style Sheets is a stylsheet language that describes how content in an html page looks. CSS was designed primarily to allow separation of a page's content and its presentation, meaning that only the content information for a page is stored within the page, and the CSS file holds the rules that govern how the content is displayed. Separating content from presentation allows for better accessibility, more control and flexibility over presentation, and less repetition of style across multiple pages.

PHP

PHP is a server-side scripting language designed for web development. PHP is reported to be installed on more than 244 million web servers world-wide. PHP was created by Rasmus Lerdof in 1995, however the reference implementation is now maintained by the PHP Group.

PHP stands for PHP: Hypertext Pre-processor, although it originally stood for Personal Home Page. PHP code is interpreted by the web server with a PHP pre-processor module, which then produces the html content that is delivered to the browser. PHP commands can be embedded within a html page.

SQL

Structure Query Language is a special purpose programming language designed for managing data held in a relational database management system. SQL was used in the project to create a table and alter its properties, as well to display table contents through PuTTY.

JavaScript

JavaScript is an interpreted programming language originally developed with client-side scripting within web browsers so that users could interact with the content of a page. It is the de facto scripting language of the web.

Software

DreamWeaver

Dreamweaver is a proprietary web development software developed by Adobe. It comes with support for Web technologies such as CSS, JavaScript, PHP and other server-side scripting languages. Dreamweaver offers a visual What You See Is What You Get (WYSIWYG) editor, though this was not used during this project as the code view was preferred. The code view editor offers syntax highlighting and code completion, making writing code faster and easier.

Notepad++

Notepad++ is a text editor and source code editor for windows. It is a lightweight editor supporting a wide variety of programming and scripting languages. I did not use Notepad++ to write pages specifically for the website, but I did make use of it as a simple and quick means of testing concepts. For example, when approaching the evaluation phase of the project, I used Notepad++ to write php pages that would allow me to enter the TREC index number of a query from the pre-determined queries we were given, and the page would then display the query. I also created a page which, when the TREC index was input, would list the 100 web address URLs in the golden results file relating to that query's index. These were not necessary for the project, but it helped plan my designs in advance, especially the components that would involve reading the data from the queries and golden results files.

FileZilla

FileZilla is an FTP client. It was not used for uploading files to the server as Dreamweaver has this built in, but it was used to easily change file permissions for files once they had been added to the server.

PuTTY

PuTTY is a free and open source terminal emulator. It supports networking protocols such as SSH and others. PuTTY was used to log in to the server via SSH in order to change the account password. It was also later used to import a table into a database from a previously created .sql file. It was also used to verify that the user evaluation form was successfully submitting data to the table in MySQL.

MySQL

MySQL is the world's most widely used relational database management system (RDBMS) as of July 2013. It acts as a server, providing access to many users to multiple databases. It was used to create a table within a database which was used to store user evaluation submission data, and so the information could be retrieved later through the website on the feedback_results.php page.

WAMP

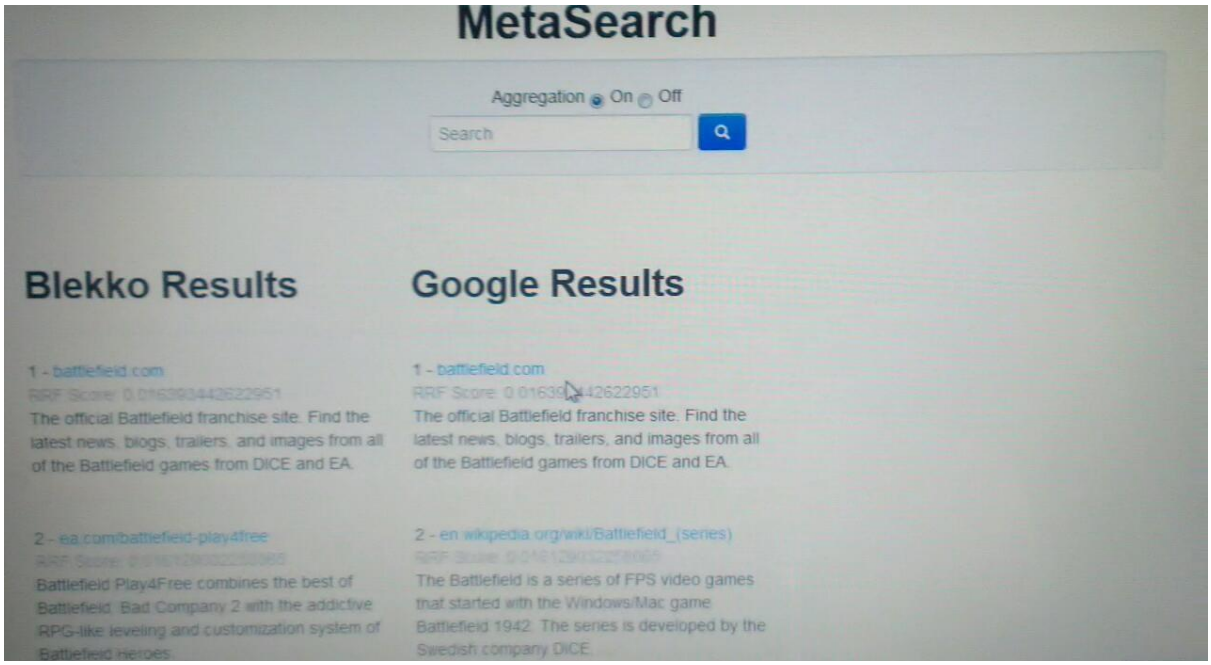
WAMP gets its name from its components. Windows, Apache web server, MySQL and PHP (alternatively, Perl or Python may be included). It runs on windows and allows the user to build and test websites locally. WAMP was used for some early building and testing, but later was used less frequently as new changes were uploaded to the live server for immediate testing.

Problems Encountered

The project went smoothly for the most part. However, there were a number of issues that occurred. The following describes these issues.

Non-Aggregated Results Display

My original design for the display of the non-aggregated search results included a column for each of the engines results, meaning three columns. When testing this however, I found that because of the way the code was written, a column would be displayed when the results for that engine were complete. This means that when the page was loaded, one column would appear, then the next, and sometime after that, the third would appear. This was very undesirable, as it gave the impression that the page was taking longer than it actually was. Even if the page takes the same time to load, if a user sees parts of a page loading at different times, it will give them the impression of slow performance.



In order to solve this, I looked at my code. In my loop, for outputting the results information, I was using echo statements to print out the data. This had the effect that data was printed as soon as possible; meaning that parts of the page loaded while other information was still being processed. To counter this problem, I declared an empty variable, then concatenated the results information onto this string. So instead of

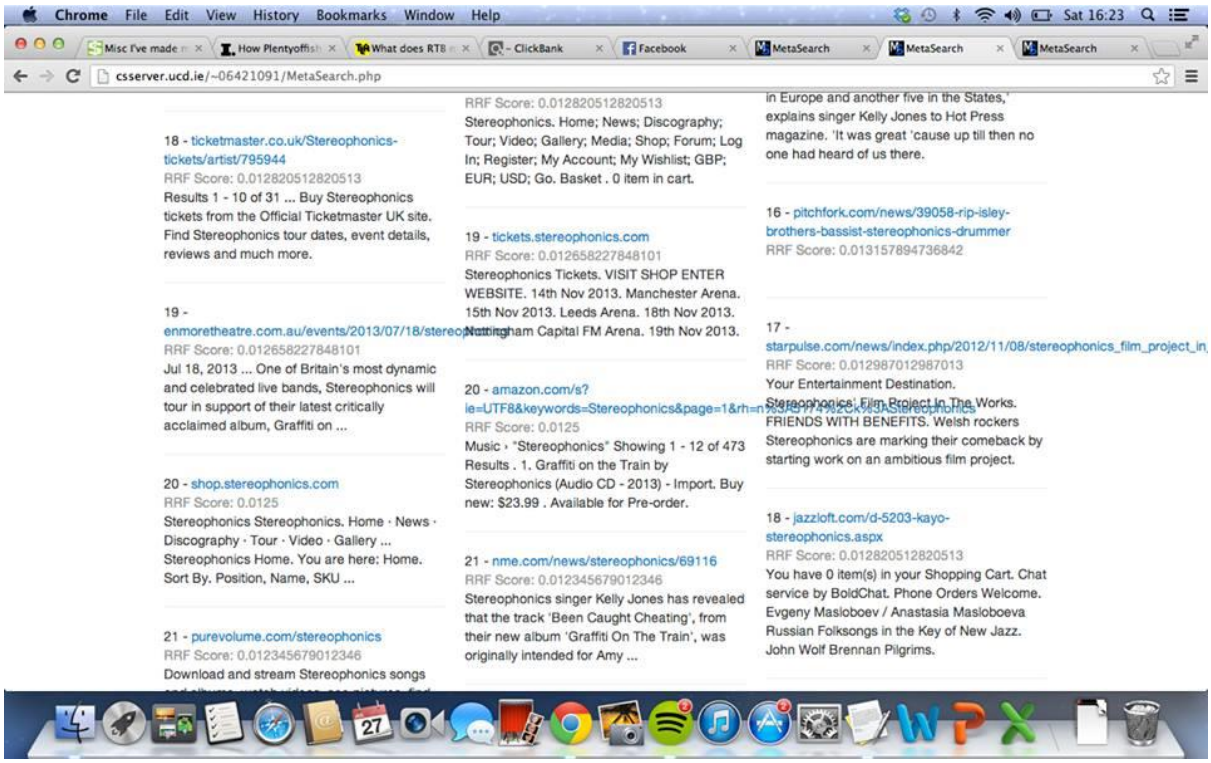
```
echo 'example';
```

I instead used

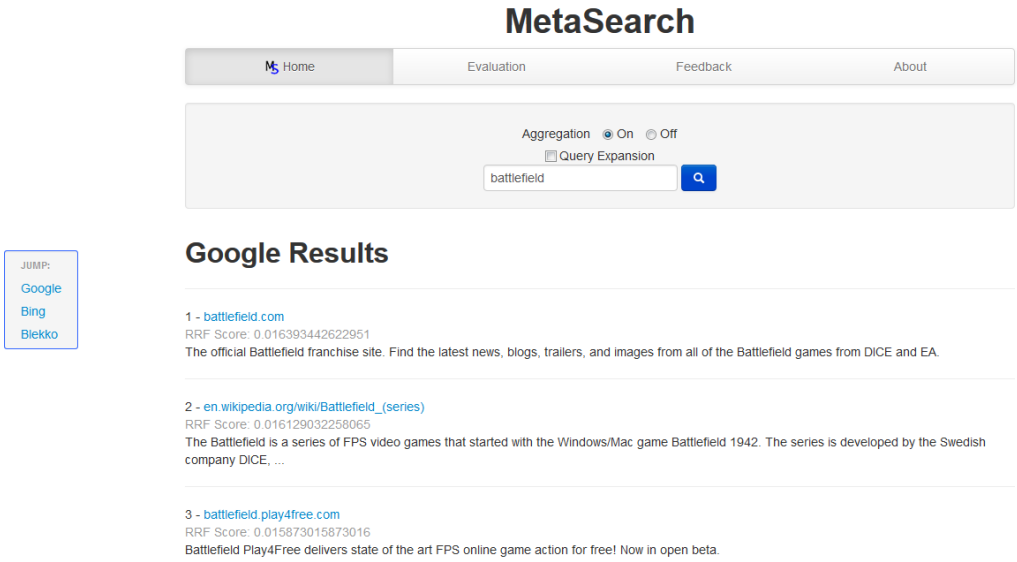
```
$output .= 'example';
```

Then when all processing was complete, the full page was output at once with `echo $output;`

This solved the problem of asynchronous column displaying. However, when I posted the site for user evaluation, a subject spotted a problem. He was using a Mac computer, with OS X operating system. The problem with CSS is that sometimes it behaves differently on different computers with different display resolutions or browsers. Even on the same computer, different browsers might display it differently. The problem the user pointed out was that links leaked out of their designated area and into adjacent columns. The following is a screenshot that he sent to me displaying the problem.



This was late in the project lifespan, so I did not have very much time left for solving problems, but also, I had posted the site for user evaluation, so I felt I had to act fast in order to prevent other users from experiencing this issue. I felt it was most likely that users browser and operating system combination that caused the problem, but I decided to chance the design to one that would not be likely to have the same problem, as poring over hundreds of CSS rules to find the problem seemed impractical. I had previously experimented with a single column layout, but this had the problem that users had to do a lot of scrolling to get to the other engine results, as each engine would be listed one after the other. I came up with the solution of adding a tab to the side of the page for non-aggregated results that would allow fast jumping to the beginning of each engine's results.



I felt this was a good solution, and it received positive feedback, with one user stating “I liked the single panel with the left pane navigation, really simple and quick way to switch between results. I thought it was a clever bit of design.”

Clustering: Manipulating Arrays

Another issue that arose and must be mentioned was the difficulty with clustering. Clustering itself was not an issue, but working with multi-dimensional associative arrays that had been passed to a function by reference, caused complications. It was very late in the project lifespan, and I had difficulties working with foreach loops and manipulating the arrays. I was able to create an array with two elements. One element held the vector information for the full collection of documents. That is, the terms present in the full collection and their frequency of appearance. The second element was an array of sub-elements that each had a similar vector but each relating to only one specific document.

Unfortunately, when I tried to do any processing on the data in these arrays, I ran into difficulty, and operations seemed to go out of scope as soon as the loop ended. In the end I decided I did not have the time to properly figure out the problem and opted to focus on working on the written report, as the deadline was very close, and I felt it was in my best interest to ensure that was completed.

Page Titles

I found that some of the page titles were extremely long and unpleasant looking. I saw some titles that were as long as a paragraph of text. This made me choose to only display the websites URL as an identifier to the user.

Evaluation Details

In order to assess the performance of the MetaSearch engine, a list of 50 queries from the Text Retrieval Conference were provided. The queries were to be run through the system, for both aggregated and non-aggregated results, and then the results were to be compared against a list of

golden standard results that were also provided. The MetaSearch engine performance could then be gauged against each of the underlying engines to determine whether it improved upon their results.

The metrics that were to be calculated were as follows, where retrieved is the number of documents returned by an engine, relevant is the total number of relevant documents available, and relevant retrieved is the number of documents retrieved that are relevant:

Precision: The number of relevant documents returned by an engine divided by the total number of documents returned. This is a measure of how many of an engine's returned pages are relevant to the query.

Recall: The number of relevant documents returned by an engine divided by the total number of relevant documents. This is a measure of how many of the total number of relevant documents are actually returned by the engine.

Average Precision: This is the precision calculated at each point in the list where a relevant document is retrieved, and then averaged over all of the results for that query.

F-Measure: Two times the Recall multiplied by the Precision, divided by the Recall added to the Precision.

$$F\ measure = \frac{2\mathrm{RP}}{\mathrm{R} + \mathrm{P}}$$

This is a measure of the trade-off between recall and precision. An engine with both high precision and high recall will have a high F-Measure. If one is high and one is low, the F-Measure will be lower.

Precision @ 10: The precision measured for just the first 10 documents returned. This is actually an important metric, because most engines display 10 results per page, and most users never read beyond the first page. So those first 10 results that the engine returns are highly important and must be as accurate as possible.

Mean Average Precision (MAP): Simply the mean of the Average Precision for multiple queries. The AP for each query is summed, then divided by the number of queries.

Results

The following is a summary of the results obtained by running an evaluation script which acquired metric scores for all 50 queries for each of the search engines in one run. For a full list of results see Appendix A.

Evaluation Metric Scores

Engine	Precision	Recall	F-measure	P@10	MAP (%)
MetaSearch	0.6242	0.6242	0.6242	0.954348	51.4044
Google	0.37645	0.368	0.372154	0.908	30.12703
Bing	0.388016	0.3788	0.383308	0.894	29.5958
Blekko	0.377289	0.3602	0.36508	0.826	27.05204

Here we see that MetaSearch aggregation engine performs significantly better than all three of the underlying engines. This shows that the MAP score for MetaSearch is approximately 71% better than Google, 74% better than Bing, and 90% better than Blekko. It is important to note however this is a simulated test, not a real world test. So it is more accurate to say that MetaSearch performed better by those margins under the testing conditions we were required to evaluate under.

Statistical Analysis

In order to ensure that the results observed were statistically significant and not due to random chance, some statistical analysis was advised. A Student's T-Test was suggested.

A T-Test is a test which is used to test the Null Hypothesis, H_0. The Null Hypothesis states that there is no significant difference between two sets of data. If the p-value returned from a T-Test is below 0.05, we can say (at 95% confidence level) that the Null Hypothesis is rejected, and that there is a statistical significance to the results, i.e. they did not occur by chance.

In this analysis, the average precision achieved by each engine for each of the 50 queries was used for comparison. This means there are 50 average precision scores for the MetaSearch engine, which are compared against the 50 AP scores for one of the underlying engines. The test is performed 3 times to test the meta-search engine against each of the other engines.

There are a number of options when performing a T-Test. One can choose paired or unpaired. Here, paired was used, because each score in the results from each engine refers to a specific query. There is also the choice between one-tail or two-tail tests. Here, one-tail tests were performed.

Average Precision T-Test P-Values

MetaSearch vs Google	MetaSearch vs Bing	MetaSearch vs Blekko
6.78E-23	7.66E-23	2.17E-21

Here we see that in each test, the p-value is many orders of magnitude below 0.05, meaning that we can say with very high confidence (> 99% CL), that the results are statistically significant. This shows that the improved performance seen earlier by MetaSearch over the three underlying engines is absolutely not by chance. There is a significant difference in the results achieved by MetaSearch and statistically they are significant.

These outcomes show that MetaSearch aggregation engine was very successful in improving on the search results of its underlying engines. Not only were the Precision, Recall and MAP scores significantly higher, but also the average score for Precision @ 10 shows that MetaSearch was typically more successful at placing relevant documents at the top of the list, where users can easily find them. Since most users don't often read beyond the first 10 results, this is another aspect in which MetaSearch improves on the underlying engines. A T-Test was performed on the Precision@10 values also, to verify that they were also statistically significant. As can be seen below, we can say at 95% confidence level that the average P@10 for MetaSearch is better than the underlying engines.

Precision@10 T-Test P-Values

MetaSearch vs Google	MetaSearch vs Bing	MetaSearch vs Blekko
0.007752	0.000369	3.4E-9

We have seen that MetaSearch outperforms the underlying engines by a fair margin. But what about the age old debate of which of those underlying engines is better? I decided it would be of interest to compare the values of the three engines against each other. As seen in the Evaluation Metric Scores table, Google scores highest in Mean Average Precision with 30.127%, Bing ranks second with 29.596%, and Blekko is last with 27.052%. Looking at these scores one might assume that Google is the best engine of the three, but the results of a T-Test refute that observation.

Average Precision T-Test P-Values

Google vs Bing	Google vs Blekko	Bing vs Blekko
0.3896	0.071701	0.018821

We can see from these results that the Google-Bing results do not meet the requirements for rejecting the Null Hypothesis, and in fact are quite a ways off that target. From this we can say that the results placing Google in the lead ahead of Bing are potentially just a result of luck on the part of Google. The Bing-Blekko comparison shows the same result. The comparison between Google and Blekko comes closest to any statistical significance. However, it still does not meet the required p-value to reject the Null Hypothesis that there is no statistical difference between them. From this we can conclude that we cannot claim (at 95% confidence level) that there is any statistical difference between any of the three underlying engines. This is an interesting point, especially considering 93% of the users who took the user evaluation survey (15 users) stated that Google was their search engine of choice.

User Evaluation

A survey page was created on the site and connected to a MySQL database to store submission. This survey asked the user a number of questions assessing their experience of using the site. The following is a table of the survey results.

User Survey Results

Name	Email	Submitted	Engine	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Stephen Somers	stevosomers@gmail.com	2013-07-27	google	5	5	4	5	5	4	3
	Seems to be some slight issues in terms of bleed-over between columns on the aggregated search page. Other than that, it's very easy to use and I like the aggregated function.									
Debbie Roche	debks@eircom.net	2013-07-27	google	5	5	4	3	5	5	5

Name	Email	Submitted	Engine	Q2	Q3	Q4	Q5	Q6	Q7	Q8
	hello patrick :)									
Irene Foley	foleyirene@gmail.com	2013-07-27	google	5	5	5	4	5	5	5
	Excellent search engine									
Tom Monahan	mosmonahan@msn.com	2013-07-27	babylon	5	4	3	5	4	1	2
	At the moment it seems a bit slow, other than that it looks good.									
carl		2013-07-27	google	5	4	3	5	5	5	3
	Excellent work, speed was very impressive for this type of webapp, interface was intuitive and efficient.									
Alex Semenov	alex.semenov@ucdconnect.ie	2013-07-27	www.google.ie	4	3	3	5	5	1	1
wfrgrthdrhd		2013-07-27	google	5	5	4	5	2	5	2
James Coll	james.evin.coll@gmail.com	2013-07-27	google	5	5	5	4	4	5	4
	Nice work. Clean intuitive interface and easy to use.									
Dean		2013-07-27	google	5	5	4	4	3	2	3
	The speed is prohibitive, probably server limitations. I tried some searches that i had been using during the week however and the the results were very good. Saw some links i hadn't seen before. With further development i would certainly consider it									
alex wright	alex.wright87@gmail.com	2013-07-28	google	5	5	5	5	5	5	5

Name	Email	Submitted	Engine	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Daniel		2013-07-28	google	5	4	5	4	3	1	2
patrick		2013-07-28	google	5	5	4	4	4	3	1
Peter Tierney	musimind@gmail.com	2013-07-29	google	4	5	5	5	3	3	5
	Confused bout the Evaluation section.Could only work the password for Previous Evaluations,not New Evaluations,and wasn't sure does the password stay the same?Couldn't get the query option to offer more words!Well laid out and about section helpful!									
Patrick	patrickromeirl@hotmail.com	2013-07-29	google	5	5	5	4	5	4	5
	Has a very professional feel excellent use of html + css well done.									
James Loughran	james.loughran@gmail.com	2013-07-29	google	5	5	4	5	4	4	3

Summary of Results

Search Engines used by users: Google – 93% Babylon – 7%

Q2 - I found the interface very easy to use

Average score: 4.866667 This was mostly very positive

Q3 - I liked how the results were presented

Average Score: 4.666667 Also scored very positively

Q4 - In general, I found the quality of the results returned were superior to my normal search engine of choice.

Average Score: 4.2 This was a positive result, with most users feeling an improvement over their usual engine.

Q5 - I found that the quality of results returned were improved when aggregation was turned on in comparison to when it was turned off.

Average Score: 4.466667 Another very positive result. Most users agreed that aggregation improved results.

Q6 - I found that using query expansion helped provide useful alternative search options.

Average Score: 4.133333 This shows that users typically found the suggestions provided for alternative searches were useful.

Q7 - The speed of the MetaSearch engine is comparable to that of my typical engine of choice.

Average Score: 3.533333 Somewhat neutral, but verging on positive. This reflects the fact that the MetaSearch engine is significantly slower than Google.

Q8 - If given the option, I would consider making this search engine my default engine.

Average Score: 3.266667 Neutral. This reflects the average user's reluctance to switch search engines from their typical choice. This was to be expected, as Google is so common and so fast, and used by so many, more accurate search results are not likely to be enough to sway most users.

Conclusion and Future Work

All in all, the MetaSearch engine was, in my opinion, a good success. The results it produced were significantly better than the underlying engines in the synthetic tests performed, ranging from 71% to 90% better depending on the engine compared against.

Further, this was by no means random chance or luck, as the statistical analysis shows that the results can be said to be significant at 99%+ confidence level.

User experience evaluation demonstrated an overall satisfaction amongst users of the site. The interface pleased the majority of people, and the results seemed to satisfy most users also.

Despite the overall satisfaction with MetaSearch, there are some criticisms. Many users cited the speed as an issue. While the speed, measuring in at around 3-3.5 seconds, is not so bad for a search engine of this nature, and given the limited resources, compared against sites like google, the difference is easily apparent. I am aware of an improvement that I would certainly like to include for future updates, and that is the use of curl_multi_exec. Most of the delay in load times comes from the delay inherent in sending many requests to external APIs and waiting for their response. Curl_multi_exec performs like cURL, but it allows for multiple requests to be sent simultaneously. This should perceivably improve performance, possibly by about 1-1.5 seconds.

I was disappointed by my failure to include a Clustering display option. This was primarily due to time constraints, but difficulties experienced worsened the matter. I feel that given 2-3 days more, I would have been able to implement Clustering, and it is my intention to do so as soon as possible.

The project in general was a great learning experience. It thought me a lot about designing websites like this, and it introduced me to the powerful ability of using external APIs to process requests. I feel my potential for creating more advance websites based on php scripting has improved a great deal, and I look forward to finding challenges which will make use of this experience.

References

Aslam, J. A., & Montague, M. (n.d.). Condorcet Fusion for Improved Retrieval.

Aslam, J. A., & Montague, M. (n.d.). Methods For Metasearch.

He, D., & Wu, D. (n.d.). Toward a Robust Data Fusion for Document Retrieval. Retrieved from http://www.sis.pitt.edu/~daqing/docs/datafusion-nlpke-117.pdf

Lillis, D. (2012). Software Engineering Project: Metasearch and Fusion.

Skorkin, A. (2010, March 1). *How Search Engines Process Documents Before Indexing*. Retrieved June 20, 2013, from http://www.skorks.com/2010/03/how-search-engines-process-documents-before-indexing/

Appendix A – Evaluation Results

MetaSearch Engine Statistics

MAP	51.40441	**Average F-Measure**	0.6242

TREC Index	Query	Precision	Recall	F-measure	P@10	Avg Precision
151	403b	0.67	0.67	0.67	1	58.7791
152	angular cheilitis	0.75	0.75	0.75	0.9	64.26368
153	pocono	0.56	0.56	0.56	1	45.54726
154	figs	0.71	0.71	0.71	1	61.29688
155	last supper painting	0.67	0.67	0.67	0.9	56.46337
156	university of phoenix	0.51	0.51	0.51	0.9	39.19456
157	the beatles rock band	0.59	0.59	0.59	0.9	45.10717

158	septic system design	0.76	0.76	0.76	1	69.82807
159	porterville	0.56	0.56	0.56	1	46.69593
160	grilling	0.67	0.67	0.67	1	56.16503
161	furniture for small spaces	0.49	0.49	0.49	0.9	33.29706
162	dnr	0.48	0.48	0.48	1	39.38516
163	arkansas	0.63	0.63	0.63	1	53.19602
164	hobby stores	0.64	0.64	0.64	0.9	46.02436
165	blue throated hummingbird	0.44	0.44	0.44	0.8	26.5655
166	computer programming	0.57	0.57	0.57	0.8	42.6151
167	barbados	0.61	0.61	0.61	1	45.81444
168	lipoma	0.67	0.67	0.67	1	52.52821
169	battles in the civil war	0.54	0.54	0.54	1	40.69806
170	scooters	0.57	0.57	0.57	0.8	38.97685
171	ron howard	0.68	0.68	0.68	1	62.4489
172	becoming a paralegal	0.68	0.68	0.68	1	60.69741
173	hip fractures	0.68	0.68	0.68	1	57.7022
174	rock art	0.69	0.69	0.69	1	60.4054
175	signs of a heartattack	0.55	0.55	0.55	1	45.8765
176	weather strip	0.63	0.63	0.63	1	50.76219

177	best long term care insurance	0.67	0.67	0.67	0.9	56.65174
178	pork tenderloin	0.64	0.64	0.64	1	52.15285
179	black history	0.65	0.65	0.65	1	56.77827
180	newyork hotels	0.67	0.67	0.67	1	57.40174
181	old coins	0.63	0.63	0.63	1	54.52334
182	quit smoking	0.68	0.68	0.68	1	62.71472
183	kansas city mo	0.71	0.71	0.71	1	58.80265
184	civil right movement	0.66	0.66	0.66	1	54.09285
185	credit report	0.6	0.6	0.6	1	52.98182
186	unc	0.55	0.55	0.55	1	46.95878
187	vanuatu	0.68	0.68	0.68	1	59.87241
188	internet phone service	0.73	0.73	0.73	1	63.47751
189	gs pay rate	0.51	0.51	0.51	0.8	32.58214
190	brooks brothers clearance	0.48	0.48	0.48	0.8	34.45404
191	churchill downs	0.67	0.67	0.67	1	53.99196
192	condos in florida	0.64	0.64	0.64	1	53.72469
193	dog clean up bags	0.58	0.58	0.58	0.6	39.7596
194	designer dog breeds	0.66	0.66	0.66	1	57.27636

195	pressure washers	0.54	0.54	0.54	1	43.67812
196	sore throat	0.65	0.65	0.65	1	57.23092
197	idaho state flower	0.63	0.63	0.63	1	54.73265
198	indiana state fairgrounds	0.65	0.65	0.65	0.9	55.7307
199	fybromyalgia	0.64	0.64	0.64	0.9	49.48273
200	ontario california airport	0.69	0.69	0.69	1	60.8336

Google Statistics

MAP 30.12703 **Average F-Measure** 0.372154061

TREC Index	Query	Precision	Recall	F-measure	P@10	Avg Precision
151	403b	0.397959	0.39	0.393939	0.9	33.09014
152	angular cheilitis	0.347368	0.33	0.338462	0.9	27.5561
153	pocono	0.309278	0.3	0.304569	0.8	24.84213
154	figs	0.319588	0.31	0.314721	0.9	25.80417
155	last supper painting	0.42268	0.41	0.416244	1	34.96864
156	university of phoenix	0.37234	0.35	0.360825	1	30.13409
157	the beatles rock band	0.383838	0.38	0.38191	1	32.99083
158	septic system design	0.4375	0.42	0.428571	1	34.15302
159	porterville	0.381443	0.37	0.375635	1	30.73263

160	grilling	0.452632	0.43	0.441026	1	33.92913
161	furniture for small spaces	0.173913	0.16	0.166667	0.7	8.064479
162	dnr	0.22	0.22	0.22	1	19.03582
163	arkansas	0.313131	0.31	0.311558	1	24.42397
164	hobby stores	0.166667	0.16	0.163265	0.8	11.35824
165	blue throated hummingbird	0.191919	0.19	0.190955	0.4	8.590516
166	computer programming	0.212121	0.21	0.211055	0.6	10.31057
167	barbados	0.232323	0.23	0.231156	0.8	14.36095
168	lipoma	0.22449	0.22	0.222222	0.7	13.7553
169	battles in the civil war	0.16	0.16	0.16	0.7	11.37631
170	scooters	0.208333	0.2	0.204082	0.5	8.423496
171	ron howard	0.49	0.49	0.49	1	43.33751
172	becoming a paralegal	0.393939	0.39	0.39196	0.9	31.64137
173	hip fractures	0.373737	0.37	0.371859	1	30.44869
174	rock art	0.414141	0.41	0.41206	1	37.10649
175	signs of a heartattack	0.391753	0.38	0.385787	0.8	28.79258
176	weather strip	0.367347	0.36	0.363636	0.9	30.74665
177	best long term care insurance	0.427083	0.41	0.418367	0.9	32.25318
178	pork tenderloin	0.333333	0.32	0.326531	1	27.94259
179	black history	0.444444	0.44	0.442211	1	35.82516

180	newyork hotels	0.618557	0.6	0.609137	1	56.30722
181	old coins	0.373737	0.37	0.371859	1	31.26368
182	quit smoking	0.530612	0.52	0.525253	1	46.51534
183	kansas city mo	0.44898	0.44	0.444444	1	40.3308
184	civil right movement	0.44898	0.44	0.444444	1	37.99081
185	credit report	0.443299	0.43	0.436548	1	35.03079
186	unc	0.43	0.43	0.43	1	38.47917
187	vanuatu	0.546392	0.53	0.538071	1	45.11918
188	internet phone service	0.49	0.49	0.49	0.9	40.47756
189	gs pay rate	0.381443	0.37	0.375635	1	30.45501
190	brooks brothers clearance	0.408163	0.4	0.40404	0.9	29.90432
191	churchill downs	0.459184	0.45	0.454545	1	41.05484
192	condos in florida	0.397959	0.39	0.393939	1	32.86097
193	dog clean up bags	0.391753	0.38	0.385787	0.9	31.85716
194	designer dog breeds	0.418367	0.41	0.414141	1	34.05508
195	pressure washers	0.38	0.38	0.38	0.9	28.65773
196	sore throat	0.434343	0.43	0.432161	1	35.38164
197	idaho state flower	0.391753	0.38	0.385787	1	33.58989
198	indiana state fairgrounds	0.371134	0.36	0.365482	1	30.12313

199	fybromyalgia	0.41	0.41	0.41	0.7	29.9587
200	ontario california airport	0.484536	0.47	0.477157	0.9	40.94353

Bing Statistics

MAP	29.5958	**Average F-Measure**	0.383307679

TREC Index	Query	Precision	Recall	F-measure	P@10	Avg Precision
151	403b	0.34	0.34	0.34	1	27.89006
152	angular cheilitis	0.40404	0.4	0.40201	1	33.01251
153	pocono	0.295918	0.29	0.292929	0.9	22.07599
154	figs	0.453608	0.44	0.446701	0.9	36.89616
155	last supper painting	0.371134	0.36	0.365482	0.9	27.72383
156	university of phoenix	0.365591	0.34	0.352332	0.5	20.75535

157	the beatles rock band	0.350515	0.34	0.345178	0.8	25.71393
158	septic system design	0.42	0.42	0.42	1	33.52036
159	porterville	0.364583	0.35	0.357143	1	27.25332
160	grilling	0.397959	0.39	0.393939	1	36.27454
161	furniture for small spaces	0.402062	0.39	0.395939	1	32.15514
162	dnr	0.308511	0.29	0.298969	1	25.43487
163	arkansas	0.505155	0.49	0.497462	1	37.06664
164	hobby stores	0.4	0.4	0.4	1	33.64376
165	blue throated hummingbird	0.377551	0.37	0.373737	0.9	22.61722
166	computer programming	0.540816	0.53	0.535354	1	47.75459

167	barbados	0.414141	0.41	0.41206	0.9	34.99833
168	lipoma	0.536082	0.52	0.527919	1	45.69
169	battles in the civil war	0.333333	0.33	0.331658	0.7	25.33681
170	scooters	0.438776	0.43	0.434343	0.9	32.65567
171	ron howard	0.459184	0.45	0.454545	1	35.77334
172	becoming a paralegal	0.37	0.37	0.37	1	29.57242
173	hip fractures	0.459184	0.45	0.454545	1	37.77849
174	rock art	0.4	0.4	0.4	0.8	28.18827
175	signs of a heartattack	0.34	0.34	0.34	1	26.94183
176	weather strip	0.357143	0.35	0.353535	0.8	26.96194
177	best long term care insurance	0.37	0.37	0.37	0.9	26.21891

178	pork tenderloin	0.353535	0.35	0.351759	1	29.494
179	black history	0.412371	0.4	0.406091	1	34.14232
180	newyork hotels	0.40404	0.4	0.40201	1	28.41223
181	old coins	0.319588	0.31	0.314721	0.9	26.77747
182	quit smoking	0.41	0.41	0.41	1	36.39919
183	kansas city mo	0.468085	0.44	0.453608	1	35.64909
184	civil right movement	0.360825	0.35	0.35533	0.7	24.25251
185	credit report	0.393939	0.39	0.39196	1	31.49482
186	unc	0.4	0.38	0.389744	0.9	28.46336
187	vanuatu	0.40404	0.4	0.40201	0.9	31.006
188	internet phone service	0.48	0.48	0.48	1	39.02688
189	gs pay rate	0.185567	0.18	0.182741	0.5	7.260962

190	brooks brothers clearance	0.239583	0.23	0.234694	0.7	14.92375
191	churchill downs	0.510417	0.49	0.5	0.8	36.55456
192	condos in florida	0.35	0.35	0.35	0.9	27.79015
193	dog clean up bags	0.340206	0.33	0.335025	0.4	14.99747
194	designer dog breeds	0.31	0.31	0.31	0.8	23.0586
195	pressure washers	0.408163	0.4	0.40404	0.9	30.32441
196	sore throat	0.408163	0.4	0.40404	1	33.39426
197	idaho state flower	0.461538	0.42	0.439791	0.9	32.66795
198	indiana state fairgrounds	0.368421	0.35	0.358974	0.8	24.51386

199	fybromyalgia	0.292929	0.29	0.291457	0.8	23.87078
200	ontario california airport	0.344086	0.32	0.331606	0.9	25.41127

Blekko Statistics

MAP 27.05204 **Average F-Measure** 0.365079673

TREC Index	Query	Precision	Recall	F-measure	P@10	Avg Precision
151	403b	0.39	0.39	0.39	0.9	32.2852
152	angular cheilitis	0.628571	0.44	0.517647	0.8	36.58763
153	pocono	0.34	0.34	0.34	1	24.86846
154	figs	0.28	0.28	0.28	1	22.76085
155	last supper painting	0.41	0.41	0.41	0.9	29.69743
156	university of phoenix	0.27	0.27	0.27	0.8	15.50953
157	the beatles rock band	0.27	0.27	0.27	0.9	17.88052
158	septic system design	0.38	0.38	0.38	0.9	33.5412
159	porterville	0.31	0.31	0.31	0.9	23.33339
160	grilling	0.25	0.25	0.25	0.6	16.44305
161	furniture for small spaces	0.36	0.36	0.36	0.6	23.59774

162	dnr	0.48	0.48	0.48	0.9	36.94792
163	arkansas	0.505051	0.5	0.502513	1	45.4861
164	hobby stores	0.43	0.43	0.43	0.8	33.69395
165	blue throated hummingbird	0.474747	0.47	0.472362	0.9	34.56016
166	computer programming	0.38	0.38	0.38	0.8	25.45614
167	barbados	0.454545	0.45	0.452261	0.8	34.32245
168	lipoma	0.454545	0.45	0.452261	1	37.72298
169	battles in the civil war	0.44	0.44	0.44	0.9	38.12735
170	scooters	0.434343	0.43	0.432161	0.7	30.09444
171	ron howard	0.41	0.41	0.41	0.9	31.14063
172	becoming a paralegal	0.434343	0.43	0.432161	0.9	38.19929
173	hip fractures	0.39	0.39	0.39	1	30.48566
174	rock art	0.38	0.38	0.38	0.9	28.67588
175	signs of a heartattack	0.821429	0.23	0.359375	0.9	21.218
176	weather strip	0.33	0.33	0.33	0.8	18.54603
177	best long term care insurance	0.31	0.31	0.31	0.8	23.78016
178	pork tenderloin	0.31	0.31	0.31	0.8	22.35373

179	black history	0.39	0.39	0.39	0.9	33.2055
180	newyork hotels	0.28	0.28	0.28	0.8	23.13832
181	old coins	0.397959	0.39	0.393939	0.9	30.0966
182	quit smoking	0.408163	0.4	0.40404	1	36.5245
183	kansas city mo	0.26	0.26	0.26	0.6	16.58918
184	civil right movement	0.38	0.38	0.38	0.9	31.52853
185	credit report	0.323232	0.32	0.321608	0.9	24.47843
186	unc	0.212121	0.21	0.211055	0.5	12.45494
187	vanuatu	0.383838	0.38	0.38191	0.9	28.22807
188	internet phone service	0.37	0.37	0.37	0.9	29.89649
189	gs pay rate	0.31	0.31	0.31	0.5	16.89605
190	brooks brothers clearance	0.25	0.25	0.25	0.8	17.76412
191	churchill downs	0.31	0.31	0.31	0.8	22.1794
192	condos in florida	0.285714	0.28	0.282828	0.7	17.90733
193	dog clean up bags	0.33	0.33	0.33	0.6	18.81491
194	designer dog breeds	0.252525	0.25	0.251256	0.7	17.38328
195	pressure washers	0.32	0.32	0.32	0.7	18.65556
196	sore throat	0.34	0.34	0.34	1	28.82588

197	idaho state flower	0.459184	0.45	0.454545	0.8	36.63402
198	indiana state fairgrounds	0.43	0.43	0.43	0.9	29.81577
199	fybromyalgia	0.414141	0.41	0.41206	0.8	32.55339
200	ontario california airport	0.43	0.43	0.43	0.6	21.71593