University of Glasgow | School of Computing Science

Exploratory Search Engine

Alex Chilikov

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 20, 2017

Abstract

With the increased use of search engines in the recent years, IR researchers have been examining different ways to optimize the effectiveness of these software products. The current paper concentrates on a specialization of the information retrieval field known as Exploratory Search presenting the “Exploratory Search Engine“ application.

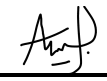
The main objective of the product is to provide alternative mechanisms for search and implement search algorithms beneficial for exploratory searchers in an attempt to satisfy the needs of these search engine users. The application addresses the main challenges exploratory searchers face which are related to their “unfamiliarity with the domain of their goal, uncertainty about the ways to achieve their goals, and/or uncertainty about their goals in the first place“ [80].

Acknowledgements

I would like to thank Dr Craig Macdonald and Graham Mcdonald for their continued support and guidance throughout the year. Additionally, I would like to thank all the participants in the evaluation phase for their valuable input and recommendations.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: Alex Chilikov Signature: 

Contents

Chapter 1

Introduction

The purpose of this chapter is to familiarize the reader with the overall aims of the project and its corresponding achievements. It illustrates the motivation for the project and presents some related products at the end.

1.1 Aims

Web searching is probably the most frequent user task in the Web, during which users usually get back a ranked list of hits. Unfortunately, such a list of ranked documents is quite often not enough. Sometimes users are unfamiliar with the domain of their goal. In this case, they struggle to formulate an accurate query to represent their information need. As a result of such an inaccurate query, the search results are most of the time not particularly helpful for the users to achieve their task [80]. In other words, users need to first learn about the topic to understand how to achieve their goal. However, this is a challenge if they cannot formulate a representative query to get search results relevant to the topic they need to learn more about. Other times, users are unsure about the ways to achieve their goals because they do not understand either the technology or the process or do not even have a clear goal. Whenever searchers experience any of these challenges they try to perform different activities in an attempt to overcome the challenges [80]. Such activities are collectively known as Exploratory search [80]. The aim of the current project is to adapt the xQuAD framework to extend the University's own Terrier.org information retrieval platform in order to identify related subtopics for a user's query. Hopefully, this will allow an ambiguous or inaccurate query entered by a user unfamiliar with the domain of their goal, to produce more diverse range of search results including different subtopics of the initial information need. In this way the user will be able to acquire more knowledge about the topic and be able to refine its query once he or she has understood the topic better. The application should additionally automatically categorize the search results in attempt to support users with unclear goals or once again to assist them to better understand the searched domain. Finally, the Exploratory Search Engine should allow the users to explore the automatically generated subtopics without expecting from them to formulate a new query (this is challenge in this scenario as discussed above). This can be achieved by simply asking the user to select the category and performing some retrieval algorithm behind the scenes to retrieve back the category specific results.

An optional objective of the project is to have a relevant documents list where users can store documents potentially relevant to their goal. The purpose of this list is to reduce the short-term memory load. When users are searching for information in a field they are not familiar with or they do not have a clear goal, it is much harder to decide which documents are relevant and which are irrelevant. Hence, the relevant documents list is supposed to assist the user to overcome this challenge by allowing him/her to store potentially relevant documents, so he can make a more informed decision later when he/she has better understood the searched domain. An additional advantage of a relevant documents list is that it supports the learning process. As mentioned above users should

first understand the searched topic. Storing crucial documents containing general overview of the searched topic in a relevant documents list can be beneficial for future reference.

1.2 Motivations

Guiding users towards the information they require is becoming more and more crucial in the digital era we are all living in today. Unfortunately, an issue identified by Stephann Makri is that many search models analyse information at a high level of abstraction and are rarely successfully guiding their users towards their information need [26]. This is an issue which can be directly addressed by the Exploratory Search Engine‘s “exploratory search“ functionality. Given an abstract high level query the application allows an “exploratory search“ of a particular sub-topic of this user query, which is further split into sub-topics providing more concrete results.

An example of users frequently experiencing the challenges mentioned by Stephann Makri are journalist. Millions of journalists spend a large amount of their working time in searching activities. “On average, the journalists spent 43.0 per cent of their working time or 3:55 hours per day on research“ considering their written survey of 601 journalists and the participation of 48 journalist in an experiment [25]. The article states that about 80 to 92% of journalists use internet search engines as part of their work, as 97% of them choose Google as their most utilized search engine [25]. Unfortunately, this leads to concerns with the accuracy of the information and a potential distortion of reality [25]. The accuracy is negatively affected by the high level of abstraction not allowing these searchers to fully understand the topic [26]. On the other hand, the distortion of reality addresses the fact that they mainly rely on Google‘s ranking algorithms to acquire information for their articles.

Taking the large amount of journalists using search engines and their influence on articles written by these information workers into account a diversification framework diversifying the result set and presenting more than one potential aspect of an ambiguous query can lead to more accurate and objective articles presenting both sides of a given topic. In addition, journalist are not always specialists in the domain they should write an article about. Hence, they experience most of the challenges mention in the previous section. The Exploratory Search Engine can assist them to overcome these challenges and achieve their goal via its abilities to generate subtopics, allow the exploration of the automatically generated subtopics and its relevant results list.

Even though there are a lot of different popular search engines used extensively by different types of information workers as mentioned above, the mainstream search engines serve a different purpose. They mainly benefit customers who are already experts or are at least familiar with the searched field for the following reasons. First, they lack a mechanism for an “exploratory search“. There is no way to further explore your search results without reformulating your query, which is problematic as discussed in the first section. It is possible to select “suggested queries“ but their relevance is again directly related to the accuracy of the initial query (which is by definition not accurate for Exploratory Search). There is no functionality for further filtering of the result set such as faceted navigation, for example, making it hard for users not aware of the searched field to identify what they are looking for. These systems are designed for clients with a more broad expectations who can be easily satisfied by the system‘s search algorithms filtering all popular documents using a technique known as *a link popularity score* [24] or filtering all articles relevant to a particular region or popular among its customers based on the customers‘ log. Unfortunately, such an approach fails to satisfy a key customer group - the group of the information workers who often do not know exactly what they are looking for and require very specific results based on the content of the documents not their popularity. The Exploratory Search Engine project is particularly appropriate in such a scenario, as it does not concentrate on the popularity of the documents or the clients log to rank them. It uses their absolute relevance to the given query to filter the most accurate results via standard weighting models such as TF_IDF or BM25 [27]. It allows a search for concrete data not trends through its “exploratory search“ mechanism which assists the users to find exactly what they need in the unfamiliar field they are searching in.

Additionally, when searcher are performing an exploratory search they frequently keep track of the relevant documents or extracts of documents they have found so far. This process assists them to organize their thoughts and gather relevant information. Unfortunately, most of the popular search engines do not have a feature to capture such needs. For this reason, users have to rely on external applications to organize their findings discovered during the search session. The exploratory search engine supports this process via a "A list of relevant documents" functionality which allows users to gather the most useful documents they have found.

1.3 Summary

The main aims and motivations for the development of the Exploratory Search Engine are presented in Chapter 1. What an exploratory search is, the main challenges it addresses, and how they are mapped to the functionalities of the Exploratory Search Engine application are also discussed. A number of motivations for the project are presented in the second part of Chapter 1, as the most significant of which are related to the high level of abstraction in the currently used search models, potential distortion of reality, the different purpose of the mainstream search engines and the benefits of the list of relevant documents for exploratory search. Chapter 2 summarizes the relevant research presenting background information and terminology, introduces the reader to several related products, and provides an overview of xQuAD. Chapter 3 discusses the functional and non-functional requirements. Chapter 4 describes the architecture of the system, as it first discusses the overall architecture and workflow and then goes into more details about the logic tier and front-end architectures and technologies. Chapter 5 concentrates on the more challenging implementation decisions and presents the GUI. Chapter 6 analyses the offline and online evaluation and the users' log acquired as a result of the latter. Finally, Chapter 7 summarizes the achievements of the project and makes suggestions for future improvements.

Chapter 2

Relevant Research

This chapter is summarization of several books, research papers, related products and information retrieval algorithms I had to get familiar with, implement or integrate as part of the Exploration Search Engine project. Section 2.1 reveals 3 categories of related products, discussing how they handle or struggle with different exploratory search challenges. Section 2.2. provides background information and terminology used within the paper. Finally, section 2.3 introduces the xQuAD library.

2.1 Related Products

2.1.1 Mainstream Search Engines

Mainstream Search Engines such as Bing, Google and Yahoo are widely used today, as their main purpose is to satisfy the needs of the general public. Google is used as a representative of this group of search engines in the current paper. Even though it is not considered a classical example of a search engine used for exploratory search, Google is a primary source of information for many searchers as the evidence presented in the Motivations section suggests. For this reason, it would be taken into account in this section of report.

Google is popular as one of the most used search engines. Unfortunately, even though it is so popular, it still does not diversify the result set as expected for some topics. If we consider a scenario where a geography teacher must give a lesson about fluvial geography, which is the study of the processes associated with rivers and streams and she should first talk about different types of banks and how we can identify useful information from their structure, she will first most likely use the help of Google and try to get more information about different types of banks. Such a query can be particularly challenging for a search engine due to the multiple meanings of the word bank. For this reason a diversification mechanism, such as the one provided from xQuAD is vital all potential customer groups to be satisfied. Unfortunately, surprisingly or not all results appearing on the first 5 pages of the result set refer to the banking industry, as there is no single result related to river banks. At this moment the teacher will most likely realize that he or she should specify her/his query further and search for “river bank“, for example . However, once again all results are about a bank called River Bank or Brad Paisleys song “River Bank“. The same experiment is performed using Yahoo and Bing with similar success. Considering that teachers should prepare for each lesson in advance, such a search process experience can get frustrating and highly inefficient as well as even ineffective. The Exploratory Search Engine tackles the presented issues by performing a secondary search during which documents relevant to subtopics identified from the initial search results are presented to the user. This allows a large range of potential intents to be covered and an ambiguous query (expected from an exploratory search task) to be successfully processed giving more appropriate results.

2.1.2 Search Engines Based on Category Overviews

There are a great number of web services attempting to handle exploratory search tasks providing a fixed hierarchical structure of the searched domain. In this context fixed refers to a hard-coded structure of subtopics which are not dynamically generated based on the search request. For example, similarly to the organization of books in libraries into categories there exist many large online text collections with assigned category labels. An example of such an organization is the Medical Subjection Heading (MeSH) including approximately 18,00 categories [2]. The MeSHBrowse interface, an application used for search within the collection, presents a graph visualizing the organization of different categories into subcategories. Figure 2.1 below shows a screenshot of the MeSH application:

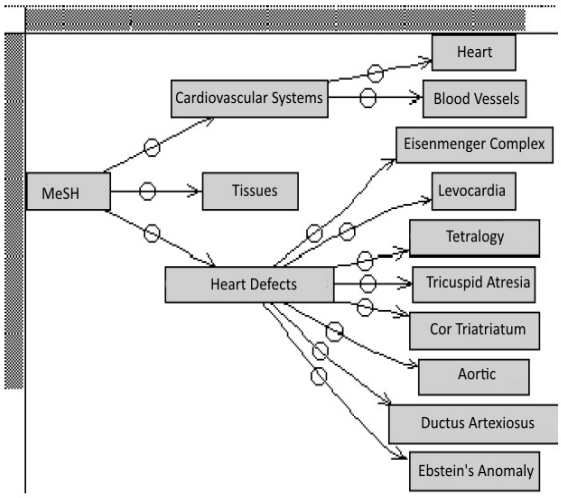


Figure 2.1: The MeSHBrowse interface for viewing category labels hierarchically.

The MeSHBrowse application can be found particular useful for exploratory search as it does not require the searcher to be an expert in the searched field to be able to formulate an accurate query. Actually, the system does expect a query to be entered at all it does not have such a functionality. It relies on the exploration of the presented categories and subcategories until the desired documents are found. Even though at first look this looks like a perfect design, it actually has a great number of drawbacks overcome by the Exploratory Search Engine Project. First, the category labels can be subjective and confusing. It is a frequent scenario to know exactly what you are searching for and to see more than 1 category on the MeSH graph which looks appropriate [2]. If the wrong path of exploration is selected, the user will either have to explore it completely or to reach a depth level where he or she will realize that this path is incorrect. Additionally, searching for information within such a graph can be a time-consuming task, especially if the desired category is somewhere deep in the hierarchical structure of the graph. The MeSHBrowse interface is just a “category graph“. Hence, once the desired category is found, alternative means are necessary to search within the category. Even a more inefficient scenario is the case when the searched information is not part of the collection. This means that the user will have to explore a large part of the graph just to realize that the information he or she is looking for is not there.

On the other hand the Exploratory Search Engine resolves most of the difficulties presented above by using a combination of automatic categorization of the result set , which represents a feature similar to the “category graph“ of the MeSHBrowse, and standard search relying on queries. In this way the user cannot choose the wrong path in the graph, as he or she first uses the search functionality which will present categories only appropriate to the user‘s search. This approach is more efficient as well in case the query is very specific because in this scenario the user of the MeSHBrowse interface will need to go deep in the hierarchy which is time-consuming. A user of the Exploratory Search Engine, on the other hand, will be directly presented the appropriate categories.

An alternative example of a web service based on category overviews is Amazon. More precisely it repre-

sents a faceted search interface where users are presented a list of alternative subcategories related to their query. Most of these types of web services rely on classifiers to produce relevant subcategories which are an example of a supervised learning [82]. Hence, they require training data to perform well, for which reason they cannot be generalized to general-purpose search engines (as this will require enormous amount of training data). Additionally, faceted search interfaces rely on an initially generated hierarchical category structure to which search results are mapped, which is a limitation, as the accuracy of the suggested subcategories for a search request directly depends on the accuracy and size of the category structure. These two drawbacks represent the main disadvantages of the faceted search interfaces when compared to Exploratory Search Engine. The current system performs categorization based on unsupervised algorithms allowing it to handle as general search queries as any other general-purpose search engine without depending on an initially generated category structure.

2.1.3 Specialized Search Engines

The final section of related products discusses two specialized search engines - CE Search Engine and Skyscanner [34]. CE Search Engine is healthcare continuing education search engine. It lists thousands of providers of CE (Continuing Education) by profession [9]. This search engine, a screenshot of which can be seen on Figure 2.2, represents an alternative way of search. The engine clusters the results and provides advanced filtering not seen in the previous two search engine categories. In this regard, it is similar to the Exploratory Search Engine project with its clustering functionality. Unfortunately, it is limited to only healthcare continuing education search. Even though such a solution can be particularly useful for a specific subset of information workers always interested in the same field such as lawyer, research scientist, and medical scientist, it is not appropriate for employees operating with wide variety of topics such as journalist for example. On the other hand, the Exploratory Search Engine is not limited to a particular search domain. It allows the user to search for any type of information he/she needs. Skyscanner, which is an example of a popular flight search engine, is similarly overspecialized

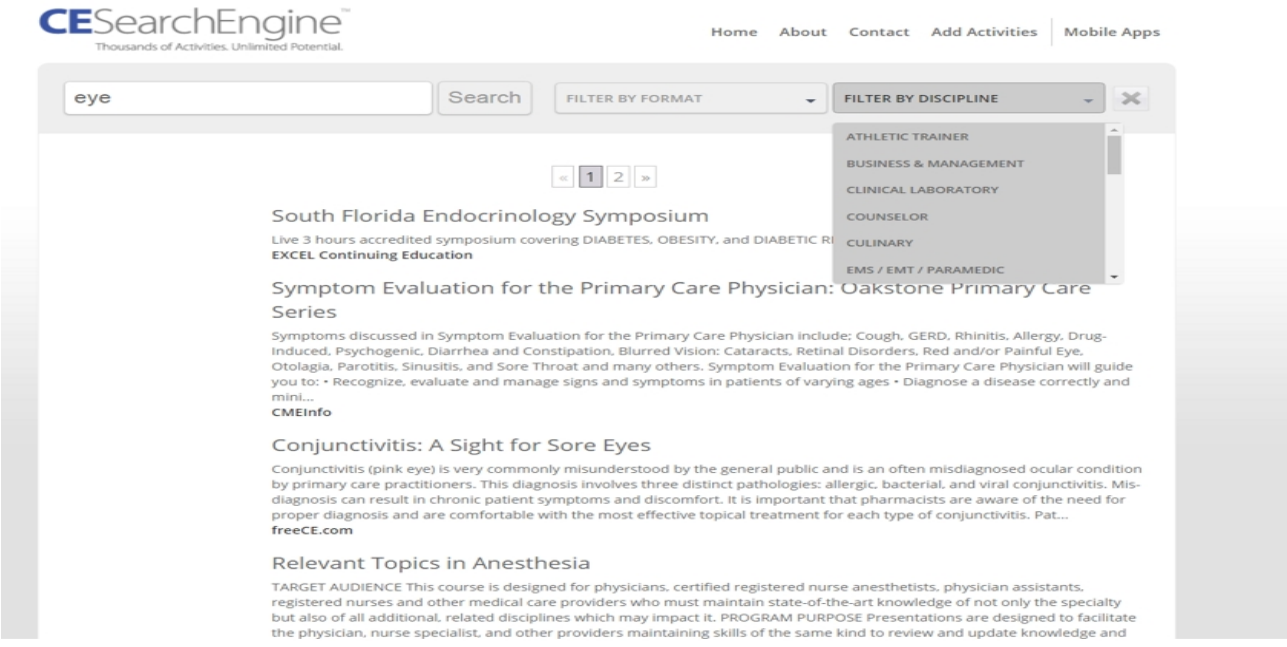


Figure 2.2: Results from CESearchEngine for the query “eye“ with a list of disciplines by which the query can be filtered.

but, on the other hand, it is capable of assisting users to find information even when they have an unclear goal and/or are unsure what they are looking for (examples of exploratory search challenges). It achieves this by presenting attractive offers to its users and/or informative articles from which they can learn more about different destinations. Such a functionality is useful for an exploratory search where users are unsure about their goals in the first place. Unfortunately, this is not provided from the Exploratory Search engine and it is quite problematic

to be implemented for any general-purpose search engine as the information need of the users is not known in advance which is the case for the flight search engine.

2.1.4 Summary

The related product section presented three types of search engines as for each type the advantages and disadvantages of the presented systems with respect to the Exploratory Search engine were analyzed. First, several mainstream systems were discussed and it was identified that they retrieved too specific results for some queries resulting in low recall and an inability of the users to accomplish their goal. On the other hand, the Exploratory Search engine overcomes this challenge by diversifying the result set presenting relevant results to different subtopics of the search query. Several search engines based on category overviews were considered as a second category of search engines. One of them (MeSHBrowse) relied on category graphs which was identified to be an inefficient way to visualize data in some scenarios as if the wrong exploration path within the graph was selected the user would waste to much time to explore its entire depth just to realize that what he was looking for is not there. Also, it was discussed that some category labels can be confusing and/or subjective. For faceted search interfaces it was discovered that their effectiveness was directly related to the accuracy and size of the hard-coded category structure. All of these challenges are tackled by the Exploratory Search engine which generates subcategories automatically for a given search request - hence the users cannot choose the "wrong path", as they are directly presented the relevant subtopics (instead of having to search for them). Also, these subcategories do not depend on a hard-coded category structure as the subtopics are generated automatically based on an unsupervised learning algorithms, consequently there is no association between the accuracy and size of the category structure and the effectiveness of the search engine because there is no such a structure at all. Finally, two specialized search engines were presented which cope with two of the most frequent exploratory search challenges, namely the unfamiliar search domain and unclear search goal. Similarly the Exploratory Search engine project is also capable to assist the users to search in an unfamiliar domain for all different types of search queries through its categorization feature while CE Search Engine is only able to achieve this for searches related to continuing education. Furthermore, the current product also guides searchers with unclear search goals via the xQuAD library which diversifies the result set presenting alternative subtopics of an ambiguous query.

2.2 Background Information and Terminology

The purpose of this section is to introduce the reader to some general information retrieval (IR) knowledge and terms used in the paper. It allows a common terminology to be established and the sources of this information to be identified.

Perhaps, the most key IR term is the index [27]. The indexing process allows the collection of documents searched for a given information need to be stored in a data structure known as an Index mapping words to the documents they occur in. The index allows the concrete documents a word occurs in to be detected in linear time. The collection of documents over which the retrieval is performed is known as a corpus.

The two main types of information retrieval systems discussed in this paper are the *ad hoc retrieval system* and the *exploratory search system*. An *ad hoc retrieval system*, which is usually what most people think of when they are asked what a search engines aim is, "provides documents from within the collection that are relevant to an arbitrary user information need, communicated to the system by means of a one-off, user-initiated query." [27]. This is the Exploratory Search applications first responsibility. On the other hand, *an exploratory search* system supports users to find the relevant information in a situation where they lack the knowledge or contextual awareness to construct the most accurate queries [79]. When the searcher uses the system to enter his or her query we can actually look at the search phrase from two different angles. On one hand, we have the *query*

itself which is "what the user conveys to the computer in an attempt to communicate the information need" [27]. The *query* is a tool, an instrument, the user uses to retrieve what he or she needs, while, on the other hand, *the information need* "is the topic about which the user desires to know more" [9]. This two terms will be frequently used especially in the Implementation chapter where it will be discussed how the *query* will be automatically modified to increase the probability of the result set to satisfy *the information need*. The next section introduces the library supporting this automatic modification of the searched query.

2.3 Explicit Search Result Diversification

Considering a query of a typical user persona it can be expected that the query is quite vague and unclear. The reason for this is that the system is supposed to be used for an *exploratory search*, hence ambiguous queries are expected. To support the user to retrieve the information, she or he is looking for, the result set must be diversified to capture multiple aspects of the search query in case the query is ambiguous or to simply provide a more diverse result set to allow further exploration in a particular direction. This functionality is provided through the xQuAD library.

The xQuAD library diversifies the result set by the use of sub-queries. The library also considers the notion of redundancy and the order in which documents are presented to the user. More specifically it regards a document, which must be relevant to query in a normal scenario, less relevant when this document is read after several similar documents have been read. The library also regards the importance of the sub-queries and how well each document satisfies each sub-query, not only the relevance of a document in the result set to the original query, which is the approach adopted by the greedy diversification algorithms [44]. The framework is also extremely flexible. It allows the integration of a custom sub-query generating class, which given a query returns a collection of sub-queries. The implementation and evaluation of alternative sub-query generating classes is the main focus of the current project. In addition, xQuAD provides its own static subquery generating class, which uses predefined collection of mappings between queries and sub-queries. This class is highly useful for the evaluation of the system's effectiveness - detailed explanation of this statement is devoted to the Evaluation chapter.

The Explicit Search Result Diversification through Sub-Queries paper also describes several techniques for generation of sub-queries [44]. The first suggestion is to use a query log and analyze reformulations of the original query [40]. Alternatively, the sub-queries can be formed using the result set itself by extraction of keywords [81]. The last method closely resembles one of the approaches implemented in the Exploratory Search Engine project which are discussed in the Implementation chapter.

2.4 Summary

This chapter introduced several categories of related products, some essential IR background information and terminology, and the xQuAD diversification library. Several examples of search engines were discussed for each category of related products. It was analyzed how they handle or struggle to handle different exploratory search challenges and how these challenges are addressed by the Exploratory Search engine. The chapter familiarized the reader with some basic IR terms and processes. What the xQuAD library is and how it is used for the project was also discussed. To summarize, the relevant research chapter introduced some of the knowledge required for implementation of the Exploratory Search engine and presented some of the challenges it must overcome. The next chapter concentrates on different system requirements which allow these challenges to be tackled.

Chapter 3

Requirements

3.1 Introduction

Most of the requirements originated from the problem definition which was analyzed and split into several categories of requirements. Others were discovered during the background reading phase or as part of the weekly meetings with the project supervisor when it was discussed how the background reading concepts can be applied in practice and integrated in the project. The set of requirements was frequently refined or modified during the development of the product. In general, all requirements were focused to handle the limitations presented in sections 1.1, 1.2 and 2.1. More precisely, these limitations were related to the three main challenges which an exploratory search system should deal with, namely users unfamiliar with the domain of their goal, users unsure about the ways to achieve their goals (as they struggle to use the technology or process), or unsure about their goals in the first place [80]. Finally, weekly status reports were used as a management technique to track the progress of implementing requirements.

3.2 Functional Requirements

The purpose of the functional requirements was to capture the features expected from the system. MoSCoW was used as a prioritization framework to document the importance of these requirements [16]. The framework uses four categories to which the specific requirements of the system are allocated based on their importance. Each functional requirement is associated with a label of the form FXX.X for easy reference where the second symbol can be M, S, C, or W for Must, Should, Could or Would respectively, the third symbol is either B or F for logic tier or front-end, and the last symbol indicates the consecutive number of the requirement. Non-functional requirements are of the form N.X where the second symbol is the consecutive number. For example, FMB.1 is a functional **(F)** "Must have" **(M)** requirement for the back end **(B)** and it is the first requirement for this category **(1)** and N.1 is a non-functional requirement **(N)** and is again the first for the category **(1)**.

3.2.1 Must have

This section lists requirements essential for the project. Without meeting these requirements it will be impossible for a potential customer to use the system as intended.

Back-end:

- An efficient mechanism for storage and retrieval of data /FMB.1/.
- Ability to diversify the retrieved document set maximizing its coverage and minimizing its redundancy /FMB.2/.
- Resolve unclear queries diversifying the result set to include multiple intents of these ambiguous queries. Identify alternative mechanisms to achieve this goal analyzing their effectiveness and efficiency when compared with each other /FMB.3/.
- Add a functionality allowing "exploratory search" of a given subtopic of the result set given a subtopic selected by the client, get all results and subtopics associated with the selected subtopic /FMB.4/.

Front-end:

- Develop a web-based GUI of the search engine /FMF.1/:
- Paging functionality /FMF.2/.
- Implement a GUI mechanism allowing "exploratory search" of a given topic, including /FMF.3/:
  - an exploration icon for exploring a given topic /FMF.4/.
  - exploration history where the user should be able to go back and forward and be able to start exploration from a particular moment in the history provided that he/she loses all the history state after the particular history moment from which he/she had started to explore from /FMF.5/.

3.2.2 Should Have

This is the type of requirements which are vital for the system but the client can still use the system without their implementation even though considerable difficulties for the client are expected as a result of the absence of the should have requirements.

Back-end:

- Examine alternatives mechanisms for partitioning of the result set of a query into topics /FSB.1/.
- Consider alternatives for labeling of the identified topics /FSB.2/.
- Implement several alternative "exploratory search" algorithms and compare the results /FSB.3/.

Front-end:

- Present the partitioned into topics query results in a readable and easy to understand format with the associated to them topic labels /FSF.1/.
- Add metadata for each result specifying the category the result corresponds to /FSF.2/.
- Map each result to its score relative to the top result. Show the information using some form of a visual representation /FSF.3/.
- Allow the user to switch between alternative "exploratory algorithms" dynamically (while exploring) /FSF.4/.

3.2.3 Could Have

"Could have" requirements are mainly targeting optional features which will improve the overall experience of the client and/or provide useful additions. They are not considered particularly important for the overall success of the final product.

Front-end:

- Allow users to customize the number of clusters per search /FCF.1/.
- Allow users to select the number of results per page /FCF.2/.
- Provide a possibility the clustering mechanism to be selected from a list of possible clustering strategies (for expert users) /FCF.3/.
- Provide a possibility the "exploratory search" algorithm to be selected from a list of implemented algorithms (for expert users) /FCF.4/.
- Present a list of possible sub-query generating strategies a user can choose from to customize how the result set is diversified (for expert users) /FCF.4/.

3.2.4 Won't Have

This section comprises requirements which are recognized as not particularly useful and potentially unnecessary. Hence, they will not be implemented either because the implementation effort does not meet the benefit or because they are outside of the project's scope.

Back-end:

- Suggestions for search queries based on the current searched query /FWB.1/.
- Dynamically updating corpus /FWB.2/.

Front-end:

- Spell checker for the search query /FWF.1/.
- Auto-complete of the search query /FWF.2/.
- Location service /FWF.3/.
- Dynamic (while typing) retrieval of the result set /FWB.4/.

3.3 Non-functional Requirements

Non-functional requirements are key especially for search engines where process time considerably affects the satisfaction of the client. This collection of requirements is related to the overall behaviour and the way the client "feels" the system. They concentrate on "how" a particular functionality is implemented instead of "what" is implemented which is the responsibility of the functional requirements.

- Response time - 5 seconds to retrieve the result set per page /N.1/.
- Result set size - at most 1000 results retrieved per query /N.2/.
- Scalability - supports a corpora with up to /N.3/:
  - 807775 indexed documents.
  - 12263460 size of vocabulary.
- Reliability & failure recoverability - if the requested query is not found, present an adequate GUI response /N.4/.
- Resource usage - efficient resource utilization on the client side /N.5/.

3.4 Summary

This chapter familiarized the reader with the functional and non-functional requirements of the project using MoSCoW as a prioritization scheme [16]. In the next chapter this requirements are taken into consideration an adequate architecture of the system to be designed.

Chapter 4

Architecture

4.1 Introduction

The purpose of this chapter is to present the overall architecture of the project and to discuss the presentation and logic tiers concrete architectures introducing the reader to the different technologies used for development of the two tiers.

4.2 Overall Architecture

4.2.1 System Diagram

The overall architecture of the system, shown on figure 4.1 below, closely follows the three-tier architectural modal [14] using a thin client to meet requirement **N.5**. The three-tier architecture as the title suggests comprises of three tiers which are the following - presentation tier, logic tier and a data tier.

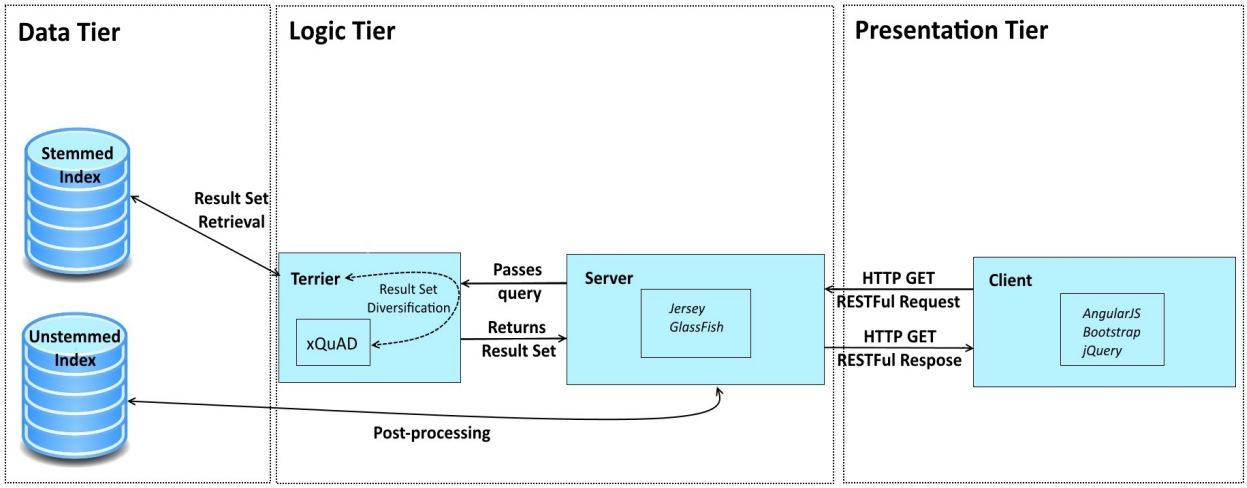


Figure 4.1: High level architecture of the application using a three-tier architectural scheme.

The Presentation tier which appears as the blue rectangle with a title Client on the diagram above includes the AngularJS presentation tier library, the Boostrap styling framework and the jQuery utility library which provides some useful scripts used to speed up the process of development. The Logical tier, on the other hand, which is written in Java using Jersey and runs on GlassFish, includes the server technologies in a combination with the Terrier and xQuAD libraries. Its main purpose is to process the search requests and to move the processed data between the Presentation and Data tiers. The Data tier, in this scenario, is formed from two indexed corpora. Their main role is to satisfy requirements **FMB.1**, **N.1** and **N.2**. The first index is solely utilized by the Terrier environment to process search requests and return back the corresponding result set to the server. The second is used only by the server for its unstemmed index to retrieve whole terms which are used for a diverse set of needs such as labeling of clusters **FSB.2**, query expansion, used for **FMB.4**, and sub-query generation **FMB.3**.

The three tier architecture is a standard choice for a web application which the Exploratory Search Engine is, as per requirement **FMF.1**. The client side is an example of a thin client architecture [14]. The term thin client refers to lightweight computer whose sole purpose is to act as a mediator between the server retrieving the result set and the user. It does not take part in any processing of the data. This is particularly important in case of search engine applications as it is much easier to scale the back-end through standard scalability techniques such vertical or horizontal scaling while such an approach is not possible on the client side. Hence, a fat client implementation can represent a bottleneck and heavily affect non-functional requirements **N.5**, **N.3** and **N.1**.

4.2.2 Workflow

Once the user has formulated his information need in the form of a query, he/she can present this to the thin client system via a search field. The entered query is then passed to the server using an HTTP GET request. The server which runs on GlassFish accepts the requests, instantiates a new session and passes the request to the Terrier environment. It accesses the stemmed index to retrieve the result set based on the query. Once the result set is ready it is directed to xQuAD, to address **FMB.2**, which uses a preselected sub-query generation algorithm to generate sub-intents in the form of subtopics based on the original query and the result set. When the sub-queries are generated and become available to xQuAD, the library sends them back to Terrier for extraction of the result sets corresponding to each of the newly generated sub-queries. Finally, xQuAD receives the collection of result sets and creates the final result set from them (**FMB.3**) eliminating the redundancy by identifying documents with similar contents (**FMB.2**). The server is responsible to accept the set of documents and to organize them into several clusters with similar documents using either flat or hierarchical clustering scheme, as per requirement **FSB.1**. After the clusters are identified, the back-end labels them utilizing a cluster labeling algorithm (**FSB.2**) discussed in the Implementation Chapter. Finally, a response object is instantiated including the set of results split into clusters, clusters‘ names, the number of pages with results, and a collection with terms for query expansion of each cluster. The last data entry is used by the client and subsequently by the server in order the “exploratory search“ functionality to meet requirements **FMB.4** and **FMF.3**. The client-side visualizes the results at which moment the user can perform an “exploratory search“ for a cluster of interest. When the user selects a cluster to be explored, based on the selected/ or default exploratory search algorithm, a new query is automatically formed. This new query is sent to the back-end. The same sequence of operations follows. The received set with documents from the client is shown on the screen to present the result of the “exploratory search“ to the user.

4.3 Data Tier

4.3.1 Introduction

The Data Tier consists of two indexed copies of the dotgov corpus. Each of them is used by different parts of the application and provide different benefits based on the concrete needs at the particular stage of the workflow.

4.3.2 Stemmed Index

The stemmed index is used only by the Terrier library. It is a standard index of the corpus mapping document terms to the terms‘ posting lists [71]. It is data structure for efficient storage and retrieval, as per requirements **FMB.1**, **N.1** and **N.3**. When the index is generated, the TermPipeline object is used to process the terms. A standard configuration of the TermPipeline removes stop words and includes a PorterStemmer which stems each term removing prefixes and suffixes. Once removed, the Terrier framework manages to match query terms to terms in the index based on the roots of the terms and hence provide more accurate list of results because “similar words generally have similar meaning and thus retrieval effectiveness is increased if the query is expanded with those terms which are similar in meaning to those originally contained within it“ [35].

4.3.3 Unstemmed Index

When the list of search results is passed from Terrier to the back-end, the unstemmed index is used to further process the request. The unstemmed index is identical to the stemmed one explained in the section above except it lacks a PorterStemmer [71]. The unstemmed index is utilized by the server as part of the clustering stage (**FSB.1**). Both the hierarchical and flat clustering algorithms make use of the unstemmed index because of its more accurate data in the form of full terms. This allows the selected algorithm to make more accurate clustering decisions. Once the set of clusters is established a labeling algorithm labels the clusters attempting to identify the information need they correspond to, as per requirement **FSB.1**. Once again the unstemmed index is applied here to generate labels containing whole terms instead of just roots contributing to requirement **FSF.1**. Another functionality which depends on the unstemmed index is the sub-query generation via the Concept Hierarchy (**FMB.3**). The Concept Hierarchy, which is explained in the Implementation chapter, also uses the unstemmed index for the generation of whole terms for the hierarchy.

4.4 Logic Tier

4.4.1 Introduction

The aim of this passage is to provide a brief a discussion of the technologies constructing the logic tier and to present its architecture.

4.4.2 Logic Tier - Technologies

Terrier

Terrier is the main building block of the project receiving queries as an input and returning a set of potentially relevant results as an output [73]. Its role is to meet requirement **FMB.1**. It provides indexing support for different file formats, different weighting models for ranking of the results, and associated meta-data for each result. Terrier receives the client‘s query through the Jersey server using a component known as a QueringManager [72]. After this it processes the request relying on the xQuAD library and the stemmed index and outputs a set of results which are passed to presentation tier by the use of an HTTP GET RESTFul response.

xQuAD

The xQuAD library, as explained in the Explicit Search Result Diversification section of the Relevant Research chapter is responsible to diversify the search results using sub-queries and to simultaneously return a result set with minimum redundancy, as per requirement **FMB.2**. As briefly touched upon in the Unstemmed Index section of the Architecture chapter the current system has a choice of 3 methods for generation of sub-queries which are used by xQuAD to diversify the search results satisfying **FMB.3**. Diagram 4.2 below illustrates the workflow of the application within the xQuAD ecosystem:

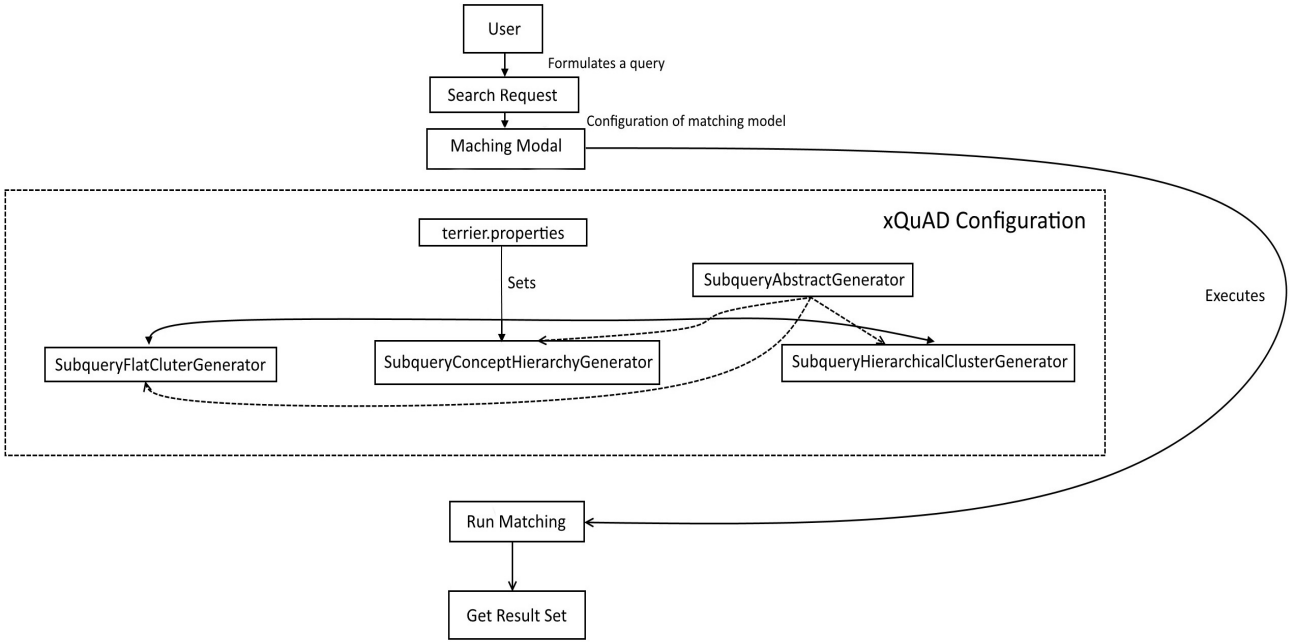


Figure 4.2: Integration of xQuAD with the Exploratory Search Engine application.

First, the user formulates a query which is passed to the server where a SearchRequest object is created to store the query terms [75]. The SearchRequest object passes through a configuration step where the matching modal is selected. The configuration of the matching model consists of two configuration settings. The first one determines the matching model name which for the current application is either "xQuADMatching" when xQuAD is used, or the default "Matching" option when xQuAD is not used. The second setting is the choice of weighting modal (BM25, PL2 etc.). After this the runMaching method of the Manager object performs the actual retrieval [72]. The runMatching functionality depends on the correct configuration of the xQuADMatching modal in the terrier.properties file where the concrete instance of the class generating the sub-queries used by xQuAD is specified. As presented in the diagram, there are three alternative ways for generation of sub-queries (meeting requirement **FMB.3**): SubqueryConceptHierarchyGenerator, SubqueryFlatClusterGenerator, and the SubqueryHierarchicalClusterGenerator. A detailed discussion of each of the algorithms is part of the Implementation section.

Jersey framework

This is the server's framework used to construct and organize the logic tier environment. A research was performed the most appropriate framework to be selected as it was identified that some of the most popular Java server-side frameworks today are Spring MVC, Jersey, JSF and Vaadin according to RebelLabs [57]. They all have their advantages and disadvantages as the next several paragraphs share these findings with reader.

First, **Spring MVC** was explored. The technology is a sophisticated full web framework built for scalability equipped with a presentation tier template engine, and a back-end framework module [57]. Unfortunately, this sophistication comes with a steeper learning curve and a much more detailed and complex documentation. Spring MVC is ideal for a long term project but it was decided that the steep learning curve outweigh the advantages of the framework.

Another option was to use **JSF** which is part of the Java EE stack, hence it is expected not to be deprecated soon. It has excellent maintenance and a large community [57]. However, once again its steep learning curve, lack of flexibility, and complexity as elaborated in the TheServerSides article - *Five drawbacks to choosing JSF as your web application framework* were sufficient reasons an alternative solution to be considered [77].

The third candidate for a logic tier core framework was **Vaadin**. The framework, at first, looks like the ideal candidate with its flat learning curve, fast development, good documentation and active community [57]. Despite all of these advantages Vaadin has one major drawback - it does not scale and, as a consequence, does not meet **N.3**, which in the world of search engines is an unacceptable trade-off.

The last option which has been selected as the most appropriate logic tier framework for the current project, is **Jersey**. It is a simple RESTFul Web Services framework which follows the JAX-RS standard for annotation-based REST handlers [57]. One drawback is that it has a smaller community but its thorough documentation compensate for this disadvantage.

GlassFish

GlassFish is the application‘s server. Developed by Oracle it is expected to have good maintenance and not to be deprecated soon. In addition, it has good documentation and large community [39]. Finally, Jersey comes with a handy User Guide explaining how to set up a Jersey application with Glassfish [62].

4.4.3 Logic Tier - Architecture

A closer look at the back-end server, the logic tier, reveals that it does not simply act as a mediator between the Terrier‘s library and the presentation tier. It is the sever-side‘s responsibility to configure all libraries and indices to work together, to synchronize access and allow simultaneous requests to the indices, cluster the data using either flat or hierarchical clustering, label the clusters, choose a sub-query generation scheme for xQuAD, calculate the relative scores for each result and finally instantiate a POJO Response object accumulating all of these data entries, which is subsequently sent to the presentation tier to be visualized. Diagram 4.3, available on the next page, shows the components listed above and the way they interact with each other.

Workflow

First a search request is sent by the presentation tier to the QueryResource class. The QueryResouce is responsible for the receipt and processing of the query request from the presentation tier through an HTTP GET RESTFul mechanism. This communication mechanism is available via the integration of the Jackson library serializing POJO objects to JSON and vice versa [36]. The QueryResource manages the entire request process relying on the support of the other classes shown on diagram 4.3. After receiving a search request from the presentation tier the QueryResource constructs a SearchRequest object and determines if diversification of the result set via xQuAD is enabled (the setting is sent as a parameter to the HTTP request). If it is enabled the matchingModal field of the SearchRequest class is set to xQuADMatching, otherwise the default retrieval configuration of Terrier is utilized.

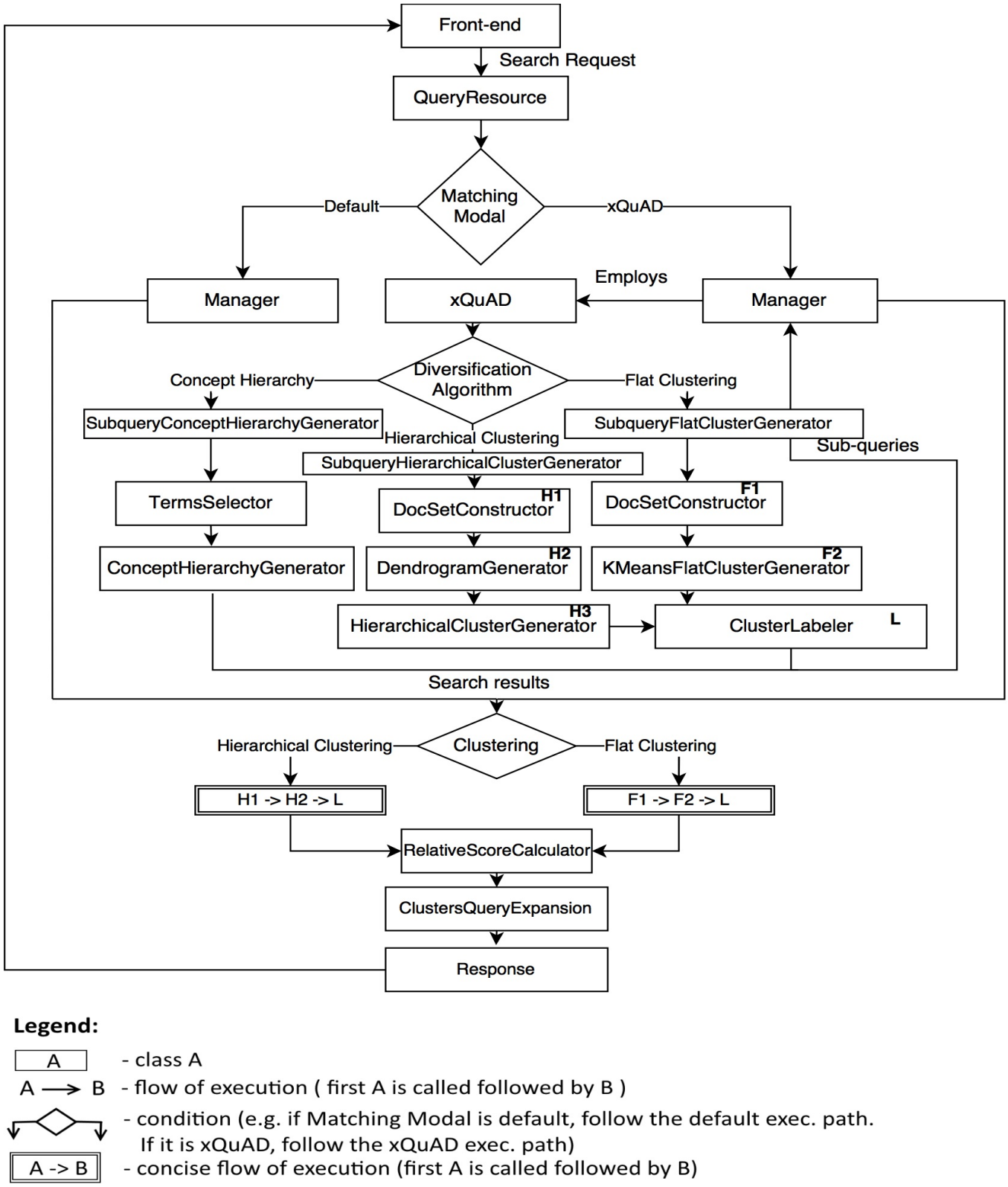


Figure 4.3: Possible execution paths of the logic tier.

If the default configuration is used, the Manager class [72] passes through several retrieval stages (part of Terrier environment) and eventually returns a ResultSet object. The object stores the document ids, the total number of results, scores for each document and its meta data. However, if the matching modal is xQuAD, the Manager employs it. First, the manager performs an initial retrieval of the search results without using xQuAD. Once the results are recevied based on the selected diversification algorithm, which is again passed as an HTTP get parameter, the Manager employs either the SubqueryConceptHierarchyGenerator, SubqueryHierarchicalClusterGenerator or the SubqueryFlatClusterGenerator to generate sub-queries for xQuAD (**FSB.1**). If the selected diversification algorithm is the Concept Hierarchy, first the terms for the hierarchy are selected through the TermsSelector class from the initial search results passed from the Manager to the SubqueryConceptHierarchyGenerator. After this the concept hierarchy is constructed using these terms and finally terms from the hierarchy are selected as sub-queries based on their positions in the hierarchy. A more detailed explanation of this algorithm is part of the Implementation chapter. Alternatively, if Hierarchical Clustering is chosen as a diversification algorithm, the SubqueryHierarchicalClusterGenerator is employed by the Manager. The SubqueryHierarchicalClusterGenerator first calls the DocSetConstructor which generates *Document* objects from the initial search results passed to the SubqueryHierarchicalClusterGenerator from the Manager. These *Document* objects store statistics for the each result in the initial search results. They are used a Dendrogram to be generated from the DendrogramGenerator. This is a data structure provided from the LingPipe library, which represents a binary tree over the result set which is manipulated by the hierarchical clustering algorithms to generate clusters. For more information regarding this process appendix C can be read or the official documentation to be examined [63]. The Dendrogram is processed by a HierarchicalClusterGenerator to partition the search results into clusters. A label for each cluster is assigned by the ClusterLabeler **(FSB.2)** . This is achieved through the *Automatic Labeling algorithm* invented by Alexandrin Popescul and Lyle H. Ungar and thoroughly discussed in the Implementation chapter [38]. The labels are used as sub-queries which are passed back to the Manager for a secondary retrieval. The final diversification option is the flat clustering. The steps are identical to the SubqueryHierarchicalClusterGenerator but this time the KMeansFlatClusterGenerator (discussed in the Implementation Chapter) is used to generate the clusters.

After the diversification the new diversified search results are categorized into several categories where the number of categories is based on the user‘s choice, default number of categories, or an automatically determined number. As part of the HTTP get request, a parameter is passed determining the categorization algorithm. If Flat Clustering is selected, the search results are split into k categories by the KMeansFlatClusterGenerator, where k is an HTTP get parameter chosen by the user, or a default value of 3. Alternatively, if the choice is Hierarchical Clustering, the HierarchicalClusterGenerator automatically determines the number of categories (it is discussed how this happens, in the Implementation Chapter) and clusters the search results. H1, H2, F1, F2, and L indicate classes associated to hierarchical clustering, flat clustering or cluster labeling as shown on the diagram. To avoid repetition the H1-H2-L and F1-F2-L blocks are used to denote that the corresponding classes are processed in this order. For Hierarchical Clustering, the the xQuAD diversification process involving classes H1, H2 and L is repeated in the same order - H1-H2-L. For the flat clustering the approach is identical.

After the search results are categorized, they are associated with relative scores from the RelativeScoreCalculator. The score calculator is a simple utility class which accepts the scores of the documents in the result set and returns their scores relative to the top result in the result set. Its purpose is to transform the scores of the documents in the format expected by the graphic bars on the presentation tier in order this meta-data to be visualized, as per requirement **FSF.4**. Finally, key terms, selected via a query expansion technique, are selected for each cluster in the result set. These terms are used to provide the first implementation of the “exploratory search“ functionality to satisfy requirement **FSB.3** (a detailed discussion is part of the Implementation chapter). These terms, relative scores for each result in the search results and the search results are gathered in a Response POJO object [67] which is sent to the client as a final step of the search request.

4.5 Presentation Tier

4.5.1 Introduction

The entire presentation tier is handled by AngularJS and Bootstrap [49] [56]. Because of its mainly data visualization purpose the presentation tier is quite straightforward. AngularJS is integrated for its inversion of control feature allowing a more straightforward organization of the presentation tier, while the Bootstrap is solely used for visualization purposes [66]. The client side of the application is built as a single-page application (SPA) which means that the whole application is mounted at a single URL [29]. This usually means that the navigation within the single-page application is only handled by AngularJS and its $route service [53]. The entire communication with the logic tier is performed as series of HTTP GET RESTFul request and responses [42]. Once a response is received only the affected part of the DOM object is updated increasing the responsiveness of the presentation tier, as per requirement **N.1**.

4.5.2 Presentation Tier - Technologies

AngularJS

AngularJS is an extremely popular presentation tier framework [61]. It is developed by Google and because of the framework‘s inversion of control and superior injection mechanism [66][64], the development of large presentation tier applications using the framework becomes straightforward.

Perhaps one of the most popular and highly useful features of Angular is its famous two-way data binding [47]. Whenever an AngularJS’s variables is changed in the HTML, a digest cycle is triggered. The process evaluates the entire state of the Angular system updating only the affected parts of the DOM instead of the whole DOM[54], as per requirement **N.5**.

Bootstrap

Bootstrap as most likely the reader is aware, is a Web UI library speeding up the process of building presentation tier applications by providing a range of UI components which can be reused and/or partly configured. It also has a JavaScript module which allows the addition of some more complex components such as pop-ups, for example. In the case of the Exploratory Search Engine it is mainly used for its renowned Grid System, beautiful glyphicons, and pop-up modals.

4.5.3 Presentation Tier - Architecture

Presentation tier architecture is primarily driven by the AngularJS structure. It strictly follows the Modal-View-Controller pattern [65] with the added additional AngularJS organizational logic as shown on Figure 4.4. As it can be seen from the figure the presentation tier is split into three interconnected parts. The next three subsections discuss these components.

The View

The View is the component observed by the user through which the communication with the client occurs. Once the View state is changed by the user, the two way data-binding of AngularJS changes the state of the modal component and triggers a digest cycle. The digest cycle goes through all AngularJS watchers to detect if the watched value has changed since the last digest cycle [52]. If there is a watcher registered for this variable and the value is changed since the last digest cycle run, the so called listener function mapped to each watcher is called [52]. [18]. The behavior of the function is implemented by the presentation tier developer in the Controller.

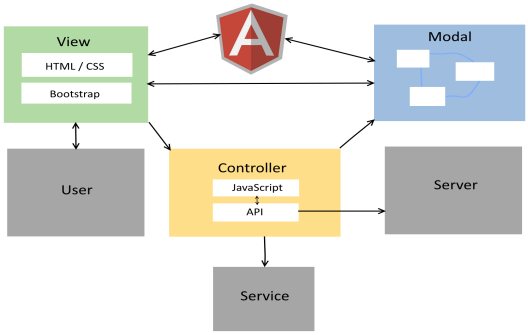


Figure 4.4: Presentation Tier architecture.

The Modal

The modal component is where the state of the presentation tier is kept. In the AngularJS world this is the $scope object which can simply be seen as a key-value pair storage system depending on both the View and the Controller components. The $scope can be hierarchical, as it is possible to nest Controllers within each other or to have other elements associated with their own $scope object such as Directives [48].The Modal is updated as a result of a user‘s change of the View or a direct JavaScript modification of the $scope within the Controller or any of its dependencies such as services or directives [55].

The controller

The controller is the engine of the presentation tier architecture responsible for the entire computational logic. It can send commands to the modal to update it and to manually trigger digest cycles updating the View component as a result. All additional functionality such as services, directives and RESTFul $http services are injected through the AngularJS injection mechanism to the Controller‘s body so that they can be accessible[50]. The controller also acts as a mediator between the user and logic tier. This is possible through the $http service. Its purpose is to facilitate communication with remote HTTP servers via a XMLHttRequest or by JSONP [69][19].

4.6 Summary

The architecture chapter introduces the reader to the overall design of the system. It presents several high level diagrams which are used the workflow and system components to be shown. It also presents the presentation and logic tiers concrete architectures. Furthermore, different libraries, technologies and design patterns used within the project are discussed. The next chapter concentrates on the concrete implementations of some of the key system components and reveals some of the challenges overcome as part of the implementation phase.

Chapter 5

Implementation

5.1 Introduction

This chapter goes into details of how some of the more challenging components of the system are built without taking into consideration the straightforward functionalities or features already presented in the required depth in the previous chapter.

5.2 Logic Tier - Implementation

In this section the flat (5.2.1), hierarchical (5.2.2) and concept hierarchy (5.2.3) algorithms implemented in the logic tier, to perform the categorization of the search results and/or their diversification through xQuAD are presented. This diversification is achieved by xQuAD which requires sub-intents of the original information need (search request) to be passed as input to the library. The flat, hierarchical or concept hierarchy algorithms are used to generate these sub-intents via several sub-query generating classes analyzed in subsection 5.2.5 and with assistance of a cluster‘s labeling algorithm explained in subsection 5.2.4.

5.2.1 Flat Clustering

The clustering algorithms utilized by the Exploratory Search Engine project use the LingPipe library to provide flat and hierarchical clustering of the result set [63]. The framework utilizes the Vector Space Model, introduced in Appendix B, to cluster the documents [27]. The flat clustering is built using an algorithm known as K-Means. It is an example of an optimization algorithm which starts by randomly choosing K cluster seeds assigning document points to the closest centroid (relying on the similarity formula above). At the end of each stage the centroids are recomputed using the following formula:

$$\text{Assing points ->} \quad Cluster(\vec{d_i}) = \underset{1 \le j \le K}{\text{Argmin}}\, d(\vec{d_i}, \vec{c_j})$$

$$\text{Recompute centroids ->} \quad \vec{c_j} = \frac{1}{n}\sum \vec{d_i}$$

Figure 5.1: K-Means Clustering Algorithm.

The process continues until Convergence of the centroids. The KMeansFlatClusterGenerator class of the Exploratory Search project provides the KMeans flat clustering functionality through the lingPipe's KMeansClusterer class [33].

5.2.2 Hierarchical Clustering

The hierarchical clustering functionality provided from LingPipe via its CompleteLinkClusterer class [32], was used to split the result set into categories and to generate sub-queries for xQuAD [63]. More information regarding how LingPipe was employed to perform hierarchical clustering and how the clusters are stored is part of appendix C. The hierarchical clustering was used during a pilot user experiment to categorize the search results. Unfortunately, as a result of the experiment a considerable issue was discovered. Query result set documents are expected to have similar statistics which represents a difficulty for a clustering algorithm which is trying to identify different statistics among the documents and to split them to categories based on these statistics. As a result most of the query result sets were split into categories extremely unevenly. A frequent case was to have a category with 95-98% of all result set documents allocated to it. The issues was investigated, as it was identified that the Dendrogram (explained is appendix C) storing the clusters is extremely nested. In other words, if the Dendrogram is partitioned to two Clusters based on the similarity score between the documents, one of the clusters will contain 1 document and the other cluster will contain all other documents. If it is partitioned into four clusters, three of them will have size one and so on until a deeper level in the Hierarchy is reached. A pictorial example of the problem is provided on figure 5.3:

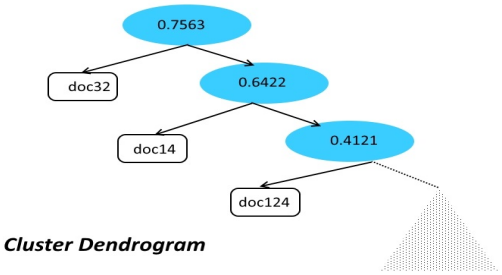


Figure 5.2: Clustering Dendrogram.

The value stored by all internal nodes represents the minimal distance score between the documents in the sub-trees. A closer look at the Dendrogram reveals a recursively repeating pattern. It was found that if the tree was traversed deeply enough the pattern disappears and the documents become more evenly grouped to clusters with a smaller distance to each other at the same time. This smaller distance was a key factor making the interpretation of the documents in the result set easier - the smaller the distance the larger the number of elements within the cluster to a certain distance at which moment the number of documents begins to decrease again. Once this trend was identified the problem was easily solved by going recursively deep down the tree by decreasing the minimal distance score iteratively until there are at least 3 categories each of them having 10% or more of the result set documents. All other clusters having less than 10% were merged into a single cluster. The graph below illustrates this observation:

5.2.3 Concept Hierarchy

The Concept Hierarchy generation algorithm used for the identification of sub-queries for xQuAD, satisfying requirement **FMB.3**, introduced in appendix D, and arguably the most complex feature of the current application,

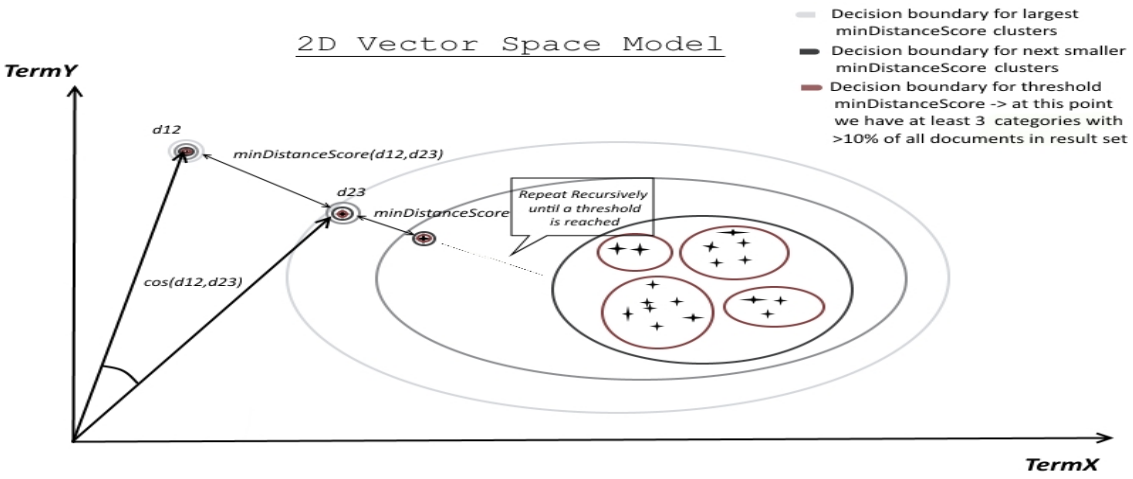


Figure 5.3: Distribution of the search results on Vector Space.

is clarified in this section. Three of its more challenging aspect are examined, namely some critical parts of the terms selection logic, the data structure used to store the Concept Hierarchy and its weighting scheme and the efficiency of the Concept Hierarchy generator. In summary, the Concept Hierarchy identifies key terms via a Terms Selection process from a given set of documents and structures them hierarchically where elements higher in the hierarchy are more general and lower in the hierarchy are more specific. The structure provides a mapping between elements higher in the hierarchy and lower in the hierarchy to indicate parent-child relationship. This relationship is used to generate sub-queries for xQuAD. The exact process is explained in the Subquery Generation Algorithms section of the current chapter.

Terms Selection

The TermsSelector class is responsible for the extraction of significant terms from the result set. Its theoretical aspect are presented in appendix D. However, when the theory was put into practice several issues were arose. First, the *Query Expansion* (QE) method [74] for selection of terms did not initially worked as expected. It outputted too general or too specific words. The general words did not reveal any potential sub-intents, hence it was of little benefit to use them for sub-query generation, which was the final goal of building the Concept Hierarchy data structure, as they would not diversify the result set, as per requirement **FMB.3**. The set of too specific words, on the other hand, consisted of words including names of products, companies and other similar overspecialized terms. Some of these words even included digits. Utilizing such a set of terms in an attempt to achieve diversification of the result set actually leads to overspecialization, in contradiction to requirement **FMB.3**. It expectedly increases the precision of the query results trading off its recall [15]. Such a compensation is undesired in an exploratory engine, as it is expected the user's goal to be to examine a diverse set of relevant documents [28]. For these reasons an additional filter was employed:

```
if (tq.getTerm().matches(".*\\d+.*") || tq.getTerm().length()<5)
    continue;
```

Figure 5.4: TermsSelector class - filter of too general or too specific terms.

The first term of the expression filters out terms which contain digits in order to exclude some of the over-specialized words. The second term handles too general words. It was detected through empirical analysis that

terms with less than 5 letters usually did not represent any concrete aspects of the information need (were too general) for which reason they were filtered out, as per requirements **FMB.2** and **FMB.3**. The Term Selector, in addition to the QE algorithm, also selects terms via the following selection criterion, explained in more detail in section appendix D:

$$X_r/X_c > 0.1[43]$$

However, once again the algorithm, as explain in section *2.4.1*, did not give the expected behavior. 10% occurrence ratio between the result set and the collection turned out to be very high threshold for the current corpus, as the average number of terms passing through the filter was empirically estimated to range from 2 to 4 terms. Such a low ratio was inappropriate for the current purpose of the Concept Hierarchy so the occurrence ratio was changed to 1%. The new version was less strict, as it allowed some terms with very low local frequency (frequency of a term within the result set) to also pass the filter. These terms did not possess any distinctive characteristics of the result set and could only add *noise* to the Concept Hierarchy. As a consequence the filter was adjusted once again to acquire its final form presented below:

$$X_r/X_c > 0.01 \cap X_r > 30$$

This last filter had an acceptable sieving barrier allowing a large enough number of terms to pass while filtering out too general or overspecialized words, and words with too low local frequency.

Concept Hierarchy Data Structure

The concept hierarchy data structure is designed to address the implementation needs of the Concept Hierarchy algorithm providing an easy access and fast generation, as per requirement **N.1**. It also addresses requirement **FSB.1** Once build, it was employed to input relevant terms for the SubqueryConceptHierarchyGenerator component which generates sub-queries for xQuAD based on the Concept Hierarchy of the result set.

The data structure represents a directed acyclic graph, as it is further explained in the Mark Sandersons and Bruce Crofts *Deriving concept hierarchies from text* article [43]. In addition to the original design of the DS, it was necessary *a weighting model* to be implemented to weight each parent-child relationship as shown on figure 5.5 below:

$$edgeWeight = P(parent|child) + P(child|parent)$$

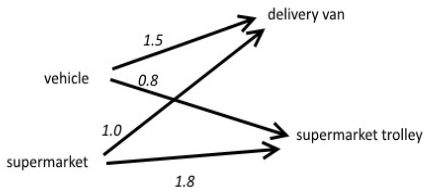


Figure 5.5: Part of the Concept Hierarchy DS.

A weight of an edge is a value between 0 and 2. The larger the value the higher the probability is the child term to be a child of the parent. In the example, the “vehicle“ and the “supermarket“ terms are examples of

general terms and the “delivery van“ and “supermarket trolley“ are instances of *specific* terms of this *general* terms. The “delivery van“ has a lower weight as a child of the “supermarket“ compared to the case when it is a child of the “vehicle“. This observation follows from the fact that the “delivery van“ is *more closely related* to the term “vehicle“ than to the term supermarket, even though the second relationship is still logical. However, the “supermarket trolley“ has a larger weight when regarded as a child of the “supermarket“ term. The weight of the edges is calculated with the equation shown on figure 5.5 using the theory introduced in appendix D. The first term verifies if the child term has a *more concrete* meaning than the parent, in other words if the documents containing the child terms are a subset of the documents containing the parent term. The larger the value of the first term the higher the probability the child variable to keep a word which has a specific meaning of the parent term. The second term of the equation checks how “important“ the child is for the parent. In other words, how large the subset is within the super-set. The larger this value is the *more general* the meaning of the child term within the parent context is. Hence, the child can be regarded as a more “direct/important“ child of the parent. For example, based on the pre-defined terminology of a parent and a child, both a “car“ and a “tire“ are children of a “vehicle“. However, they can also be seen in the following context: a “car“ as a child of a “vehicle“ and a “tire“ as a child of a “car“. Diagram 5.6 below presents a visual clarification of *edgeWeight formula* introduced in this paragraph.

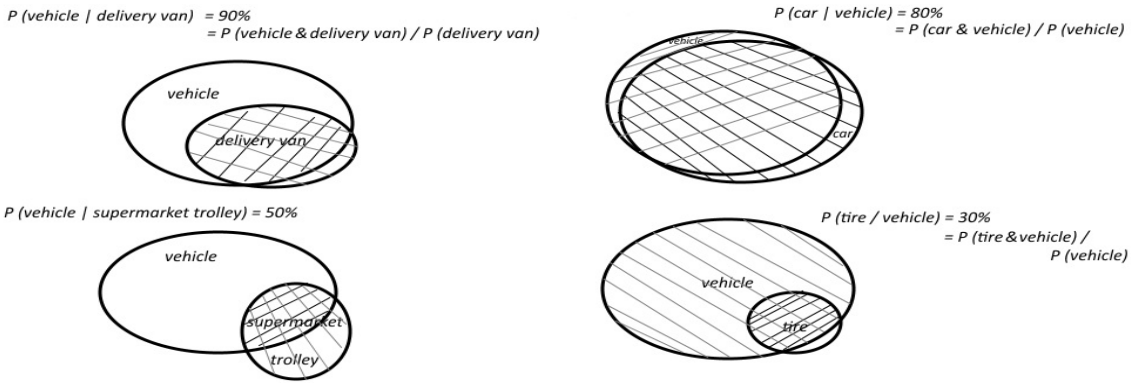


Figure 5.6: Visualization of the edgeWeight formula using supersets and subsets to show the importance of the two terms in the equation.

5.2.4 Automatic Labeling of Document Clusters

The reason the searched result is split into clusters is to be shown to the user the different topics the results of his or her query can be partitioned to. However, without labels clusters themselves are of little use because this means that the user should manually go through each document in the cluster to find out what the topic of the cluster is and then to refine his or her query to reflect this additional knowledge but this is what the user has to do anyway if the result is not split into clusters. Therefore, the correct labeling of the clusters is essential.

Labeling of clusters is notoriously hard as most of the labeling schemes use the cluster centroid to come up with a satisfying label (the centroid is the document in the centre of the cluster). For example, one approach is to use the frequency of the terms in the centroid document looking for terms with high frequency to form a label. Unfortunately, this approach suffers some limitations because the words selected in this way usually have a very general meaning not representing the specific context of the cluster.

The approach adopted by the presented system, on the other hand, tries to extract as much information about the cluster as possible gathering statistics from all documents of a particular cluster to generate a label. The algorithm closely follows Alexandrin Popescul and Lyle H. Ungars suggestion in their paper called *Automatic Labeling of Document Clusters* [38]. The approach considers two aspects of a term in the cluster - its frequency and its predictiveness. Both characteristics are taken into account in the following formula indicating the weight of a particular term:

$$p(word|class)\frac{p(word|class)}{p(word)}$$

Figure 5.7: Frequent and Predictive Words Method - weight of a term formula [38].

The formula can be partitioned into two parts and each part to be examined separately. The first term p(word | class) / p(word) remind of TF-IDF [27]. This term allocates more weight to terms occurring frequently in a given cluster and infrequently in the entire collection, which is exactly a characteristic required by a label of a cluster; p(word | class) demonstrates the frequency of a word in a given cluster while p(word) is the frequency of the word in the collection. Using the first term of the formula *the collection specific stop words* are filtered out. This is a term used by Alexandrin Popescul and Lyle H. Ungar to describe the collection of words which appear frequently in the cluster but it is not a subset of the stop words set and its members do not appear frequently in the collection. On the other hand, the second term p (word | class) whose meaning was already described above is used as it is experimentally proven that the results are better using it [38]. Once the weights of all terms in the cluster are calculated, the label can be directly formed from them using for example the first x terms with highest weight. Using the formula words which appear frequently in the cluster and are at the same time specific to the cluster are selected. This allows the drawbacks of the previous labelling approach to be avoided first by looking at the entire cluster not simply the centroid and second by filtering out obscure and non-cluster specific words.

During the evaluation stage several issues related to the effectiveness of the labels were identified. For some queries, there were clusters with the same names. Initially the clusters‘ labels were considered too specific by the users - containing names of people, places or specialized terms which cannot be helpful for a user who is not an expert in the searched field. After the implementation was modified using techniques similar to the ones used in seciton 5.2.3 the the opposite problem appeared - the labels became too general, almost synonyms of each other (not independent sub-topics). The final problem was that some of the category labels were only related to one or two of the clusters‘ documents in which the term used as a category label appears many times. These issues were addressed by refining the Alexandrin Popescul and Lyle H. Ungar labeling algorithm as follows:

First, it was not a surprising fact that the same labels were generated for some of the clusters. All clusters included relevant documents for the same query. Hence, it was expected that their statistics is quite close. To resolve the issue the p(word) frequency was changed with the frequency of the word in the results set. This was a successful strategy as it reduces the weight of terms appearing many times in the result set. In other words, it filtered out popular words with the same statistics among the clusters. This was the same set of words leading to clusters with the same labels. An additional measurement to prevent repeating clusters was also added. All currently selected clusters‘ labels were added to a set and all new cluster label candidates were checked for presence in this set. If the candidate term was in the set, it was ignored in the selection.

The second issues was related to cluster labels which are too general or two specific. It was identified through debugging that it was possible terms with very low p(word | class) (these terms are very specific) to have high weight because of their low result set frequency (the second term on Figure 5.7). Even though this did not mean that the algorithm was wrong, it was problematic, as it could cause difficulties for people who were not experts in the searched field to understand the category‘s label. To resolve the problem a filter ignoring all terms with p(word | class) smaller than 40 from consideration for cluster labels was added. The filter performed well ignoring very specific terms but the next user study led to the conclusion that the new cluster labels were too general - almost synonyms of each other. A final refinement was made to make the filter less strict. The new filter ignored terms with a cluster frequency smaller than 20. This lead to the removal of the too specific terms but at the same time the labels were not too general either.

The final and most complex modification of the labeling algorithm was related to the category labels which were only related to one or two of the clusters‘ documents. An investigation of the issue found that such a

result was observed when one or two documents contained a term a very large number of times leading to a high p(word | class) even though the term is not contained (many times or at all) in the other documents which were part of the cluster. A technique known as *Term Frequency Absorption* used by PL2 to address a similar challenge was integrated [1]. *Term Frequency Absorption* changes how the p(word | class) function grows. The initially presented (Alexandrin Popescul and Lyle H. Ungar's) p(word | class) function grows linearly with the increase of terms within the cluster, while the new/modified function used for the current project grows logarithmically with the increase of the term's frequency within the document.

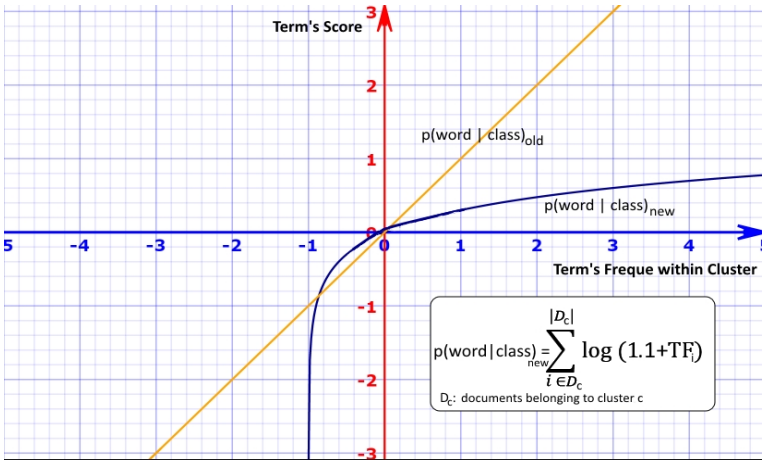


Figure 5.8: Term Frequency Absorption.

This allows documents where a particular term occurs extremely frequently not to deviate the final result for the cluster's label, while at the same time terms occurring to a moderate extent across all documents are not penalized. This formula generates a label where the label term occurs many times across all/most of the documents part of the cluster. The 1.1 constant is added to the TF to prevent the formula to equals 0, which can lead to division by 0 later in the implementation code and can cause unexpected behavior. The p(word) term is also refined to address the changes of the p(word | class) term. The new p(word) term is:

$$p(word)_{new} = |D_c| log(1 + \frac{TF_r}{|D_c|})$$

The new p(word) can be interpreted as the absorbed term frequency within the result set where the term appears uniformly across all result set documents. Such a change is required - we cannot simply get the logarithm of p(word) because for p(word | class) we have a sum of logarithms for all documents where the term appears.

5.2.5 Sub-query Generating Algorithms

The objective of this section of the Implementation chapter is to reveal in further detail the implementation of the three sub-query generation algorithms, as per requirement **FMB.3**, and to discuss some of the implementation challenges overcome during the development process.

Sub-query generation via the Concept Hierarchy

The need for a weighting model for the Concept Hierarchy DS arisen as a result of the integration of the Concept Hierarchy in the xQuAD library. The DS is used for the selection of sub-queries which are employed by xQuAD the searched result to be diversified. The sub-query selection process is the following: the children of all query

terms are retrieved from the Concept Hierarchy DS and are used as sub-queries which are passed to xQuAD for diversification of the result set. The drawback of this approach becomes apparent when some of the query terms have a large number of children. This considerably slows down the diversification process and the entire query retrieval as a result violating non-functional requirement **N.1**. As a consequence, the need for sorting of the children in the Concept Hierarchy and the use of only the children with highest weight emerges. An empirical analysis identified that no more than 5 sub-queries can be processed by xQuAD, with the current corpus and computational resources, for less than 5 seconds in order **N.1** to be met. However, according to the initially implemented algorithm the children of each query term had to be used as sub-queries, hence the number of children per query term had to be dynamically determined based on the number of query terms following the logic illustrated in figure 5.9 below:

```
int maxNumberOfSubqueries = 5;
int numberOfChildrenPerQueryTerm = Math.max(1, maxNumberOfSubqueries / queryTerms.length);
```

Figure 5.9: Number of children per query term - Refinement.

However, the formula had a drawback. The number of total sub-queries exceeded the maximal possible number of sub-queries in order **N.1** to be met when the number of query terms was greater than five. Unfortunately, the only alternative option for refinement was to have a mechanism to determine only 5 query terms for which the child with highest weight to be used as a sub-query but this approach results in a deviation from the expressed information need by the user. For this reason, the formula was left as shown on figure 5.9 above. As a summary of this section, it can be pointed out that the use of the Concept Hierarchy as a method for sub-query generation can potentially slow down the system below the acceptable level (**N.1**) providing that the number of query terms exceeds five.

Subquery generation via Flat & Hierarchical Clustering

Except a sub-query generation based on the Concept Hierarchy algorithm, two alternative mechanisms were also implemented to meet requirements **FMB.2**, **FMB.3** and **FSB.1**. The first method depends on the *Flat Clustering* to identify sub-intents while the second relies on the *Hierarchical Clustering* to perform the same. They both first split the result set into n clusters. After this the clusters are labeled and finally the labels themselves are used as sub-queries which are passed back to xQuAD for diversification. These two applications of the clustering algorithms reveal two additional approaches solving the sub-query generation problem (**FMB.3**).

5.3 Presentation Tier - User Interface and Implementation

The implementation of the client side includes a collection of several presentation tier technologies and GUI components. The presentation tier technologies were discussed in the Architecture chapter while the research related to the graphical design is the topic of appendix E. The purpose of this section is to briefly introduce the reader to the GUI workflow and to describe the different features listed in the requirements section. Finally, the "exploratory search" functionality is presented in more detail discussing its implementation and user interface. Introductory figures showing all parts/functionalities of the GUI are available as part of appendix H.

5.3.1 GUI Design and Workflow

Initially two alternative GUI designs were considered as presented in appendix E . The first one is a classical search engine design. The alternative design is a modification of the Scatter/Gather Clustering design [10] again

introduced in appendix E. After a discussion with the supervisor and due to reasons explained in appendix E the last GUI design was ignored and the more classical one was selected, the workflow for which is as follows . Whenever a user enter a search query and clicks the search button the *ng-click* AngularJS handler is triggered [51]. The handler passes the control to the custom AngularJS *RetrivalService*. The service directs the request to the appropriate method within the service, which in this scenario is the getResult() method. The method depends on another AngularJS service - the $http service [50]. This service performs a standard HTTP GET RESTFul request to the server appending the query and different settings to the URL of the HTTP request. When the response is received the state of $scope.results, which is the object storing the data received from the HTTP GET RESTFul response, is updated within the successCallback function of the $http service. After an execution of the function the AngularJS system triggers a new digest cycle (automatically invoking the $scope.digest function), so that the $scope.results changes propagate properly to presentation tier in order the result to be rendered and presented to the user [54]. Figure 5.10 is a summary of the workflow described above.

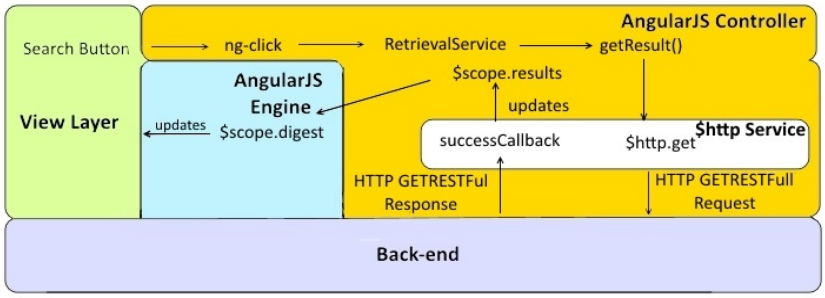


Figure 5.10: Presentation Tier workflow.

5.3.2 Relative scores

The relative scores in the form of Graphical Bars (discussed in appendix E) for each result were implemented, per requirement **FSF.3**,and can be seen on Figure 5.11. Initially, there was a discussion if the relative scores would be useful or not as they can be partly deceptive because they are relative to the top result (do not indicate absolute relevance score). Hence, even if all results on the first page have a relative score in the range 90%-100% percent they can still be irrelevant given that the top result is irrelevant. However, for a user not familiar with the searched domain, visual presentation of the scores can still be helpful. Once the first document is read the user can much more easily make an objective judgment whether the next documents on the page are relevant not without reading their abstracts. This is particularly important, as it is possible that the user cannot easily understand the documents‘ abstracts without further research considering that the information need corresponds to a specialized field the user is not an expert in (this is the case, as it is assumed that the user performs an exploratory search).

5.3.3 Categorization of the Search Results

The clustering functionality, partitioning the documents in the result set into categories, is implemented on the client side, as shown on figure 5.11. From the screenshot several new components can be identified. A category meta-data is added to each result showing the cluster it corresponds to (meeting requirement **FSF.2**). The field is colored in green for easier distinction with the document extract and visual comfort in agreement with the color of the relative score bar. A new Category component is added showing labels of the categories, the result set is clustered to, as per requirement **FSF.1**. Initially, each of these labels included the 3 terms with highest weight for the corresponding cluster (calculation of the weights is explained in section 5.2.4). However, this approached was abandoned in favour of labels using only the term with the highest weight to further satisfy requirement

FSF.1. This decision was necessary due to the frequently observed lack of relationship between terms used for labeling a particular cluster.

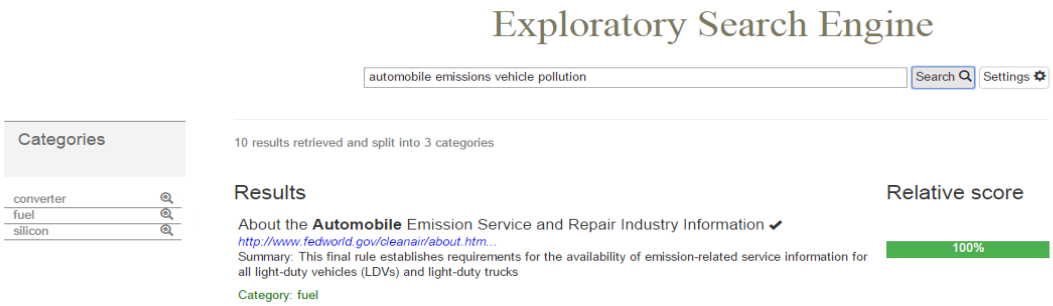


Figure 5.11: Implementation of the clustering functionality as part of the GUI.

5.3.4 Exploratory Search and Recursive Exploration

After the addition of the clustering feature the exploratory search logic was also integrated, as per requirement **FMF.9**. The responsibility of this functionality is to further support the user in his process of getting familiar with the searched field he is exploring and assist in the reformulation of the initial query. The exploratory search feature describes the functionality which retrieves new search results when a particular category is selected from the list of categories for the current search results. The functionality has been implemented using two alternative algorithms described in the next two sections.

Search based on the category‘s name

If *Search based on category‘s name* is selected as an exploratory search algorithm from the settings menu and the user clicks on a category from the list of categories for the current search results, an HTTP GET request to the server will occur. The request is the same as this one for a standard search request. However, the query value of this request will correspond to the initial query with the category label added. It was initially empirically experimented only the category name to be used but this method did not give satisfying results, as it was observed that sometimes the selected category, which is a subtopic of the initial information need, can be as well a subtopic of another topic. This case can lead to significantly bad results in case that the category covers this other topic more broadly than the initial information need. Figure 5.12 illustrates this observation:

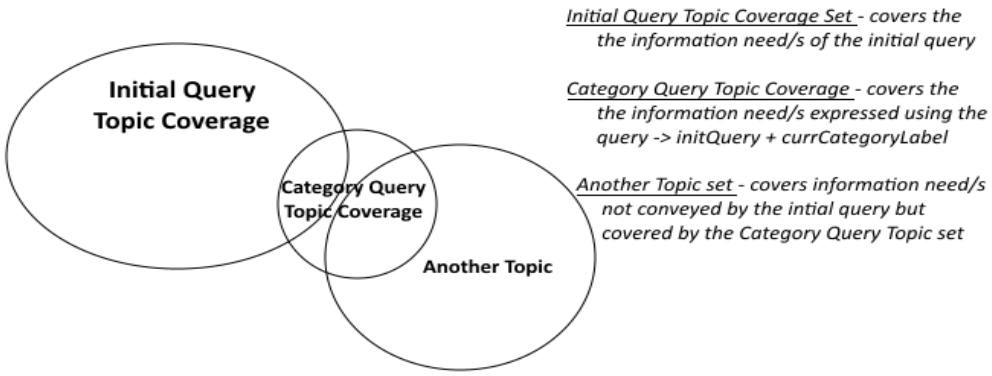


Figure 5.12: Topic coverage of the initial search query vs coverage of an automatically generated query (via QE) for a subcategory of the search results.

The exploratory search functionally allows the result of an exploratory search itself to be exploratory searched - *recursive exploration*. In a scenario when *a search based on categories's name* is selected, the initial query, the name of the first explored category and the name of the second explored subcategory are gather in one query separated by spaces. An example of *recursive exploration* is shown on Figure 5.14.

Search based on the content of the searched document

The second method for exploratory search, namely *Search based on the content of the searched document*, depends on a Query Expansion(QE) approach using the DFRBagExpansionTerms class provided by Terrier [70]. The class is particularly useful in an exploratory search scenario as it is able to identify key words in a set of documents.If it is populated only with documents from the cluster under interest and its method getExpandedTerms is called, the set of returned terms is a cluster-specific list of terms. Hence, these terms can be appended to the original query and can be used as a query for an exploratory search. Initially, the number of terms selected through QE was chosen to be 3. However, this resulted in the same terms being selected for all clusters. Such a behavior was expected taking into account that all of the documents in the result set are supposed to be similar. Figure 5.13 below visualizes this observation:

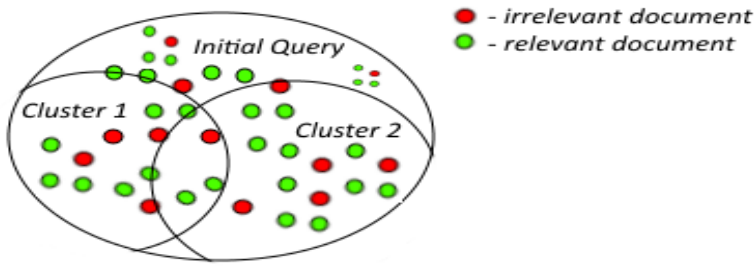


Figure 5.13: Initial and subtopics' coverage.

Most of the documents in the clusters are expected to be relevant to the search request for a successful retrieval because the clusters are subsets of the search results. It is known that the relevant documents have similar statistics, as they have been retrieved for same search request. Hence, there are only two ways the QE algorithm to give different terms for different clusters. The first way is the cluster to cover a great number of irrelevant documents and the second is to simply extract more terms from the DFRBagExpansionTerms DS until all terms specific to to the current search request topic are selected. The first approach is not feasible due to the reasoning above (clusters contain mainly relevant documents for a successful search), therefore the second approach was chosen and the number of terms selected through QE was changed to 5 in an attempt different terms for the different clusters to be extracted. It was detected that using more terms was not effective because such an approach leads to introduction of too much noise and deviation from the initial information need. For this reason an empirical analysis was conducted identifying a necessity for an additional restriction - the optimal depth of a *recursive exploration* was limited to the 5th level after which a further exploration was not allowed. The screenshot 5.14 below shows an example of a recursive exploratory search session.

The screenshot above shows the first 3 results associated with an exploratory search session with an original query term "economy" followed by an exploration of the category "fuel", which is one of automatically identified clusters of the result set retrieved from the search for "economy". The exploratory search result is exploratory searched on its own by the selection of the new "caf" sub-category of the "fuel" category. After this the "committee" and "fleet" categories are explored in a similar fashion. A more thorough understanding of the functionality can be acquired by use of the product.As it can be seen the depth of the search if 5, so a message "Max Depth Reached Click for Search Mode" has appeared to inform the user further exploration is not allowed.

An alternative known as "historical exploratory search" is possible, where the user can click on any of the previously explored categories to go back in the exploratory search history to take a different *exploratory path*.

Figure 5.14: Exploratory Search Session for query “economy“.

5.3.5 List of Relevant Documents

The final functionality provided from the Exploratory Search Engine is the list of relevant documents. Given a field the searcher would like to learn more about, he or she will most like pass through a series of search requests and exploratory search sessions to get a better understanding of the topic. In order this process to be facilitated, the user is able to mark documents as relevant and store these documents in a list, as shown on figure 5.14. The list of relevant documents is cached on the client site. When a result set is initially retrieved from the server all documents are irrelevant. On execution of the HTTP GET callback their state is evaluated using the cached set of document ids including all documents selected as relevant. If there are matching document ids, their state is set to relevant. This approach has several advantages. First, it is very efficient as there is no step requiring more than O(n) time complexity. Second and third, it reduces the communication overhead and load on the server side not transmitting data which only the frond-end depends on. Finally, it addresses the privacy of the user. Figure 5.15 below visually presents the design of the feature.

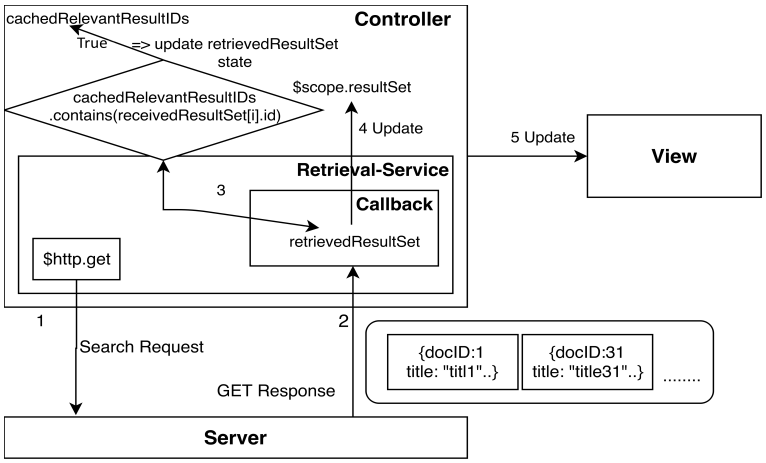


Figure 5.15: Design of the List of relevant documents feature.

5.4 Summary

This chapter made the reader aware of some the challenges experienced on the server and client sides, as it analyzed the implementation of both. The user interface was also presented. Most of the challenges were related to the Hierarchical clustering, Concept Hierarchy algorithm and the Exploratory Search. On the presentation tier, the final user interface was presented. It was discussed how it can be used by an end user and how some of the more demanding components were designed. The Next chapter describes the evaluation of the product.

Chapter 6

Evaluation

After the implementation was completed the project was evaluated using a variety of evaluation approaches. First an offline evaluation was performed to measure the effectiveness and diversification of the search results. After this more quantitative and qualitative feedback was gather analyzing users‘ log data and using a Guided Think-Aloud in a combination with three questionnaires. The evaluation answers the following research questions: *Which of the three sub-query generating algorithms generates queries resulting in a higher search result effectiveness and better diversification of the search results according to an objective offline evaluation and the users‘ subjective opinion?*, *How does xQuAD using the most effective sub-query generating algorithm (identified from answering the previous question) performs when compared to Terrier without xQuAD and xQuAD using ‘gold standard‘ sub-queries?*, *Which exploratory search algorithm is found as most effective by the users?*.

6.1 Offline Evaluation

Offline evaluation provides objective measurement of the system's effectiveness in the form of different evaluation measures such as recall, precision, MAP, etc. . The current offline evaluation results are achieved using a Financial Times corpus with a corresponding list of topic and qrels files [76]. The most effective mechanism for generation of sub-queries for xQuAD among the provided algorithms is selected analyzing the performance of the system against these files. Once the most effective mechanism is identified, it is tested how well it diversifies the search results. To identify the most effective sub-query generating algorithm, first the flat algorithm had to be configured for the current corpus. To achieve this its performance was evaluated varying the number of clusters used for generation of sub-queries.

6.1.1 Tuning of the Flat Clustering Sub-query Generating Algorithm

The table below presents the performance of the algorithm for 2, 3 and 4 clusters. MAP, p@5, p@10 and recip_rank measures are used to determine the optimal configuration of the algorithm. [46].

Number of Clusters \ Measurement	MAP	p@5	p@10	recip_rank
2 clusters	0.3575	0.5600	0.5300	0.7585
3 clusters	0.3583	0.5630	0.5450	0.7785
4 clusters	0.3563	0.5611	0.5310	0.7631

Figure 6.1: Performance analysis of the Flat Clustering Sub-query Generating algorithm.

As presented on figure 6.2 the mean average precision and recip_rank are highest when 3 clusters are used. The latter indicates that the first relevant document is ranked higher in the ranking for the second configuration. On average the ratio between relevant retrieved documents and retrieved documents is again higher for this configuration according to the MAP.

6.1.2 Comparison of Sub-query Generating Algorithms

The Hierarchical Clustering and Concept Hierarchy Sub-query Generating algorithms both autoconfigure based on the collection as discussed in the Implementation chapter for which reason their configuration is not manually tuned. This section of the evaluation compares the results of the three sub-query generating algorithms relying on the same evaluation measures for consistency with the tuning above. The table below summarizes the results:

Sub-query Generating Algorithm \ Measures	MAP	p@5	p@10	recip_rank
Hierarchical Clustering	0.3572	0.5498	0.5401	0.7781
Flat Clustering	0.3583	0.5630	0.5450	0.7785
Concept Hierarchy	0.3561	0.5560	0.5350	0.7775

Figure 6.2: Comparison of Sub-query Generating Algorithms.

It can be seen from the results above that the Flat Clustering has a higher MAP and better overall performance as a result of this than the other two algorithms. This can be justified by the fact that the algorithm is tuned for the current experiment. On the other hand, if hierarchical clustering is used, it is possible the sub-query generated by the 'last cluster' to negatively affect the search results (because its label will most likely not be a representative subtopic of the search request). The 'last cluster' includes all documents which are not assigned clusters in case there are no at least 3 clusters each with at least 10% of all search results (as discussed in section 5.2.2). The Concept Hierarchy chooses the number of subtopics based on the number of query terms (with a set upper boundary which is almost always reached - as explained in the Implementation chapter). As a result of this, it is possible the Concept Hierarchy Sub-query generating algorithm to struggle to find the correct number of subtopics leading to worse results then the Flat Clustering approach. The Flat and Hierarchical algorithms, on the other hand, both perform better against the Concept Hierarchy. This can be explained by the fact that the Concept Hierarchy limits the possible subtopics to the direct children of the query terms in the Concept Hierarchy even though it is not always the case that the terms which represent subtopics of individual query terms simultaneously represent subtopics of the query itself - the whole query phrase. As a result the Concept Hierarchy algorithm assumes independence between the query terms, which is clearly an incorrect assumption most of the times. In addition to this, the Flat and Hierarchical sub-query generating classes rely on the cluster labeling algorithm which performs absorption to avoid deviating the label towards a particular subset of the cluster's terms, which is not implemented for the Concept Hierarchy. Figure 6.3 below present a recall-precision graph to provide a better granularity of the observed results.

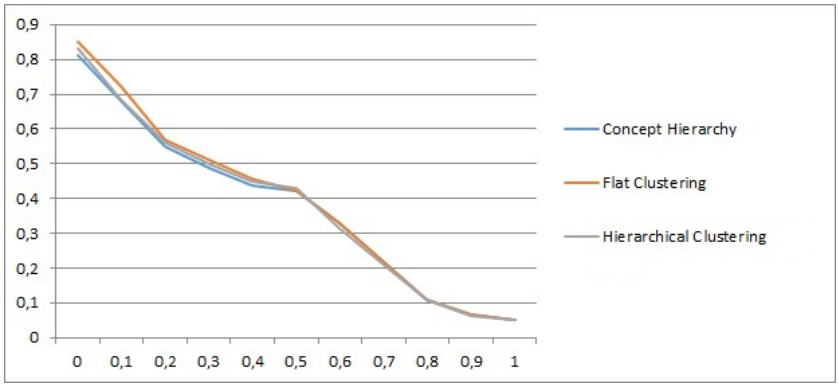


Figure 6.3: Recall-Precision graph of the three Sub-query Generating algorithms.

Figure 6.3 illustrates a similar result. An interesting observation here is that as the recall increases the precision of the algorithms gets closer. However, this should not be a concern because the users will most likely concentrate at the top of the ranking where the Flat clustering algorithm dominates.

6.1.3 The Baseline, Flat Clustering Sub-query Generating and the Gold Standard

After the most effective sub-query generating algorithm among the three presented is identified, its performance is compared with this of the baseline algorithm - Terrier without xQuAD and the Gold Standard. The Gold Standard is Terrier +xQuAD generating sub-queries statically from a file. In other words, the Gold Standard relies on subtopics of the tested search requests chosen from professional relevance assessors from the present test collection. It is assumed that these subtopics used as sub-queries provide the best achievable performance of xQuAD. Once again a table summarizing the results is provided below:

System Configuration \ Measures	MAP	p@5	p@10	recip_rank
Terrier without xQuAD	0.3568	0.5540	0.5280	0.7683
Flat Clustering	0,3686	0,5858	0.5330	0,7855
Official Subtopics	0.4076	0.6100	0.5450	0.8105

Figure 6.4: The Baseline, Flat Clustering Sub-query Generating and the Gold Standard.

As it can be seen from table 6.4 above the Flat Clustering has a slightly better MAP representing its overall performance against the baseline. Expectedly the Official Subtopics configuration presents the optimal results. Interesting the initial precision at p@5 and p@10 is considerably improved with the Flat Clustering when compared to the baseline. This is an important benefit of the presented algorithm as this is the part of the result set where most of the users concentrate. The recip_rank is again slightly better.

Appendix I contains a graph indicating the recall-precision relationship for the three configurations. An interpretation of the graph reveals that the initial performance of the three algorithms at the very top of the ranking is quite similar. As the ranking decreases the precision of the Flat Clustering sub-query generating algorithm changes from precision closely following the baseline to precision getting closer to the Gold Standard even though the Gold standard is reached quite down in the ranking. This can be justified by the fact that at the very top of the ranking all of the three configurations retrieve the documents with the highest statistics for the query terms. As we go down the ranking the statistics of documents satisfying the query terms decreases by definition allowing the impact of the sub-query terms to grow retrieving a more diversified set of results which are not retrieved by the baseline. Hence, the xQuAD gets closer to the Gold Standard down in the ranking. The Baseline and the Gold Standard are almost a constant distance away from each other which is again expected as the Baseline never manages to retrieve the subset of relevant results provided from the Gold Standard due to the lack of diversification capabilities.

6.1.4 Metrics Awarding Diversity

Although precision and recall are extremely popular and widely used IR metrics, they provide only a general idea of the system‘s performance. More specifically they indicate how many relevant results are retrieved and how high in the ranking but the two metrics are too simplistic to inform us if the result set is actually diversified or not. The results from the previous sections show us that the overall performance is improved with the Flat Clustering while this section looks into its diversification capabilities. Table 6.5 below represents a quick check acting as a proof that the Flat Clustering sub-query generating algorithm is the most appropriate diversification mechanism among the three compared algorithms.

Sub-query Generating Algorithm \ Measures	a-NDCG@5	a-NDCG@10	a-NDCG@20	MAP-IA
Hierarchical Clustering	0.422723	0.451932	0.465813	0.149213
Flat Clustering	0.428974	0.457206	0.469884	0.151191
Concept Hierarchy	0.418508	0.450364	0.459459	0.148798

Figure 6.5: Diversification capabilities of the three Sub-query Generating algorithms.

The a-NDCG and MAP-IA are used as they are metrics awarding diversity allowing the diversifying features of the algorithms to be explored [22]. These metrics are calculated using a tool known as ndeval, provided by the TREC 2014 Web Track [78].Once we know the Flat Clustering performs better than the other two algorithms, we can compare it with the baseline (Terrier without xQuAD) and the *Gold Standard* (Official Subtopics).

Sub-query Generating Algorithm \ Measures	a-NDCG@5	a-NDCG@10	a-NDCG@20	MAP-IA
Terrier without xQuAD	0.3918934	0.410576	0.430187	0.135127
Flat Clustering	0.418974	0.433206	0.449884	0.141191
Official Subtopics	0.448523	0.469146	0.470121	0.160053

Figure 6.6: Diversification capabilities of the Baseline, Flat Clustering and the Gold Standard.

Again it can be seen that the measurements for the Flat Clustering sub-query generating algorithm are better than the baseline even though they are not particularly close to the Gold Standard. This demonstrates that the Flat Clustering not only improves the overall precision but it also leads to a more diversified result set. The graph below shows the per-query MAP-IA performance for the three configurations allowing a closer look at their performance.

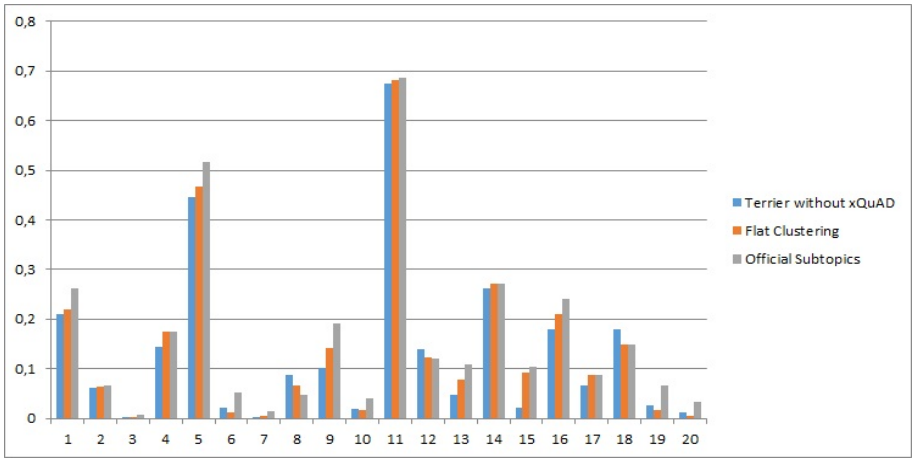


Figure 6.7: Per-query MAP-IA performance.

It can be seen that the MAP-IA is between 0.1 and 0.7 for most of the queries except queries 2, 3, 6, 7, 8, 10, 17, 19, 20, for which an interesting pattern is observed. For example, the topic of query 3 is *International Art Crime*, it has 9 subtopics and extremely low MAP-IA for all system configurations. A a straightforward explanation of the observation can be found in the fact that baseline Terrier struggles to retrieve relevant results for the initial query. The diversification is performed as a second step based on the initially retrieved results. Hence, if the initial results are poor, the clusters will also not be representative and the automatically generated sub-queries will further deviate from the correct sub-topics. This is a crucial observation based on the dependence between the two libraries - the more effective the initial ranking of Terrier is the more accurately the xQuAD library performs. However, if Terrier's initial ranking is ineffective, the Flat Clustering sub-query generation algorithm can lead to even more ineffective final ranking, as observed on the column bar chart above.

6.1.5 Evaluation Results

In summary, the offline evaluation led to the following discoveries. First, it identified that the Flat Clustering Sub-query Generating Algorithm performs optimally for the current corpus when it uses 3 clusters. The Flat Clustering sub-query generating algorithm was also found to have higher effectiveness and better diversification capabilities than the Hierarchical and Concept Hierarchy sub-query generating algorithms. Finally, it was analysed that it performed better than the baseline and worse than the "gold standard". Next section examines what the users' opinion of the system is.

6.2 User Evaluation

Once the offline evaluation was completed, an online/user evaluation was conducted to identify if there was a niche market for the current application and how effective and usable the current application was. It consisted of one pre-evaluation questionnaire, a guided think-aloud plus another questionnaire to preserve the overall feedback and comments from the think-aloud, and a final post-evaluation questionnaire to capture the user's input related to the performance, usability and UI design of the system.

6.2.1 Pilot Study

First a pilot study was conducted to evaluate the feasibility and time required for a single experiment, and to discover any potential issues with the system. It consisted of 6 students from the School of Computing Science in the University of Glasgow. The first observed issue was related to the size of the categories formed from the search results, as some of the categories had only 1 or 2 documents. This issue was resolved, as its implementation is the topic of section 5.2.2 of the Implementation chapter. There was a great number of comments related to the category labels as well. Some subjects discussed that some of the category labels are the same for different categories, other labels are too general or too specific or some of the labels were only related to a small subset of the category's documents instead of being relevant to all documents in the category. All of this concerns were addressed in section 5.2.4 of the Implementation chapter. It was also indicated that a single experiment takes a considerable amount of time - about an hour or even more in some cases. This was not desirable, as it could lead to significant fatigue among the participants (they had to simultaneously work on the tasks and think-aloud during the period) and to biased input as a result. To deal with this challenge the number of search tasks was reduced from three to two, and a logging functionality was added to automate part of the process. The search tasks were also clarified and associated with their corresponding subtopics. The users had to use this information to associate the documents from the result set to their corresponding subtopics. This data was utilized to evaluate the diversification capabilities of the different system configurations. As suggested by the supervisor the subjects were given a topic to search for instead of the exact query, which was the initial approach, to increase the external validity [8] of the experiment. Finally, some users commented that the performance of the system was slower than expected. As a result of this observation a client-side cache was added to store the top 100 results. In this way, the initial load time was slightly longer but after this initial period the performance of the system was instantaneous.

6.2.2 Pre-evaluation Questionnaire

For the actual user study 10 students from the School Of Computing Science at the University of Glasgow were selected. All of the subjects filled in a pre-evaluation questionnaire prior to using the system. The purpose of the questionnaire was to be found out if there was a niche market for the product and to be determined if the

tested subjects were representatives of the targeted group of users. The first two questions were used to filter out subjects who were not representatives of the targeted group, as their statistics is shown on figure 6.15 below:

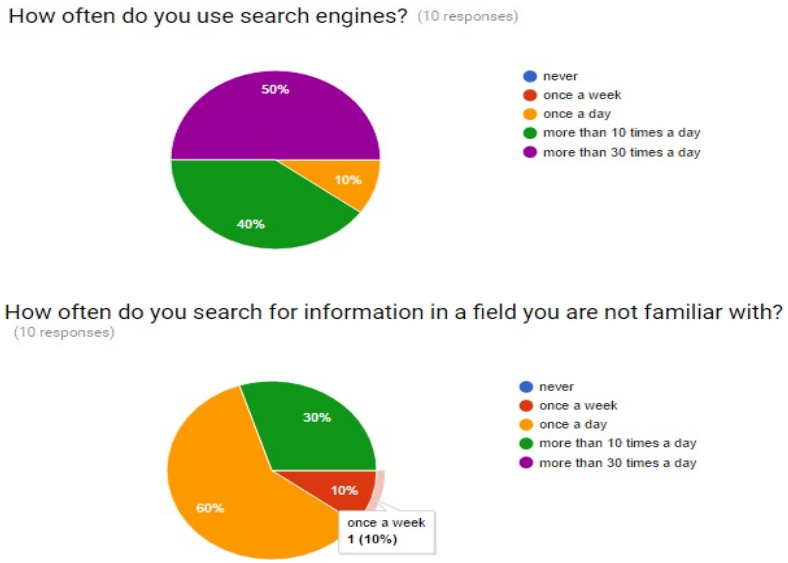


Figure 6.8: Filtering of the Targeted Group.

As it can be seen the majority (90%) of the tested group uses search engines regularly, as many of them also frequently search for information in a field they are not familiar with. These two observations can be used as a proof that a representative group of participants is selected. The table below summarizes the results from the group of questions examining if there is a niche market for the current product:

Question \ % of the time	0-25%	25-50%	50-75%	75-100%
How often is the retrieved information relevant to you?	0%	30%	60%	10%
How often do you have difficulties formulating your query when searching for information in a field you are not familiar with?	10%	50%	30%	10%

Figure 6.9: Niche Market - Group of Questions.

The first question shows that only 10% of the users are fully satisfied with the search results when they search in a field they are not familiar with (first question in the table is related to the last pie chart above), as they also frequently struggle to formulate their query when searching in a field they are not familiar with. The purpose of these two questions is to identify a pattern. They act as a proof that there is a direct connection between the inability to formulate a query when searching in a field the user is not familiar with and his/her satisfaction with the search results. The purpose of the current application is to close exactly this gap via its exploratory search feature and categorization of the result set. This problem is well-know in the IR field as the ASK Hypothesis [4]. The final question related to the Niche Market group of questions tries to identify what features are currently missing which can potentially improve the effectiveness of a search system when searching in a field the user is not familiar at. Several alternative features are presented, as the most useful ones according to the users are a categorization (40% of the tested subjects want it) , graph with categories (30%), and graphical bars representing document scores (20%). The desires of the users closely follow the implemented functionalities of the presented system, except the *graph with categories* functionality which is not implemented. A justification of this decision is the topic of section 2.1.2 of the Relevant Research chapter.

6.2.3 Guided Think-Aloud Study

Experimental Design

After the pre-evaluation questionnaire the participants were asked to play with the system and experiment with its functionality while they are trying to achieve two search tasks. The search tasks were taken from the same topics file used for the offline evaluation (topics file is a file containing search tasks) for consistency. A search task includes a topic and a description. The topic is what the user should search for and the description explains the user persona the subject should pretend to be and particular information the subject should search for. In addition to the topics file, the subtopics file (also used for the offline evaluation) was also utilized. The purpose of a subtopics file is to provide subtopics for each search task from the topics file. The participants were provided with these subtopic lists associated to the two search tasks and were asked to find documents relevant to as many of these subtopics as possible. A successful completion of a task means that the subject has managed to find documents related to a larger number of the provided subtopics. This was used to measure how well the different system configurations diversify the result set.The two search tasks are summarized in the table below.

	Topic	Description	Subtopics
Task 1	commercial cyanide uses	What are the industrial or commercial uses of cyanide or its derivatives? While you are searching for information try to identify as many uses as you can from the following subtopics list:	- chemical weapons - synthetic fibres - suicide agent - in fishing - metal coating/galvanizing etc.
Task 2	Wildlife Extinction	The spotted owl episode in America highlighted U.S. efforts to prevent the extinction of wildlife species. What is not well known is the effort of other countries to prevent the demise of species native to their countries. What other countries have begun efforts to prevent such declines? Try to find information for as many of the following countries as you can:	- Finland - saima ringed seal - Brazil - golden lion - Kenya - elephants - Columbia - Andean condor - South Africa - quagga, white rhino etc.

Figure 6.10: Search Tasks.

As the lists with subtopics are relatively long, only part of the lists are presented in the table 6.10 to keep it concise. The first task consists of 7 subtopics, as it is an example of an *exploratory diversification task* - the subject must understand the subtopics and the retrieved documents well to be able to make a judgment. The second search tasks is an instance of a *factual diversification task* - the subject must simply identify facts, look for key phrases or terms in the retrieved documents. In our example, he/she should simply find information about something relate to wildlife extinction for as many of the mentioned countries and species as he/she can. A simple skimming through the results is sufficient. This task consists of 25 subtopics. The two tasks are deliberately selected in this way to evaluate how the different sub-query generating algorithms cope with different exploratory search tasks and a varying number of subtopics.

The “User Personal Evaluation of Search Engines “ and “Web search engines evaluation based on features and end-user experience“ papers describe user evaluations of search engines and were used as an inspiration for the current user evaluation [20] [23]. They both rely on the use of Balaced Latin Squares to reduce possible biases caused by fatigue, learning effects and practice effects. The 10 subjects were split into two groups - group A and group B. Group A first performs the first search task and group B performs the second task first. The guided think-aloud consists of four search sessions for each of which both search tasks are performed. There is 1 search session for the baseline - Terrier without xQuAD. The other three search sessions are Terrier+xQuAD+Flat Clustering Sub-Query generating, +Hierarchical Clustering Sub-Query generating , and +Concept Hierarchy Sub-Query generating respectively. Group A starts with search session group 1 (search session group 1 includes 1st and 2nd search session) first, while group B begins with search session group 2 (including the 3th and 4th search sessions) first. The Latin Squares below give a pictorial summary of this evaluation design:

Group \ Sequence	1	2	3	4	5	6	7	8
Group A	Topic1+ Conf. 1	Topic1+ Conf. 2	Topic1+ Conf. 3	Topic1+ Conf. 4	Topic2+ Conf. 1	Topic2+ Conf. 2	Topic2+ Conf. 3	Topic2+ Conf. 4
Group B	Topic2+ Conf. 2	Topic2+ Conf. 3	Topic2+ Conf. 1	Topic2+ Conf. 2	Topic1+ Conf. 2	Topic1+ Conf. 3	Topic1+ Conf. 1	Topic1+ Conf. 2

Legend:

Topic No. \ Topic Name	*Topic Name*
Topic 1	*Commercial Cyanide Uses*
Topic 2	*Wildlife Extinction*

System Configuring \ Configuration Description	*Configuration Description*
System Configuration 1	*Terrier*
System Configuration 2	*+ xQuAD* *+ Flat Clustering Sub-Query Generating*
System Configuration 3	*+ xQuAD* *+ Hierarchical Clustering Sub-Query Generating*
System Configuration 4	*+ xQuAD* *+ Concept Hierarchy Sub-Query Generating*

Figure 6.11: User Study Design.

Guided Think-Aloud Study - Qualitative Results

The results of the evaluations of each of the system configurations above are captured by the evaluation's interviewer in a Think-Aloud questionnaire which consist of the following two statements for each of the system configurations: *"The search returned results relevant to the query."* , "I managed to accomplish the assigned task.". Each is graded from 1 to 5, where 1 is strongly disagree and 5 is strongly agree. In an attempt to keep this chapter as concise as possible the scores for the two statements for the four system configurations are presented in a tabular form below. The last column is the average score per statement for a given system configuration.

Question \ % of Subjects per Score *-> Strongly Disagree (1) ... Strongly Agree (5)*	1	2	3	4	5	Mean Score
Baseline – Terrier Without xQuAD						
The search returned results relevant to the query. (10 responses)	0%	0%	30%	50%	20%	3.9
I managed to accomplish the assigned task. (10 responses)	0%	0%	40%	50%	10%	3.7
Terrier +xQuAD +Flat Clustering Subquery Generating						
The search returned results relevant to the query. (10 responses)	0%	0%	10%	40%	50%	4.4
I managed to accomplish the assigned task. (10 responses)	0%	0%	0%	60%	40%	4.4
Terrier +xQuAD +Hierarchical Clustering Subquery Generating						
The search returned results relevant to the query. (10 responses)	0%	0%	20%	50%	30%	4.1
I managed to accomplish the assigned task. (10 responses)	0%	10%	30%	40%	20%	3.7
Terrier +xQuAD +Concept Hierarchy Subquery Generating						
The search returned results relevant to the query. (10 responses)	0%	0%	20%	50%	30%	4.1
I managed to accomplish the assigned task. (10 responses)	0%	10%	20%	30%	40%	4.0

Figure 6.12: Comparison of the Four System Configurations.

As it can be seen from table 6.12 the scores for the first statement are similar for almost all of the system configurations. Only the Flat Clustering has a slightly higher value. The goal of the first question is to evaluate the general effectiveness of the system. This can be related to the first part of the Offline Evaluation where the MAP, p@5, p@10 etc. metrics demonstrating the general effectiveness were measured. It is also used to determine if any of the algorithms negatively impact this general effectiveness. The scores for all of the three configurations using xQuAD are higher or equal to the baseline for the first question, so we can conclude that the addition of xQuAD does not lead to a decrease of the general effectiveness. It can even slightly improve it

in some cases. The accomplishment of the assigned task is related to identifying as many as possible documents related to the subtopics presented for each search task. For this reason, the second question can be interpreted as the factor demonstrating the diversification capabilities of each of the system configurations. Similarly to the MAP-IA and a-NDCG from the Offline Evaluation, this question awards diversity. It is clear that it is the highest for the Flat Clustering in agreement with the results of the Offline Evaluation. Surprisingly the mean score for the Concept Hierarchy is identical to this of the baseline. The Hierarchical Clustering is somewhere in the middle. The mean scores for all of the four system configurations for the two questions are quite close to each other. This does not come as a surprise. The Offline Evaluation also indicated close results. The human factor should also be considered. Even though the offline evaluation is capable of considering all documents in the result set, the subjects only looked at the documents from the first 1-2 pages, which is most likely the reason why the results are so close to each other.

Categorization of the Results

The categorization functionality was also qualitatively assessed. Similarly to the previous questions score of 1 means *Strongly Disagree* and 5 is for *Strongly Agree*.

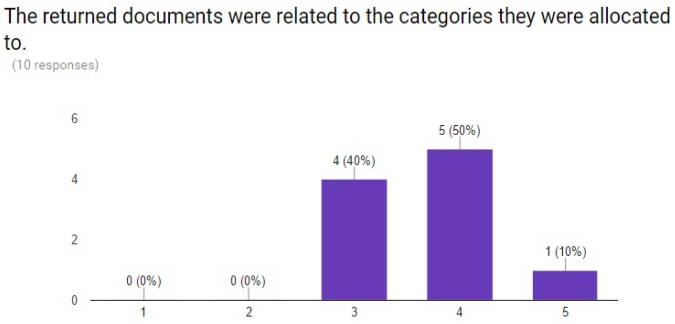


Figure 6.13: General effectiveness of the Categorization functionality.

The first question‘s purpose (Figure 6.13) is to get an overall feedback of the subjects‘ satisfaction with the functionality. With an average score of 3.7 it can be concluded that the features is behaving correctly. The second question, Figure 6.14, examines if the categories are independent or whether their document sets intersect. As it can be seen most of the participants believe the sets are independent even though there is a small number of subjects which cannot decide. Some of the comments related to this part of the experiment state that some of the category labels are too general. This is the reason why the score for this question is relatively low. The issue is partly address in section 5.2.4. . Two alternative solutions are to use a larger stop words list than the one provided from Terrier or to allow the users to select the category‘s label as too general, at which moment the current label can be added to the existing stop words list and a new label can be generated.

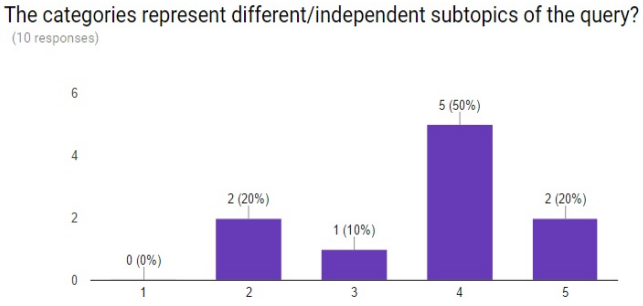


Figure 6.14: Independence of the search results‘ categories.

Effectiveness of the Exploratory Search Feature

The last section of the Think-Aloud study captured subjects‘ opinion on the Exploratory Search feature. Participants were asked to use it for the two search tasks in an attempt to identify more relevant documents. The two exploratory search configurations were used and their performances compared. Group A first used the exploratory search based on the category‘s name while Group B worked with the exploratory search based on the content of the category‘s documents first. Both groups found the exploratory search based on the category‘s name much more effective with an average score of 4.4 compared to 2.5 for the exploratory search based on the content of the category‘s documents. A reoccurring comment was that the results get more general instead of more specific and sometimes even unrelated. This phenomenon is explained in detail in section 5.3.4 of the Implementation chapter and can also be related to the fact that Terrier makes query terms appear bold in the document abstracts. When this type of an exploratory search is used a query expansion is performed based on the content of the cluster, which is the reason why some terms which the user has not specified in his/her query appear bold in the results. This confused the subjects during the experiment and this is the reason why some of them thought that the results were unrelated.

6.2.4 Post-evaluation Questionnaire

After the experiment all subjects were asked to fill in a post-evaluation Questionnaire. The questionnaire was designed using a standardized System Usability Scale (SUS) create by John Brooke in 1986, which represents “a quick and dirty usability scale“[6]. It consists of four straightforward questions whose purpose is to figure out how useful the current product is. It was expected that the user would be tired after the Think-aloud evaluation. For this reason, the length and level of difficulty of the questions in the post-evaluation questionnaire were both minimized as much as possible. The two questions in the table below summarize subjects‘ point of view on the usability of the system.

Question \ % of Subjects per Score *-> Strongly Disagree (1) ... Strongly Agree (5)*	1	2	3	4	5	Mean
I think that I would like to use this system frequently. (10 responses)	0%	0%	20%	40%	40%	4.2
I thought the system was easy to use. (10 responses)	0%	0%	10%	50%	40%	3.9

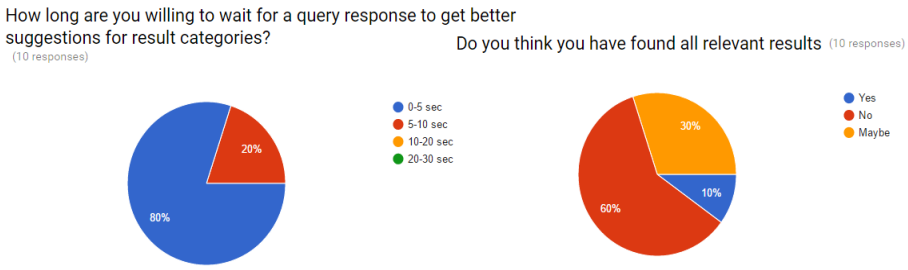


Figure 6.15: System Usability.

As it can be seen from the table above the majority of the participants are positive about the usability of the system as about 80% of them say that they will use it again. The third question tries to capture how long the users are willing to wait for a query response. The objective of adding this question is that the addition of xQuAD and the sub-query generating algorithms slowed down the system with approximately 3-6 seconds per request. It was desired to be evaluated if this efficiency drop is tolerated by the user. Unfortunately, as it can be seen from the results, this represents a difficulty for the participants of the experiment - an issue which should be addressed in the next version of the current product. The majority of the people responded negatively to the last question as expected. The idea here is once again to remind the users that searching a field they are not familiar at is challenging and a tool such as the Exploratory Search Engine is essential to help them find the information they are looking for.

6.2.5 Evaluation results

In summary, the user evaluation identified the following main results. The Flat Clustering sub-query generating algorithm was again found as more effective than the other two algorithms similarly to the offline evaluation even though the quantitative results were extremely close, which can be explained by the fact that the users looked only at the first page with search results. The categorization of the search results functionality was found useful with an average score of 3.7 from 5.00 max. The exploratory search relying on the category‘s name was expectedly regarded as more effective by the users, as predicted in section 5.3.4 than the exploratory search based on the content of the category‘s documents. Finally, the evaluation of answers from the post-evaluation questionnaire revealed that 84% of the experiment‘s participants would like to use the system frequently and 78% of them found it easy to use. Next section analyses the logs gathered from the user evaluation.

6.3 Log Analysis

In addition to the Guided Think-Aloud and questionnaires data about the users‘ experience with the system was gathered via a logging module recording users‘ sessions while they are trying to complete the search tasks. This was achieved using an additional pop-up window for the evaluation version of the application which appears when a user selects a document as relevant. This pop-up has the two lists with subtopics for the two search tasks represented as radio buttons. While the users were playing with the system they were asked to select documents satisfying the particular search task as relevant by clicking on a button next to each document entry and to choose the subtopic they believe the document is relevant to from the pop-up. The button the users select is the same button used to add a document to the List of Relevant Documents explained in section 5.3.5. The log data was organized and stored in 8 separate log files - 4 per each search task. Each of these 4 log files preserves log data for one of the four system configurations - baseline, baseline+xQuAD+Flat Clustering Sub-query Generating algorithm, baseline+xQuAD+Hierarchical Clustering Sub-query Generating algorithm and the baseline+xQuAD+Concept Hierarchy Sub-query Generating algorithm. A separate python script, available in appendix F.1 was written to automatically summarize and analyze the logs. The script inspected the diversification capabilities of the different system configurations, the accuracy the Category Labeling algorithm and the general effectiveness of the four configurations. These estimates were gathered as follows:

Evaluated Feature	***Utilized Log Data Analysis***
Diversification Capabilities	Compare the *number* of *categories* the selected relevant documents correspond to
General Effectiveness	Compare the *number* of selected relevant docs. among the 4 system configurations
Accuracy of the Category Labels	Compare the *relationship* between the *category label* and the *selected subtopic*

Figure 6.16: Utilized log data analyses.

The table above represents a summary of the used log data analyses. How they can actually be used to evaluated the system can be best presented through an example. The figure below shows what a log line contains to provide more context to the presented evaluation technique.

queryID	docID	category label	subtopic selected by user	subtopicID	rank
366i	FT932-5802	gold	ore processing: leaching out...	0	0

.....

Figure 6.17: Log record.

6.3.1 Diversification and General Effectiveness

The table below represents a summarized view of the log data for the first search task related to the first two log data analyses from table 6.16. More precisely it shows the distribution of identified relevant documents by the users among the subtopics and the total number of found relevant documents from the users. The cells in the table contain the number of relevant documents for the specific subtopic. The values come from the 10 participants in the evaluation. For example, value 15 for column *Ore Processing* and row baseline stores the number of relevant documents found by all 10 participants using the baseline configuration for subtopic *Ore Processing* for the first query. If the same document is found as relevant by several subjects it will appear as many times as the number of subjects. If a cell value is divided by 10, you will get the average number of relevant document identified by a subject. This value may appear low. There are two reasons for this. The first is that the subjects examined only the first 10 results. Second reason is that they did not manage to relate all of the top 10 results to particular subtopics from the subtopics list.

System Configuration	Ore Processing	Suicide Agent	Metal Coating	In Fishing	Chemical Weapons	Commodity to be Recovered	All Relevant Documents
Baseline	15	0	1	2	11	0	29
+xQuAD+Flat Clustering	12	6	8	7	11	0	44
+xQuAD+Hierarchical Clustering	10	6	9	4	3	3	35
+xQuAD+Concept Hierarchy	10	0	6	8	9	3	36

Figure 6.18: Distribution of relevant documents among the subtopics per each system configuration - 1st search task.

As it can be seen the subjects have mainly found relevant documents from two subtopics for the baseline - *Ore Processing* and *Chemical Weapons*. Only 3 relevant documents are found from the other subtopics from the 10 subjects. This shows a relatively poor performance of the baseline for the *diversification capabilities* feature (in agreement with the metrics awarding diversity from the Offline Evaluation section) . The total number of relevant documents found by the subjects for the configuration is 29 - this shows the relatively lower *general effectiveness* of the baseline compared to the other configurations (this feature can be related to the recall from the Offline Evaluation section with which this observation is consistent). This measure is higher for the 3 xQuAD configurations as it is the highest for the Flat Clustering Sub-query generating algorithm which agrees with the observations made from the Offline and User Evaluations. The distribution of relevant documents among the subtopics is again higher for the 3 xQuAD configurations. Unexpectedly the +xQuAD+Hierarchical Clustering Sub-query generating algorithm has managed to retrieve relevant documents from all subtopics according to the users log while the +xQuAD+Flat Clustering has not. This is an example of inconsistency of the users‘ log results with the results of the metrics awarding diversity from the Offline Evaluation and User Evaluation according to which the +xQuAD+Flat Clustering has better *diversification capabilities*. Even though the +xQuAD+Flat Clustering does not provide the optimal diversification here, it still has the highest *general effectiveness* with total number of relevant results of 44.

System Configuration	1	2	3	4	5	6	7	8	9	10	11	12	13	≥ 3 RPS	> 4 RP	All Relevant Documents
Baseline	0	3	0	0	0	7	9	0	0	8	0	1	2	4	3	30
+xQuAD+Flat Clustering	6	2	2	6	5	10	11	1	1	10	3	0	0	7	5	57
+xQuAD+Hierarchical Clustering	3	12	4	4	0	9	10	0	0	8	5	0	0	8	5	55
+xQuAD+Concept Hierarchy	3	7	4	8	5	9	14	3	0	10	3	0	0	10	6	66

Figure 6.19: Distribution of relevant documents among the subtopics per each system configuration - 2nd search task.

The log analysis of the first task revealed that the addition of xQuAD can assist the users with their task to identify documents related to a diverse set of subtopics. The first search task required a good understanding of the subtopics and consisted of a small number of subtopics. The goal was to find different uses of cyanide.

Compared to it the second search task, the results for which are shown on table 6.19, is much more factual - subjects should just skim the result set for particular phrases contained in the document abstracts. There are much more subtopics for this task for which reason there are presented with IDs instead of with their actual titles. The greater or equal to 3 RPS (Relevant Per Subtopic) column shows the number of subtopics with more than 3 relevant documents found by the participants in the experiment. Similarly to the first search task, it can be seen that the *general effectiveness* is worse for the baseline. However, here the +xQuAD algorithms have much better *general effectiveness* than for the first task - the total number of relevant documents is almost twice more here for them. This can be explain by the larger number of subtopics, which means that the algorithms perform better with more subtopics, and the fact that the 2nd search task is easier for the users. The *general effectiveness* is the highest for the Concept Hierarchy. If we consider the greater or equal to 3 RPS, the +xQuAD+Concept Hierarchy shows again the highest *diversification capabilities* - it has managed to find 10 subtopics with more than 2 relevant documents per each (total number found by the 10 participants in the experiment). However, an interesting fact is that if we consider the greater or equal to 4 RPS, it can be seen that the 3 +xQuAD algorithms have values starting to converge to each other. This shows that even though +xQuAD+Concept Hierarchy shows better *diversification capabilities* for the 2nd search task, it does not equally distribute the relevant documents among the subtopics. It is simply capable to identify a few more documents for the unpopular subtopics from the other 3 configurations.

6.3.2 Accuracy of the Category Labels

After the *diversification capabilities* and the *general effectiveness* of the four system configurations were analyzed the *accuracy of the category labels* was investigated relying on the log entries. Each log record has a *category label* entry and a *subtopic selected by the user* entry as shown on table 6.17. This mapping was used to identify if there is a relationship between the automatically generated category labels and the official subtopics. To achieve this another python script, available in appendix F.2, was written counting how many times a particular *category label-subtopic* pair occurs in the log files (for all system configurations for each search task) filtering out pairs with a count/weight less than 6 for the first search task and 9 for the second search task (there is a difference because there is a higher overall number of relevant documents for the second search task as shown on figure 6.19). The script outputs the filtered pairs and their weights in a .gv format [60]. This format is readable by the dot tool provided by Graphviz, which can generate graphs from .gv formatted files [59] [58]. The graphs for the two search task are available as part of appendix I.2.

As it can be seen from Figure I.2 the automatically generated category “gold“ is related to the “ore processing:leaching out metals in mining“ and the “metal coating or galvanizing“ official subtopics as the number of relevant documents selected by the participants in the experiment are 47 and 17 respectively. These two pairs represent a logical relationship between the automatically generated category labels and the official subtopics. The categories “chemicals“ and “industries“ also are correctly associated to the “chemical weapons“ official subtopic. On the other hand, the category “will“ represents an example of a too general word - an issue discussed in the User Evaluation section of the current chapter and partly solved in section 5.2.4. .“Times“ is related to “in fishing“ where the category “Times“ is related to FT (Financial Times). “Norma“ is a frequently appearing name of a person for the documents related to the “in fishing“ subtopic. An association can also be found between “rules“ and “suicide agent“.

Unfortunately, the categories are not particularly helpful for the second search task - Figure I.3. This does not mean that the clustering and the labeling algorithms are incorrect. It is simply related to the fact that automatically generated categories correspond to a different set of subtopics from the official subtopics (which are a list of countries with endangered species for each). The categories look quite appropriate for a “wildlife extinction“ query. Categories such as “endangered“, “species“,“wild“, “natural“ and “conservation“ appear correct. The “leakey“ category is a particularly constructive example as well. It refers to Richard Erskine Frere Leakey which is a Kenyan conservationist. The category is successfully associated with Kenya. The weights also seem correct -

strong logical relationships between the automatically generated categories and the official subtopics have higher weights.

6.3.3 Evaluation Results

In summary, the following conclusions were reached as a result of the log analysis. First, in agreement with the offline and user evaluations the Flat Clustering sub-query generating algorithm was found more effective than the baseline, xQuAD+Hierarchical Clustering, and xQuAD+Concept Hierarchy when there are no many subtopics of the original search topic. However, in contradiction to the offline and user evaluations xQuAD+Concept Hierarchy was found to have higher effectiveness than the other 3 system configurations when the number of subtopics is large. Also, It was discovered that the category labeling algorithm was able effectively identify sub-topics of the original search topic although these automatically generated sub-topics did not always correspond to the official subtopics provided with the test collection. The next section presents the different test strategies employed to test the system‘s functionality.

6.4 Testing

Several different testing strategies were explored to test the overall effectiveness of the system. First, unit tests were implemented to evaluate the performance of individual classes of the application in isolation [68]. Once the correctness of these classes was confirmed, they were tested in combination as components. In other words, integration testing was performed [30]. Finally, the complete and integrated software was tested via System tests. The purpose of these tests was to evaluate the systems compliance with the specified requirements [3].

6.4.1 Unit Testing

The unit testing provided a mechanism the main features of the system to be tested in isolation. The clustering algorithms, concept hierarchy related functionality, document data structure used to generate the clusters and several utility classes were tested via JUnit [31]. Most of the them relied on the IndexTestUtils Terrier class to generate a small representative index of simple documents represented as strings. This allowed the expected results of the functionalities above to be manually computed. Once IndexTestUtils is used to generate an index, the index can be queried.

For the tests related to the cluster generating algorithms the index was queried with a test query and the result set was clustered using either Flat or Hierarchical clustering (indexing and querying stages are required as the clustering algorithms rely on the posting lists and the ResultSet objects) (the tests are still considered unit tests as both the indexing and querying stages are only part of the Terrier environment). The documents in the test index are chosen in a way that the expected clusters are easily noticeable. Also, there is a single word which appears many times in all documents in the result set. The objective of this is to evaluate how the clustering algorithms perform with noisy data. Once the results are received from the tested clustering algorithms, they are compared with the expected clusters. The expected clusters are chosen by manually performing the same clustering algorithms. The cluster labeling algorithm is tested in a similar fashion, as the documents are split into clusters manually to test the functionality in isolation. Again words appearing many times in all documents are deliberately added to evaluate if the cluster labeling algorithm is able to identify frequent terms unique to the cluster. The concept hierarchy is tested by first comparing the terms selected by the term selector algorithm with the expected manually selected terms (performing the cluster labeling algorithm by hand). The concept hierarchy data structure is then generated using the selected terms and again compared with the expected concept hierarchy structure. Finally, the document object, storing key document statistics used by lingPipe to construct the clusters,

is also tested. Its word to counter map is traversed to verify that the correct TF*IDF is computed for each term in the document and its cosine distance with a second document is calculated to confirm the correctness of the document distance functionality. The utility classes are also tested but their tests are not discussed in the current paper due to their low-level programming nature.

6.4.2 Integration Testing

Once the correctness of the core application classes was established via the unit tests discussed above, an integration testing was perform to analyze how well they work together. The application can be naturally split into 3 main components. The first and second components generate sub-queries relying on flat or hierarchical clustering respectively. As part of the process they use the *Document* data structure, *Cluster Labeling* algorithm, several utility classes and either the flat or hierarchical clustering classes. On the other hand, the third component employs the *Concept Hierarchy* data structure, *Term Selecting algorithm*, *Document* data structure and several utility classes to generate sub-queries. These three components are tested by three integration tests. Figures G.1 and G.2 visualizing the components and the performed tests are available in appendix G.

6.4.3 System Testing

After it was verified that all of the system‘s components are functioning correctly four system tests were performed to test if the four possible execution paths of the system function as expected. To perform the tests the server was started and a test client was created using a ClientBuilder class [12]. The client instantiated from the builder performed 4 search request via 4 HTTP GET RESTFul requests to the server. For each of the four requests the returned search results, clusters and query expansion terms for each cluster were compared with the expected. The four tested configurations were the following - baseline (Terrier without xQuAD), Terrier+xQuAD using Flat Clustering sub-query generation and Flat Clustering for categorization of the search results, Terrier+xQuAD using Hierarchical Clustering sub-query generation and Hierarchical Clustering for categorization of the search results, and finally Terrier+xQuAD and Concept Hierarchy sub-query generation. In this way the behavior of the whole system was inspected.

6.5 Summary

The evaluation chapter presented the different mechanisms used to evaluate the system and analyzed the results of the different evaluation experiments considering the research questions introduced at the beginning of the chapter. The next chapter summarizes the current paper indicating the satisfied system requirements and discussing possible future work.

1. *Which of the three sub-query generating algorithms leads to highest effectiveness and best diversification of the search results?* - Through the findings of the offline and user evaluations (which were in agreement with each other), it was determined that the Flat Clustering sub-query generating algorithms leads to highest effectiveness and best diversification of the search results.

2. *How does system using the aboved-mentioned most effective sub-query generating algorithm performs when compared to the baseline and the ‘gold standard‘?* - Sections 6.1. and 6.2 indicated that the Flat Clustering sub-query generating algorithm performs better than the baseline but worse than the ‘gold standard‘.

3. *Which exploratory search algorithm is found as most effective by the users?* - The user evaluation showed that the algorithm based on the category‘s name best satisfies the users‘ needs.

Chapter 7

Conclusion

The current paper presented the aims and motivations of the Exploratory Search Engine project, the related research, system requirements, architecture and implementation, as the last chapter discussed the evaluation of the system. The current chapter summaries the satisfied requirements and reveals several areas of the application which can be improved if work was to continue on the project.

7.1 Satisfied Requirements and Reflection

As part of the final iteration of the Exploratory Search Engine application the majority of the project‘s functional and non-functional requirements were met. First, all “Must have“ requirements were satisfied. The efficient mechanism for storage and retrieval of data was achieved using Terrier,a client-side cache, and lingPipe for efficient clustering. Several methods for diversification of the search results were implemented and evaluated achieving maximum coverage and minimal redundancy via xQuAD. Two “exploratory search“ algorithms were implemented and a web-based GUI was developed to demonstrate the project. Finally, an “exploration history“ (FMF.5) feature was added. As to the “Should have“ requirement, two alternatives for labeling of the identified search result topics were discussed and the more effective one was implemented (section 5.2.4). Two algorithms for categorization of the search results were implemented and on the presentation tier relative scores for each search results were displayed. Also, the user was given the opportunity to switch between “exploratory algorithms“. For expert users all “could have“ requirements were satisfied.

The majority of non-functional requirements were also met. Efficient resource utilization on the client side was achieved using a thin-client architecture. The scalability, reliability and failure recoverability were achieved. Unfortunately, the response time non-functional requirement was not met when xQuAD is enabled due to the slow sub-query generating algorithms. In summary, all requirements except the last mentioned were achieved.

Working on this project was an opportunity to experience what it is like to work on the development of a search engine system. The project also had a considerable research aspect, as I had to read different research papers to get familiar what ‘Exploratory Search‘ is, what the challenges it attempts to solve are and how they can be solved using existing mechanisms for identification of sub-intents. I had to customize these algorithms for sub-intent identification, to integrate them to the project and to evaluate their effectiveness. I had never done an offline evaluation or a log analysis of a system before, hence this was once again a chance for me to develop new skills. At the beginning, working on the project was quite stressful, as I was not familiar with the IR field and had difficulties understanding research papers and algorithms/concepts which my supervisors were discussing with me. However, once I acquired the required knowledge, I enjoyed working on the project and started to experiment/modify some of these algorithms/concept to improve the retrieval capabilities of the system.

7.2 Future Work

There were several implementation ideas which were not implemented as part of the current version of the application. However, if there was more time or the work was to continue, they would represent beneficial additions to application.

7.2.1 Utilization of the List of Relevant Results

The list of relevant results is just a presentation tier functionality allowing collection of the relevant documents at the moment. However, this user input can be beneficial not only to the user but to the search engine as well. The gathered relevant documents can be used to improve the relevance of the result set via, for example, a query expansion of the documents in the list of relevant documents [74].

7.2.2 Effectiveness/Efficiency Tradeoff

In an attempt to improve the effectiveness of the Terrier search engine the current application slowed down its retrieval process. Several different techniques are available this drawback to be addressed. For example, static or dynamic pruning can be used to speed-up search [11]. If more memory, hard disk space and users were available caching of subtopics of frequent queries would be extremely beneficial, as this will completely cut the time for sub-query generation. Index compression and paired posting lists are other alternatives the efficiency to be increased.

7.2.3 Enhancement of the Automatic Labeling of Document Clusters

Several drawbacks of the automatic labeling algorithm were identified during the evaluation of the application. Most of them were related to too general terms which were used as category labels. There are several techniques this issue to be solved. First, a larger stop words list can be used to filter out too general terms. Alternatively, users can be allow to judge the effectiveness of the labels. If a particular number of users considering a label as too general is reached, the label can be added to a black list.

7.2.4 Search Autocomplete

Search suggestions for automatic completion of the search query can be particularly useful for an exploratory search. As a remind - users performing an exploratory search are not experts in the searched field, have unclear goal and experience difficulties to formulate their query by definition. Hence, such a functionality relying on log data and/or the n-gram statistics generated from the searched corpus can be highly useful [13].

7.2.5 Automatic Detection of the Number of Search Result Categories

An issue not addressed by the implementation of the Flat Clustering algorithm in the current implementation is the problem of determining the number of clusters. At the moment a default value or a value selected by the user is used. However, an ideal situation will be this number to be automatically determined via some statistical methods, such as X-means clustering [37], the elbow method [21], Akaike information criterion (AIC) [45], Bayesian information criterion (BIC) [5], or the Deviance information criterion (DIC) [7].

Appendices

Appendix A

Dependency Injection via Jersey

The Jersey framework contributes to the simplicity of server architecture via its Dependency Injection mechanism. The Dependency injection pattern makes the code more reusable, testable and readable [64]. The code is more reusable because the dependencies are injected externally. If a different implementation of the injected interface is necessary, this change can be easily specified without changing the code of the component. Additionally, classes are more testable because the injected dependencies can simply be replaced by injecting mock implementations. Furthermore the code becomes more readable, as the dependency injection scheme requires the list of all dependencies to be stated in the ApplicationBinder class. These dependencies are then annotated with an @Inject annotation inside each class which depends on them, so they can be easily noticed. For example, the AppConfig dependency is instantiated in the QueryResource class of the Exploratory Search Engine application, as shown on Figures A.1:

```
public class ApplicationBinder extends AbstractBinder {
    @Override
    protected void configure() {
        bind(AppConfig.class).to(AppConfig.class);
    }
}
```

Figure A.1: The ApplicationBinder class defining all dependencies.

The ApplicationBinder class includes a list of pairs indicating all dependencies in the system. In the example above it has only one dependency which is the AppConfig class. The bind function specifies that when a field annotated with @Inject of type AppConfig class is found it should be instantiated with the class pointed out in the “to“ method which in the scenario above is again the AppConfig class.

```
@ManagedBean
@Path("queryResource")
public class QueryResource {

    @Inject
    private AppConfig appConfig;
```

Figure A.2: Injection of the AppConfig class in the QueryResource class.

Appendix B

Vector Space Model

The Vector Space Model relies on the concept that documents can be represented as vectors. It is used for diverse set of information retrieval operations such as scoring documents, classification and clustering. Each document is represented as a vector in the Vector Space Model in the following way. An n-dimension space is created where n is the number of terms in the corpus lexicon. A document vector is a vector with n components where each component corresponds to a term weight generated using some weighting scheme such as tf-idf [27]. Viewing each document in a corpus as a vector forms a set of vectors in a vector space with one axis for each term. Once the vector space is built, the identification of similar documents and their grouping into cluster becomes straightforward. Their similarity is calculated using the cosine similarity formula where V(d1) and V(d2) indicate the document vectors of document d1 and d2. Once it is known how similar two documents are, splitting them to clusters is trivial.

$$\mathrm{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

Figure B.1: Cosine similarity formula [27].

Appendix C

Hierarchical Clustering and the Dendrogram Object

The hierarchical clustering generates expectedly a hierarchical representation of the documents in the result set. For this reason, it is considered more informative when compared to the flat clustering [27]. An additional advantage is that the number of clusters must not be specified in advance, as all documents are simply stored in the hierarchical structure. As a result, the allocation of documents into clusters must be recomputed from the hierarchical structure each time when it is necessary the clustered documents to be shown to the user. In other words, the flexibility of unspecified number of clusters comes at the cost of lower efficiency.

The LingPipe framework allows the generation of hierarchical clusters from a data structure known as Dendrogram [63]. This is the mechanism used by the Exploratory Engine application for generation of hierarchical clusters. A Dendrogram is a binary tree over the result set of a query which stores in each internal node the distance between the two sub-branches. The documents themselves are stored as leaves of the Dendrogram. Figure C.1 below presents a Dendrogram clustering strings and the corresponding distance between the strings:

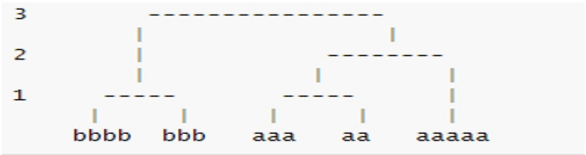


Figure C.1: An example of a Dendrogram from the LingPipes tutorial page [63].

The Dendrogram is constructed using a set of documents where each instance of the Document object must have cosine distance and vector product methods and length and cosine fields. All of these methods are required by the LingPipe framework to construct the clusters. The cosine value corresponds to the value computed from the Cosine similarity formula introduced Appendix B. The length value is the vectors length. The product is the standard vector product. Finally, the cosine distance according to the LingPipes tutorial is calculated as follows: Cosine is a proximity measure in that a higher cosine value between two vectors indicates that they are more in the same direction. Thus to define a distance, we subtract the cosine between document vectors by one [63].

Appendix D

Concept Hierarchy

The Deriving concept hierarchies from text paper written by Mark Sanderson and Bruce Croft examines how a hierarchical organization can be created from a set of documents without using standard clustering techniques [43]. The algorithm gathers key words and phrases from the collection of documents and organize them hierarchically. The approach is particularly useful for two main reasons. First, compared to the standard clustering approach the concept hierarchy delivers more information in the form of hierarchical relationship. Second, it manages to overcome the popular clustering problem related to the labeling of the clusters. Instead of splitting the collection of documents into similar clusters which can be labeled with great difficulty and frequently have ambiguous and not representative labels the Concept Hierarchy approach identifies the hierarchy of topics in the corpus first without associating them to clusters. Ordinary clustering algorithms first identify the similarities between the documents and gather them into clusters using documents with similar set of words and phrases. Unfortunately, labeling such a selection is quite hard, as it is based on similar terms and phrases not on an unique term. Hence, it is necessary a general enough term or terms to be identified to summarize all of these common terms in the cluster. Such a task is hard and leads to a too general and frequently vague labels. This issue is based on a approach known as polythetic clustering [41]. On the other hand, the concept hierarchy generates monothetic clusters clusters where membership is based on only one feature, the concrete term in the hierarchy itself, which can be directly used as a label [43].

The Concept Hierarch algorithm is implemented in the Exploratory Search Engine application to provide an alternative way for generation of sub-topics for xQuAD - it is expected to be particularly effective due to its monothetic nature as described above. The algorithm can be used as well as an alternative of the clustering scheme but this use is not presented by the current application. The algorithm consists of two stages the terms selection and the hierarchy creation. The next several paragraphs give a more thorough explanation of the algorithm.

D.1 Term Selection

Given a collection of documents the terms selectors responsibility is to identify key phrases and words which can be used for the generation of the Concept Hierarchy. It uses several different sources and methods to gather the most significant words. The initial source is the query terms themselves. A *Query Expansion* is performed as an alternative method to gather more diverse set of terms, more precisely [43]:

"This works in the following manner, an initial set of documents is retrieved in response to the original query and the best matching passages of the top ranked documents are examined to find words and phrases that commonly co-occur with each other across many of the passages."[43]

The third approach is statistical and uses an empirically derived threshold. It is defined in the following way:

$$X_r \, / X_c > 0.1$$

where x_r is the frequency of occurrences of term x in the retrieved set and x_c is its frequency of occurrences in the collection.

D.2 Hierarchy Creation

The generation of the hierarchy is based on the identification of parent-child relationships [43]. If we have two terms x and y where x is the parent then the following dependencies must be satisfied:

$$P(x|y) \geq 0.8, P(y|x) < 1$$

Figure D.1: Deriving Concept Hierarchy from text [43].

In other words x subsumes y if the documents, which y occurs in, are a subset of the document which x occurs in. The first inequality checks if x occurs in almost all document which y occurs in. Note that the value on the right hand side is not exactly 1.0 as it was proven via empirical observations that such a strict requirement is frequently not satisfied even in case when x subsumes y. The value of 0.8 was identified as high enough through experimentation. The second inequality actually confirms the set of documents y occurs in is really a subset, as it is possible the two words to be synonyms and the set of documents of x and y to be the same set of documents. This reasoning is a bit more subtle for which reason it is explained in more detail in the Implementation Chapter.

Appendix E

GUI Design

The Modern Information Retrieval book by Ricardo Baeza and Berthier Ribeiro, and more precisely Chapter 10 *User Interface and Visualization* was a main source of the design decisions [2]. The chapter discusses different search engine designs highlighting some of their drawbacks and advantages.

E.1 The Scatter/Graph Clustering Design

The first design considered appropriate for the system was an adaptation of the Scatter/Gather clustering design [2]. The Scatter/Gather shows a textual representation of the document clusters. It represents an accordion graphical control element where each section is a cluster containing the label of the cluster, its size and number. The content of each section includes topical titles that hint the theme of the cluster.

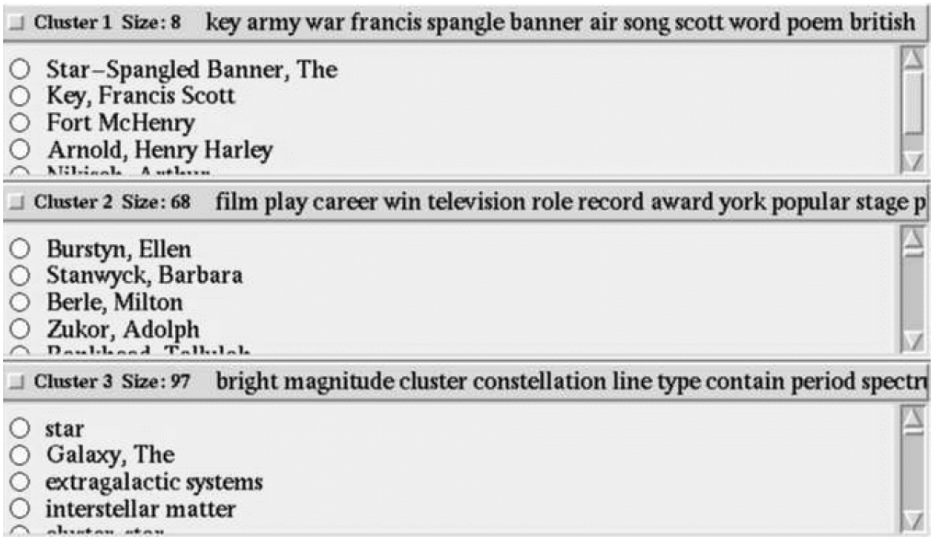


Figure E.1: Display of Scatter/Gather clustering retrieval results [10].

On the other hand, the adaptation of this approach had to include the results themselves in each section of the accordion instead of a list of topical titles. The approach would reduce the working memory of the reader as the results would be split into clusters, so he would not have to remember or write down a list of his/her favorite results corresponding to a given cluster. Also, the design had the advantage that once the relevant cluster was identified, the user could browse only in this section of the accordion speeding up the process instead of scrolling the page looking for more results from this topic.

Unfortunately, the drawbacks over-weighed the benefits and the design was not implemented. Some of the main concerns were related to ranking, paging, space utilization and number of clusters. If results are allocated to categories then the initial ranking of the documents will be lost and there will be only ranking within the accordion sections which represents a considerable concern. The paging functionality can get complex and confusing as well. Most likely each accordion section will need its own pagination to allow flexibility but such a design is intuitively confusing as when a user see several lists with results he or she will logically assume they have relatively the same rank even though in reality they do not. Space utilization was another concern. If the number of clusters is large, a large part of the page will consist of clusters‘ titles instead of results even when some of the sections are collapsed.

E.2 Graphical Bars

An alternative design decision explored in the Modern Information Retrieval book was the TileBars [2]. Reading the chapter was an inspiration for the addition of the graphical bars presented on figure F.2 to the current project.

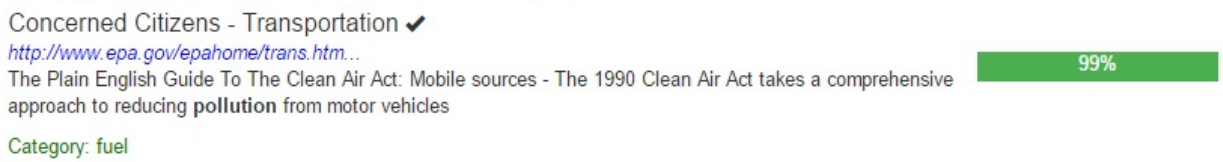


Figure E.2: Relative result scores represented as graphic bars in the Exploratory Search Engine application.

After the user enters a query and the results are retrieved a graphical bar appears next to each title showing the percentage of match to the original query . As there was no way to measure what 100% match with the original query means the graphical bars were modified in the Exploratory Search Engine application to only show the percentage of relevance to the top result, as shown on figure F.2:

The design decision has plenty of benefits. First, it allows the user to make a quick decision if a particular document is relevant only by looking at the bar which takes considerably less time than reading the result‘s abstract. Also, it is easy to capture how effective the current search is. To achieve this the user has to look at a random result from the first page and if the result is considered irrelevant, for example, and its relative score is high, then most likely the entire result set does not contain useful information. In the scenario of an exploratory search where the user is not a specialist in the field he or she is searching, the graphic bars can easily navigate him/her to the most appropriate cluster as it is easy to see the documents from which cluster have the highest total percentage of match.

E.3 Graphical Overviews

The Modern Information Retrieval book also assisted in the identification of design options which must be avoided, namely the graphical overviews. There were a lot of disadvantages mentioned during a study related to different types of graphical overviews explained in the book [2]. For example, one drawback was related to the fact that a graphical overview of a large corpus can be quite complex and include many different topics. Visualization of such a complexity requires a zooming functionality. Such a necessity can represent a technical challenge as it requires some form of a hierarchical organization and an efficient algorithm to retrieve the expected results during zooming. Other users disliked having to look through the entire map to find a theme and maybe

the most concerning point was that labels were misleading and confusing [2]. For these reasons it was concluded that graphical overviews should be avoided.

E.4 Standard Relevance Feedback

A relevance feedback list or any other form of a feedback structure can be used to support the user in his/her process of gathering relevant documents from the result set. Additionally such a feature will generate more data which can be sent to the back-end to improve the effectiveness of the next query based on the feedback given for the current one. The standard relevance feedback graphical design presented in the *Modern Information Retrieval book* consist of a list of titles (the titles of the documents from the result set) and checkboxes next to each title allowing the user to select the document as relevant or to ignore the checkbox in which case it will be considered that he/she has no opinion about the document [2]. Alternative options include the use of two checkbox one for marking relevance and another for irrelevance (where no selections means neutral opinion) or a relevance scale [17].

The standard relevance feedback has the two advantages mentioned above but it is not a very popular feature among the most used search engines, hence it can represent a design challenge to develop an interface intuitive enough to be unambiguous and motive users to experiment with it [2]. The Implementation chapter illustrate how the functionality shown in the book is implemented in practice and how some of the concerns identified from the research are addressed in the implementation.

Appendix F

Scripts

All additional scripts which were used during the evaluation of the application are available in this appendix.

F.1 Automatic Log Analysis

The following script summarizes the evaluation log results presented in section 6.3:

```
import sys

with open(sys.argv[1]) as f:
    content = f.readlines()
# you may also want to remove whitespace characters like `\n` at the end of each line
content = [x.strip() for x in content]
del content[0]
catDict ={}
catSubtopic={}

newContent = []
numRelDocs = 0
numOfUsers = 10
for line in content:
    lineSplit=line.split(" | ")
    if len(lineSplit)==1:
        continue
    if lineSplit[3] in catDict.keys():
        catDict[lineSplit[3]]=catDict[lineSplit[3]]+1;
    else:
        catDict[lineSplit[3]]=1;
    if (lineSplit[2]+"->"+lineSplit[3]) in catSubtopic:
        catSubtopic[lineSplit[2]+"->"+lineSplit[3]]= catSubtopic[lineSplit[2]+"->"+lineSplit[3]]+1
    else:
        catSubtopic[lineSplit[2]+"->"+lineSplit[3]]= 1
    numRelDocs+=1
print "1)"
for key in catDict.keys():
    print key + ":"+str(catDict[key])
print "-------------"
print "2)"
for key in catSubtopic.keys():
    print key + ":"+str(catSubtopic[key])
print "-------------"
print "3)"
print "Average number of relevant documents selected by a user"
print float(numRelDocs)/numOfUsers
```

Figure F.1: Automatic log analysis.

F.2 Graphical Visualization of the Accuracy of the Category Labels

The following script generates a .gv formatted file used for the creation of a category label to subtopic weighted graph using Graphviz:

```
#!/usr/bin/python
import sys

with open(sys.argv[1]) as f:
    content = f.readlines()
# you may also want to remove whitespace characters like `\n` at the end of each line
content = [x.strip() for x in content]
catDict = {}
catSubtopic = {}

for line in content:
    lineSplit = line.split(":")

    if lineSplit[0] in catDict.keys():
        catDict[lineSplit[0]] += int(lineSplit[1])
    else:
        catDict[lineSplit[0]] = int(lineSplit[1])


secondDict = {}
counter = 6
for key in catDict.keys():
    if catDict[key]<9:
        continue
    keys=key.split("->")
    if keys[1] not in secondDict:
        secondDict[keys[1]]=counter
        counter+=1
    print keys[0] + " -> " +"{ makeString" + str(secondDict[keys[1]]) + " }" + " [label=\""+str(catDict[key])+"\"];"
for key in secondDict:
    print "makeString"+str(secondDict[key]) + " [label=\"" + key + "\"]"
```

Figure F.2: Category label - subtopic weighted graph generating script.

Appendix G

Integration Testing Diagrams

Figures G.1 and G.2 visualize the three integration tests.

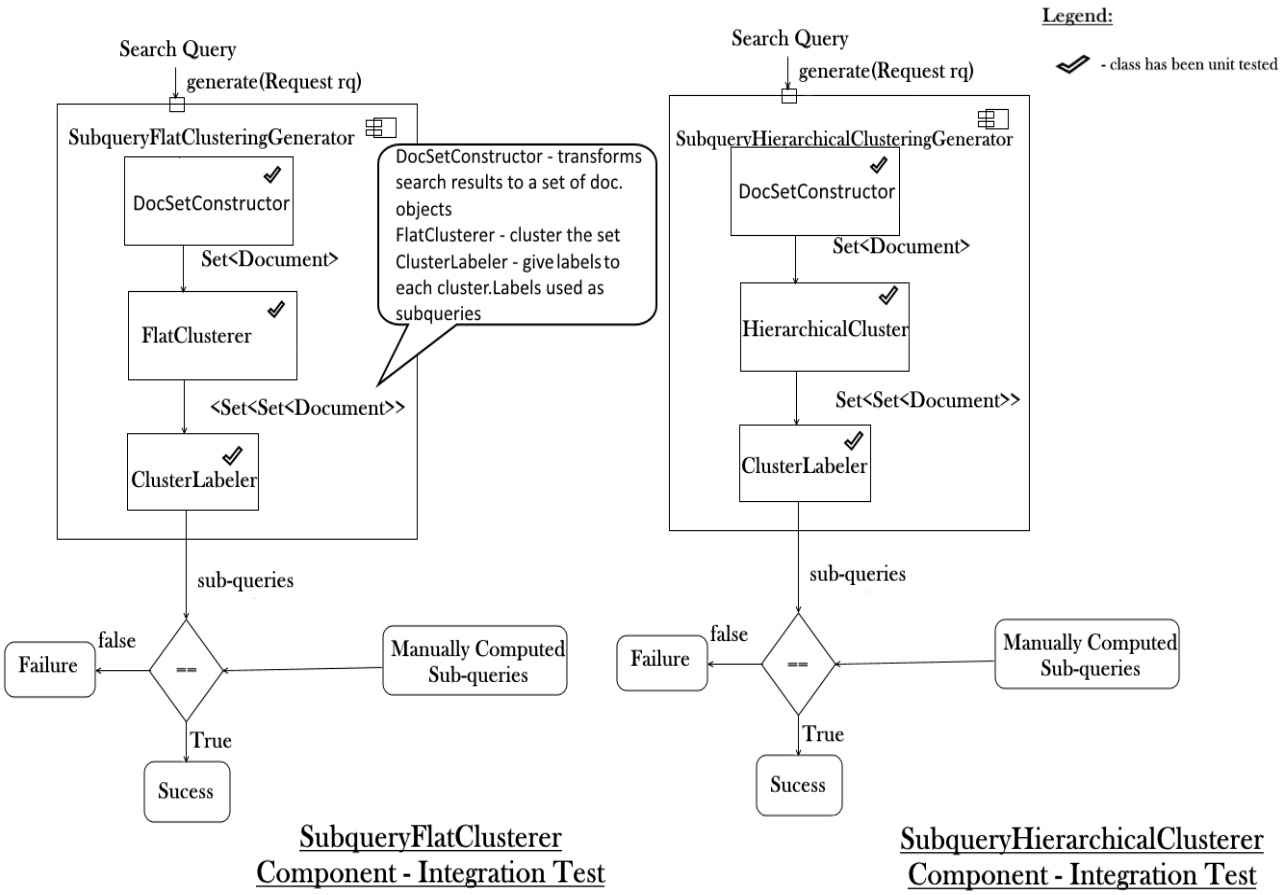


Figure G.1: Integration Testing - Sub-query Flat and Hierarchical Clustering Generators.

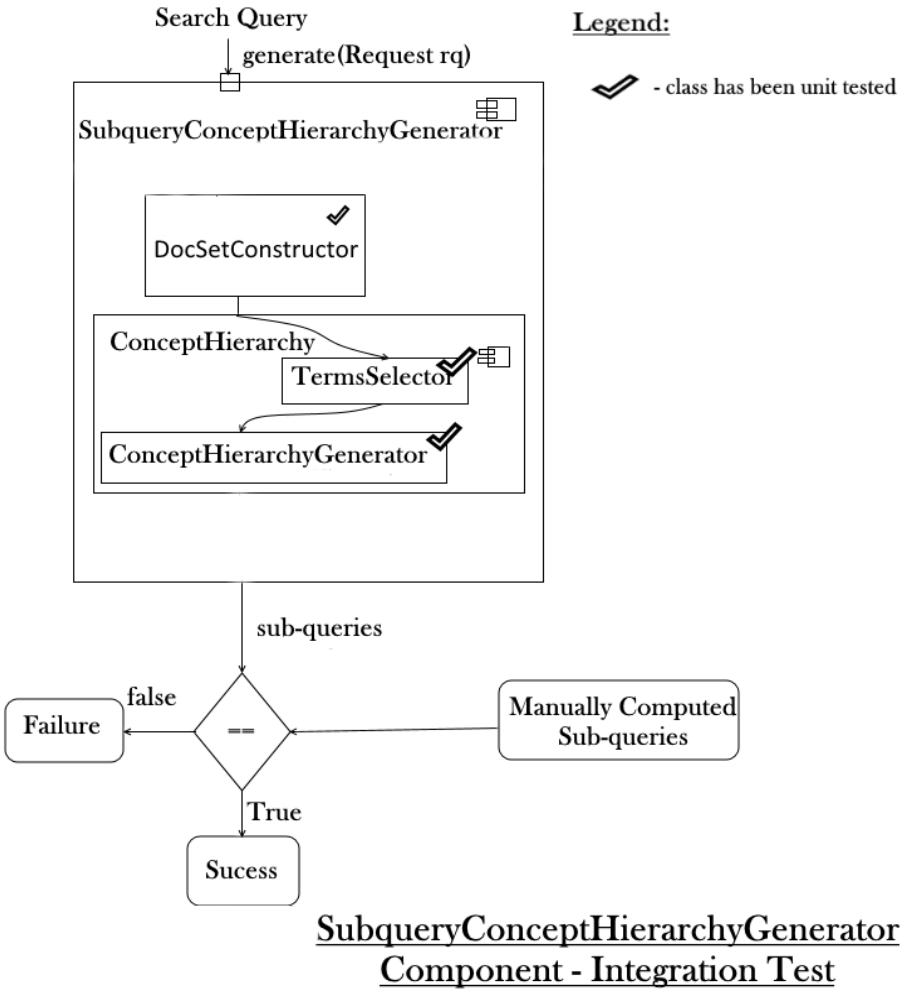


Figure G.2: Integration Testing - Sub-query Concept Hierarchy Generator.

Appendix H

GUI - Introductory Figures

Figures H.1 and H.2 below illustrate the different parts/functionalities of the Exploratory Search Engine GUI interface.

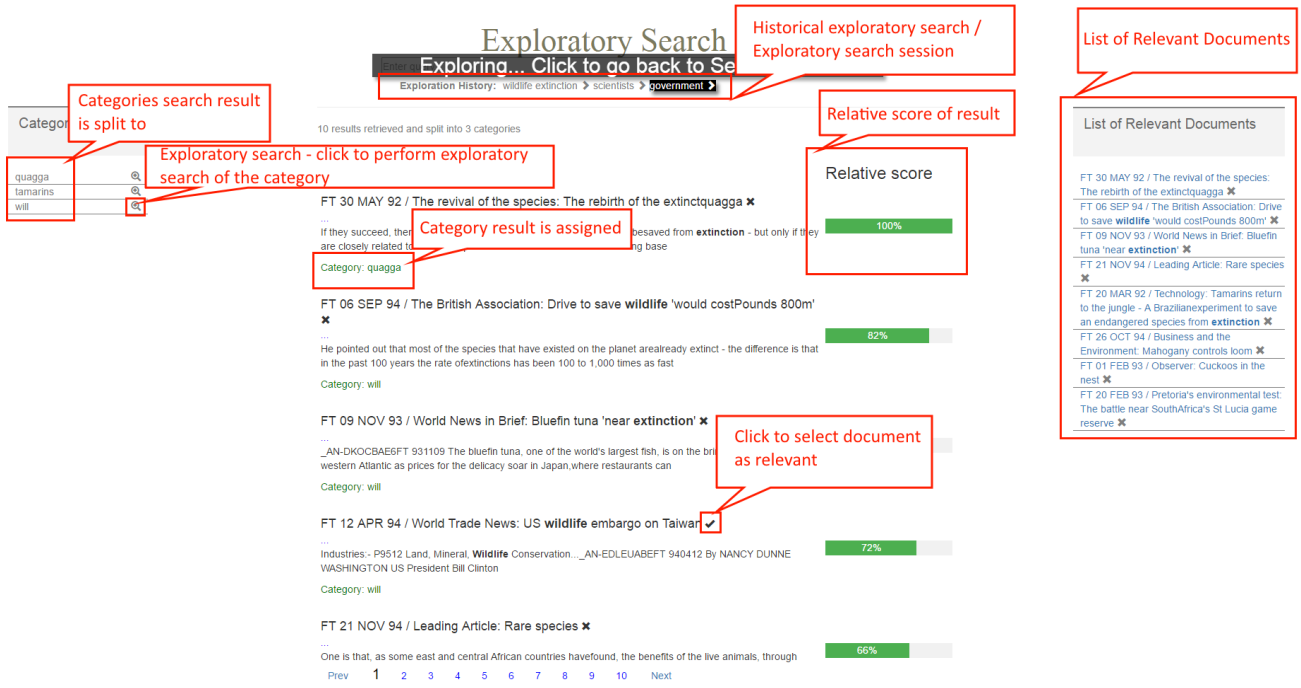


Figure H.1: GUI - Introductory Figure.

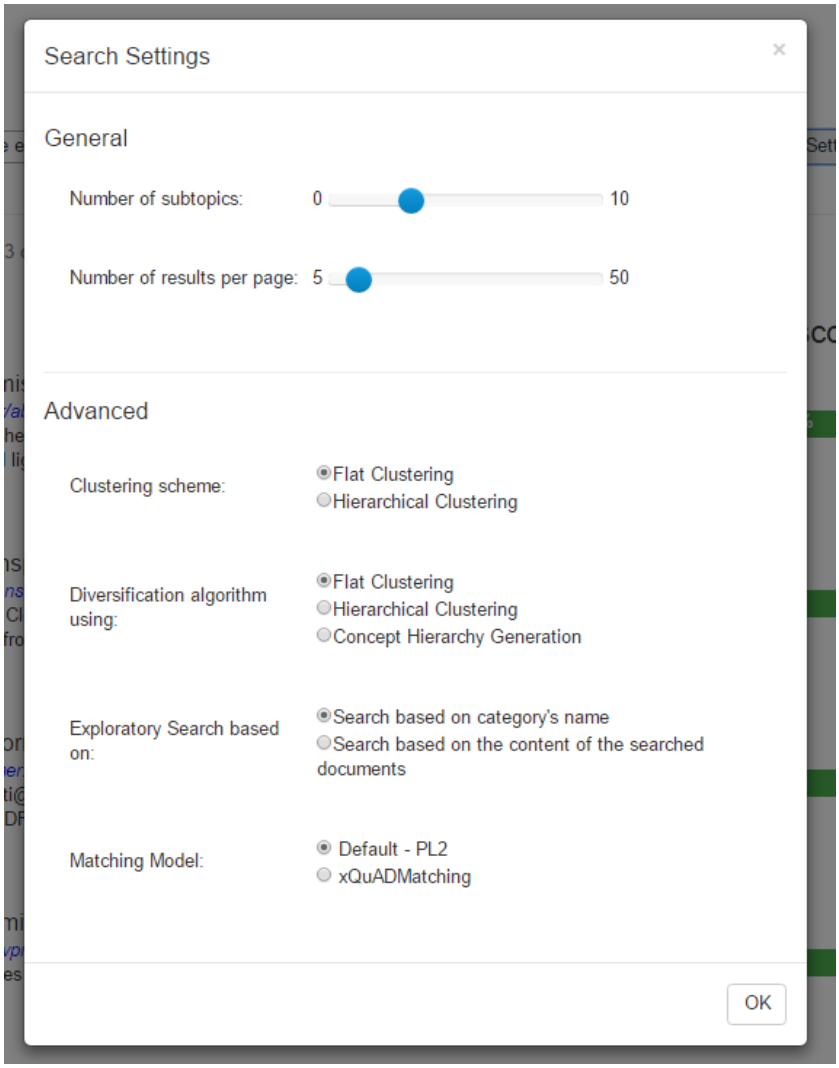


Figure H.2: GUI - Settings.

Appendix I

Evaluation - Figures

I.1 Recall-Precision graph of the three Sub-query Generating Algorithms

The graph below indicates the recall-precision relationship between Terrier without xQuAD, Terrier +xQuAD via the flat cluster sub-query generating algorithm, and Terrier +xQuAD relying on the official subtopics.

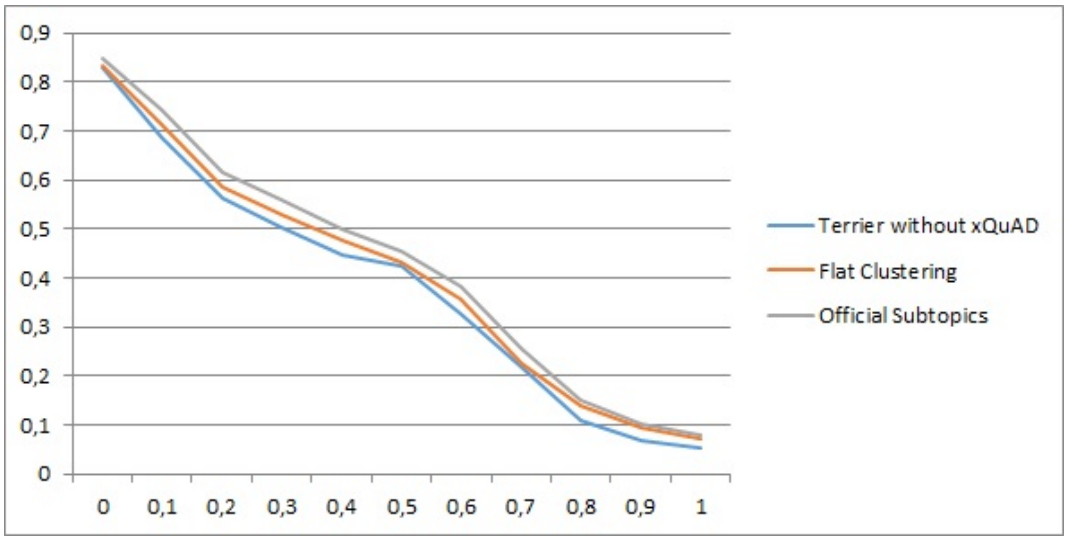


Figure I.1: Recall-Precision graph of the three Sub-query Generating Algorithms.

I.2 Category label - Subtopic Weighted Graphs

Figures I.2 and I.3 show a category label to a subtopic directed weighted graphs constructed from the users‘ logs gathered during the evaluation of the project for the two tested search tasks.

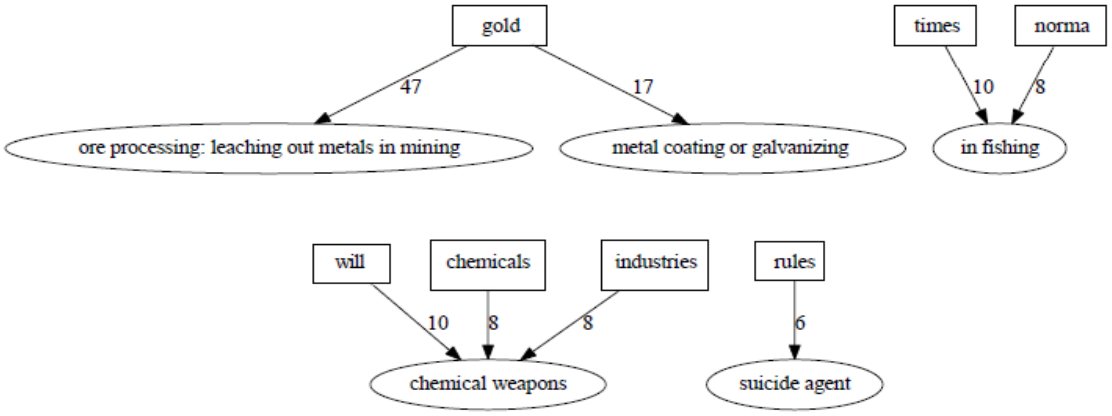


Figure I.2: Search task 1 - Category label-subtopic weighted graph.

Square node indicates an automatically generated category label, while an oval is used as a sign for an official subtopic. The weight of an edge shows the number of relevant documents selected by the participants where documents are associated to the particular official subtopic and the automatically generated category.

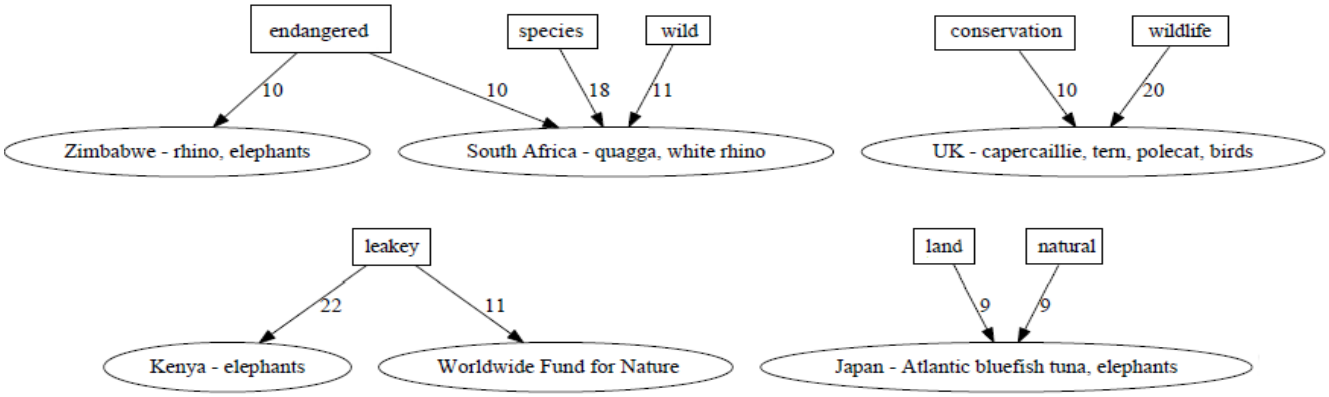


Figure I.3: Search task 2 - category label-subtopic weighted graph.

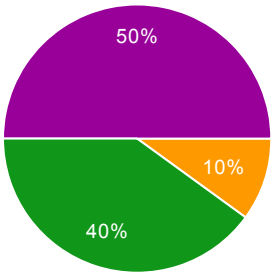
Appendix J

Pre-Evaluation Questionnaire

The next 2 pages represent the pre-evaluation questionnaire‘s results generated by Google Forms.

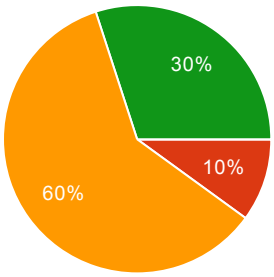
Summary

How often do you use search engines?



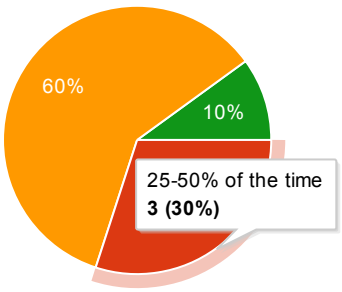
never	**0**	0%
once a week	**0**	0%
once a day	**1**	10%
more than 10 times a day	**4**	40%
more than 30 times a day	**5**	50%

How often do you search for information in a field you are not familiar with?



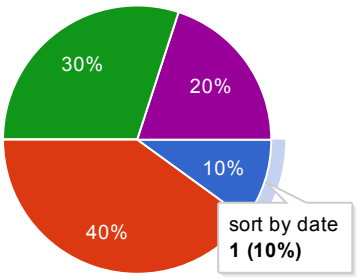
never	**0**	0%
once a week	**1**	10%
once a day	**6**	60%
more than 10 times a day	**3**	30%
more than 30 times a day	**0**	0%

How often is the retrieved information relevant to you?



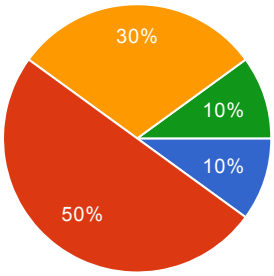
0-25% of the time	**0**	0%
25-50% of the time	**3**	30%
50-75% of the time	**6**	60%
75-100% of the time	**1**	10%

In a scenario where you search for information in a field you are not familiar with, give examples of search engine features supporting your search?



sort by date	**1**	10%
Categorization	**4**	40%
List with relevant results	**0**	0%
Graph with categories	**3**	30%
Graphical bars representing document scores	**2**	20%

How often do you have difficulties formulating your query when searching for information in a field you are not familiar with?



0-25% of the time	**1**	10%
25-50% of the time	**5**	50%
50-75% of the time	**3**	30%
75-100% of the time	**1**	10%

Appendix K

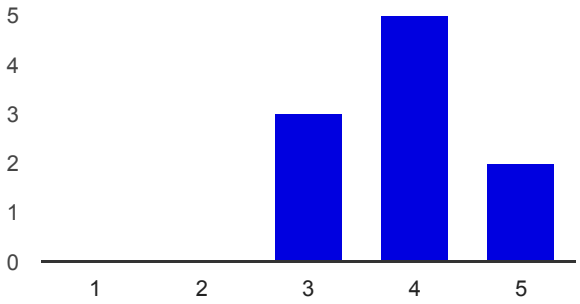
Guided-Think Aloud Supplementary Questionnaire

The next 8 pages represent the guided-think aloud supplementary questionnaire‘s results generated by Google Forms.

Summary

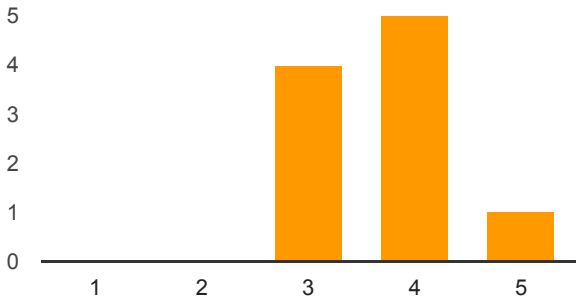
Baseline evaluation stage

The search returned results relevant to the query.



Strongly Disagree:	1	**0**	0%
	2	**0**	0%
	3	**3**	30%
	4	**5**	50%
Strongly Agree:	5	**2**	20%

I managed to accomplish the assigned task.



Strongly Disagree:	1	**0**	0%
	2	**0**	0%
	3	**4**	40%
	4	**5**	50%
Strongly Agree:	5	**1**	10%

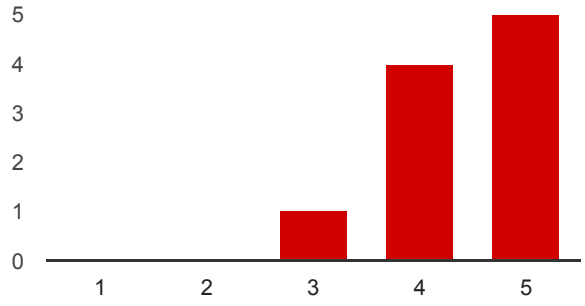
Comments:

The extract is too small . The content of the top documents repeats. All of the top documents include only a description of the diet.

Found uses: - ore processing: leaching out metals in mining (gold, zinc, lead, copper) - commodity to be recovered from waste or manufactured and sold - metal coating/galvanizing content of documents repeats

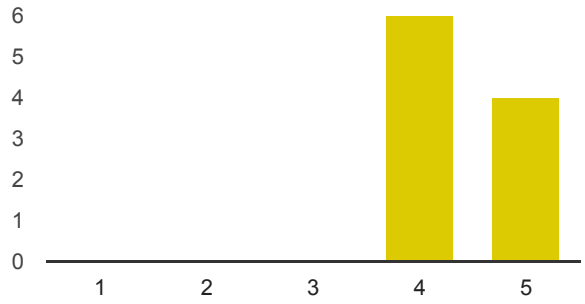
Second system configuration

The search returned results relevant to the query.



Strongly Disagree:	1	**0**	0%
	2	**0**	0%
	3	**1**	10%
	4	**4**	40%
Strongly Agree:	5	**5**	50%

I managed to accomplish the assigned task.



Strongly Disagree:	1	**0**	0%
	2	**0**	0%
	3	**0**	0%
	4	**6**	60%
Strongly Agree:	5	**4**	40%

Comments:

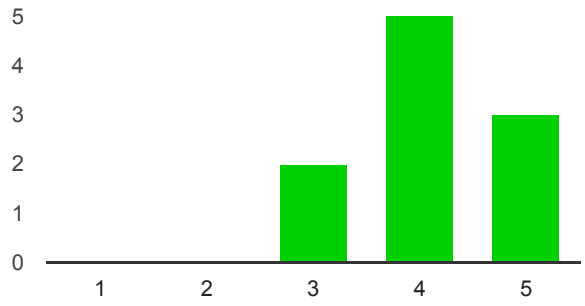
Information is not so repetitive (small number of relevant results). Clinical trials of the effects of the diet are discussed. Articles from specialists commenting the diet are also part of the top results.

The websites were sorted in a different way showing the more useful to me at the top.

Found uses: - leaching out metals in mining (gold was a category name) - commodity to be recovered from waste or manufactured and sold (waste was a category) - chemical weapons - in fishing - metal coating/galvanizing (2 uses not found)

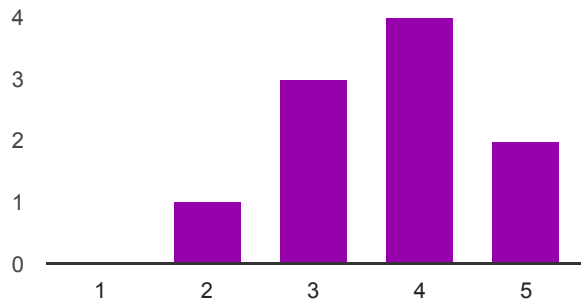
Third system configuration

The search returned results relevant to the query.



Strongly Disagree:	1	**0**	0%
	2	**0**	0%
	3	**2**	20%
	4	**5**	50%
Strongly Agree:	5	**3**	30%

I managed to accomplish the assigned task.



Strongly Disagree:	1	**0**	0%
	2	**1**	10%
	3	**3**	30%

	4	**4**	40%
Strongly Agree:	5	**2**	20%

Comments:

Most relevant top results are similar (small number of relevant results), as some of the more irrelevant subtopics such as the documents about the trials and the articles are in different order.

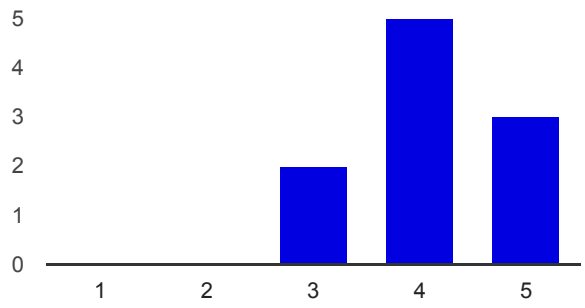
- ore processing: leaching out metals in mining (gold, zinc, lead, copper) - commodity to be recovered from waste or manufactured and sold - in fishing - metal coating/galvanizing

did like the fact that I have selected the number of categories

most of the top results are similar

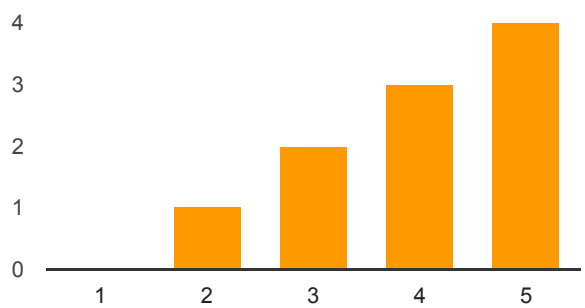
Fourth system configuration

The search returned results relevant to the query.



Strongly Disagree:	1	**0**	0%
	2	**0**	0%
	3	**2**	20%
	4	**5**	50%
Strongly Agree:	5	**3**	30%

I managed to accomplish the assigned task.



Strongly Disagree:	1	**0**	0%
	2	**1**	10%
	3	**2**	20%
	4	**3**	30%
Strongly Agree:	5	**4**	40%

Comments:

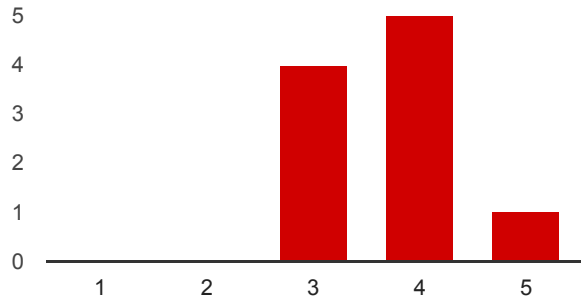
The document about the trials appear before some more relevant results describing what DASH diet is.

the results got too general.

results are to general.

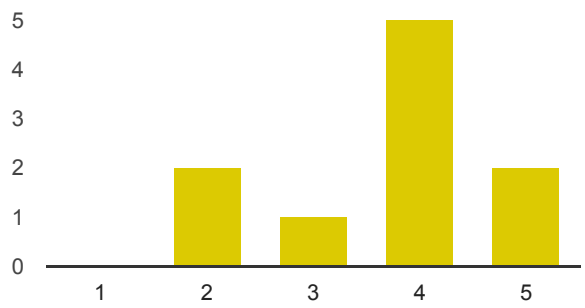
Categorization of the Results

The returned documents were related to the categories they were allocated to.



Strongly Disagree:	1	**0**	0%
	2	**0**	0%
	3	**4**	40%
	4	**5**	50%
Strongly Agree:	5	**1**	10%

The categories represent different/independent subtopics of the query?



Strongly Disagree:	1	**0**	0%
	2	**2**	20%
	3	**1**	10%
	4	**5**	50%
Strongly Agree:	5	**2**	20%

Comments:

Most of the returned documents are related to a single category, while the other 2 categories include only 1-2 docs each appearing down in the ranking and usually not related to the query. The inaccurate categories still describe documents related to the category topic even thought the topic is not a subtopic of the query. Categories` labels appear different but it is difficult to determine if the content of the documents between the groups is similar due to the limited number of docs in the other 2 groups.

The search engine shows too many results based on single category and the rest of the categories are left with only 1 or 2 search results.

Some clusters have same name. Some of the category labels are too general (looks like they are synonyms to each other)

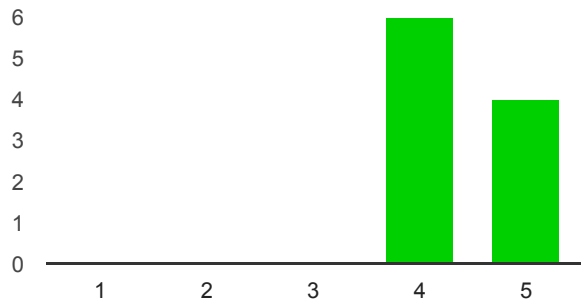
some of the category labels are too general (look like they are synonyms to each other)
some category labels are only related to one or two documents in which the category label appears many times

some of the categories are related to each other. Other have very general verbs as titles for the category

Exploratory search feature

First system configuration

The exploratory search feature was accurate.



Strongly Disagree:	1	**0**	0%
	2	**0**	0%

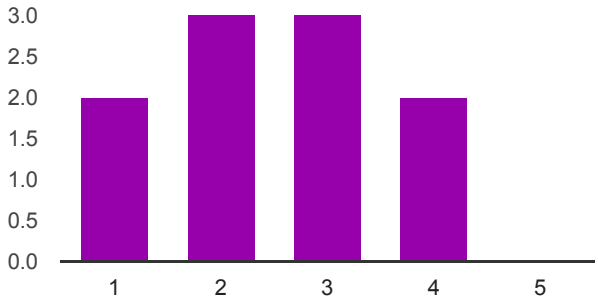
	3	**0**	0%
	4	**6**	60%
Strongly Agree:	5	**4**	40%

Comments:

If an exploratory search of the category with most results (related to comments above) is performed, the results are quite accurate. If some of the other categories is used, the results are so relevant.

Second system configuration

The exploratory search feature was accurate.



Strongly Disagree:	1	**2**	20%
	2	**3**	30%
	3	**3**	30%
	4	**2**	20%
Strongly Agree:	5	**0**	0%

Comments:

The results get more general than more specialized

Sometimes unrelated results

if a category with a well defined category names is explored - results are quite accurate.

results get too general

general

it looks like random words are bolded in the search results

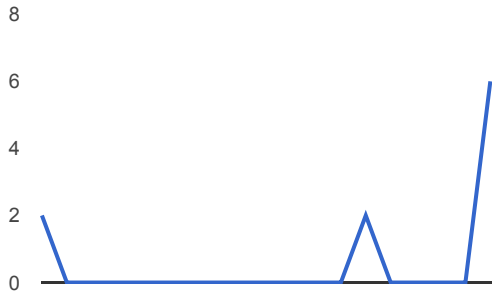
unrelated

words which are not part of search get bold

results get too general

first exploration is correct but it gets too general after this

Number of daily responses



Appendix L

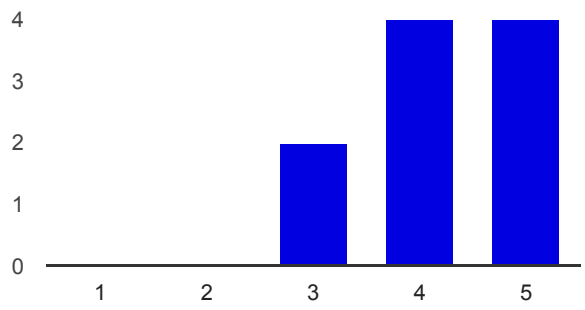
Post-Evaluation Questionnaire

The next 4 pages represent the post-evaluation questionnaire‘s results generated by Google Forms.

Summary

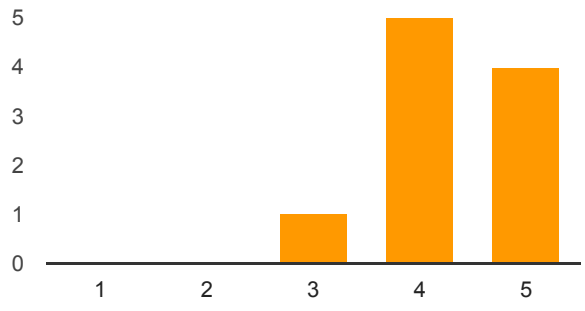
Usability and SUS Scale

I think that I would like to use this system frequently.



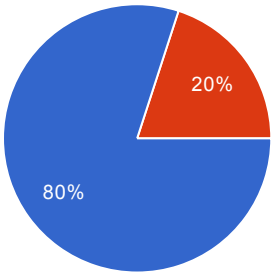
Strongly Disagree:	1	**0**	0%
	2	**0**	0%
	3	**2**	20%
	4	**4**	40%
Strongly Agree:	5	**4**	40%

I thought the system was easy to use.



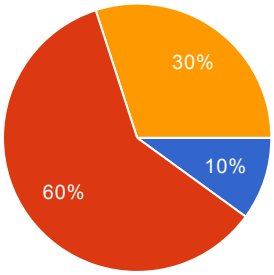
Strongly Disagree:	1	**0**	0%
	2	**0**	0%
	3	**1**	10%
	4	**5**	50%
Strongly Agree:	5	**4**	40%

How long are you willing to wait for a query response to get better suggestions for result categories?



0-5 sec	**8**	80%
5-10 sec	**2**	20%
10-20 sec	**0**	0%
20-30 sec	**0**	0%

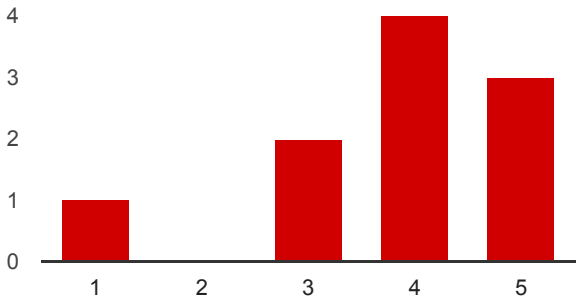
Do you think you have found all relevant results



Yes	**1**	10%
No	**6**	60%
Maybe	**3**	30%

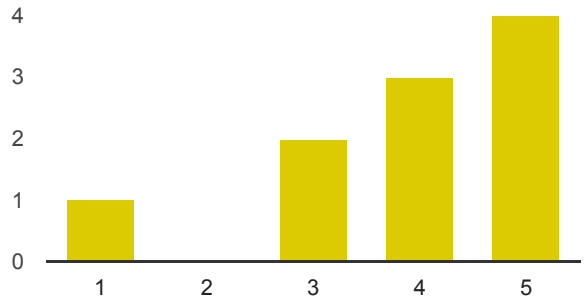
Functionality-related Questions?

I found the functionality partitioning the results into several categories useful.



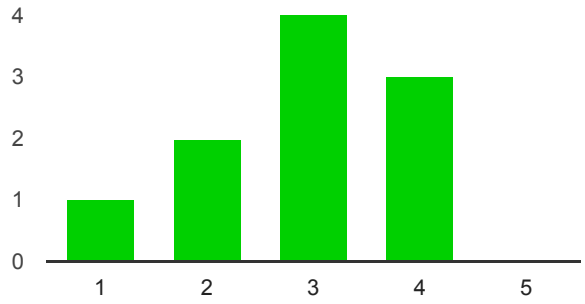
Strongly Disagree:	1	**1**	10%
	2	**0**	0%
	3	**2**	20%
	4	**4**	40%
Strongly Agree:	5	**3**	30%

I found the exploratory search functionality useful?



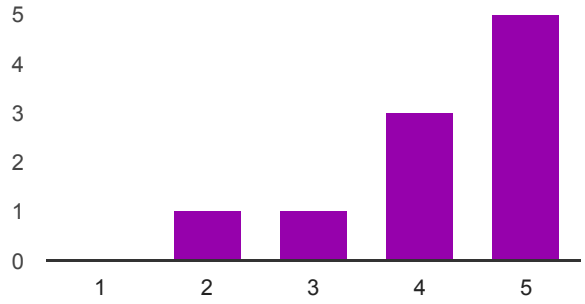
	3	**2**	20%
	4	**3**	30%
Strongly Agree:	5	**4**	40%

I found the "list of relevant documents" useful.



Strongly Agree:	1	**1**	10%
	2	**2**	20%
	3	**4**	40%
	4	**3**	30%
Strongly Disagree:	5	**0**	0%

I found the relative score functionality useful.



Strongly disagree:	1	**0**	0%
	2	**1**	10%
	3	**1**	10%

	4	**3**	30%
Strongly Agree:	5	**5**	50%

What do you think about the metadata provided for a single result. Are the extract, title, category, and URL sufficient or further data is needed? Give examples of additional data if you believe such one is necessary?

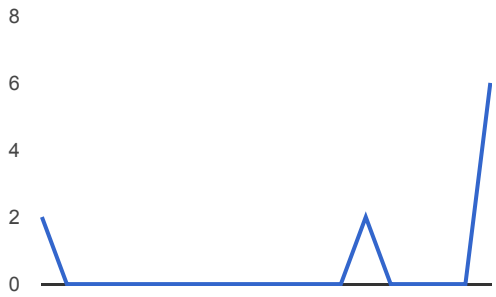
date, source of result

Pictures

Is it possible to belong to several categories ?

would be beneficial if the list of relevant documents had some additional features such an opportunity to add notes or to save it.

Number of daily responses



Bibliography

[1] G. Amati. Probabilistic models for information retrieval based on divergence from randomness. *PhD thesis - University of Glasgow*, 2003.

[2] Ricardo Baeza and Berthier Ribeiro. Modern information retrieval. *ACM Press New York*, pages 269–280, 1999.

[3] Boris Beizer. Software system testing and quality assurance. *Van Nostrand Reinhold Co*, 1984.

[4] Belkin, Nicholas J., Robert N. Oddy, , and Helen M. Brooks. Ask for information retrieval: Part i. background and theory. *Journal of documentation*, 1982.

[5] Bhat, H. S., and Kumar. On the derivation of the bayesian information criterion. 2010.

[6] John Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 1996.

[7] Gao Hong Katarzyna Bryc and Carlos D. Bustamante. On identifying the optimal number of population clusters via the deviance information criterion. *PloS one 6.6*, 2011.

[8] Calder, Bobby J., Lynn W. Phillips, and Alice M. Tybout. The concept of external validity. *Journal of Consumer Research*, 1982.

[9] CESerchEngine. Home page. `http://cesearchengine.com/`, 2017. [Online; accessed 28-Jan-2017].

[10] Douglass R. Cutting, Jan O. Pedersen, David Karger, and John W.Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. *In Proc. of the 15th Aunnual Internation ACM/SIGIR Conference*, pages 318–329, 1992.

[11] Carmel David. Static index pruning for information retrieval systems. *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval. ACM*, 2001.

[12] Jersey Official Documentation. ClientBuilder documentation. `https://jersey.java.net/apidocs/2.22/jersey/javax/ws/rs/client/ClientBuilder.html`, 2016. [Online; accessed 5-March-2017].

[13] Brown Peter F. Class-based n-gram models of natural language. *Computational linguistics 18.4*, pages 467–479, 1992.

[14] Gallaugher, John M., Ramanathan, and Suresh C. Choosing a client/server architecture. *Information Systems Management*, 13, 2007.

[15] Donna Harman. Information retrieval evaluation. *Synthesis Lectures on Information Concepts, Retrieval, and Services 3.2*, pages 1–119, 2011.

[16] Sarah Hatton. Choosing the right prioritisation method. *Univ. of Western Australia, Crawley*, 2008.

[17] ick Belew. Rave reviews: Acquiring relevance assessments from multiple users. *Working Notes of the AAAI Spring Symposium on Machine Learning in Information Access, Stanford, CA*, 1996.

[18] Jain, Nilesh, Ashok Bhansali, and Deepak Mehta. Angularjs: A modern mvc framework in javascript. *Journal of Global Research in Computer Science 5.12*, pages 17–23, 2015.

[19] jQuery Official website. Working with JSONP. `https://learn.jquery.com/ajax/working-with-jsonp/`, 2016. [Online; accessed 28-Jan-2017].

[20] Kaur, Maninder, Nitin Bhatia, and Sawtantar Singh. Web search engines evaluation based on features and end-user experience. *International Journal of Enterprise Computing and Business Systems*, 2011.

[21] Kodinariya, Trupti M., and Prashant R. Makwana. Review on determining number of cluster in k-means clustering. *International Journal 1.6*, 2013.

[22] Liang, Shangsong, Zhaochun Ren, and Maarten De Rijke. Fusion helps diversification. *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, 2014.

[23] Liu and Bing. User personal evaluation of search engines. *Department of Computer Science - University of Illinois at Chicago*, 2012.

[24] Cheng-Jye Luh, Sheng-An Yang, and Ting-Li Dean Huang. Estimating google's search engine ranking function from a search engine optimization perspective. *Online Information Review*, 40:239–255, 2016.

[25] M. Machill and M. Beiler. The importance of the internet for journalistic research. *Journalism Studies*, 10:178–203, 2009.

[26] Stephann Makri. Investigating the information-seeking behaviour of academic lawyers: From elliss model to design. *Information Processing and Management*, 44:613–634, 2008.

[27] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. Introduction to information retrieval. *Cambridge University Press*, 2008.

[28] Gary Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM 49.4*, pages 41–46, 2006.

[29] Mikowski, Michael S., and Josh C. Powell. Single page web applications. *B and W*, 2013.

[30] msdn.microsoft official website. Integration Testing. `https://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx`, 2016. [Online; accessed 5-March-2017].

[31] JUnit official website. JUnit library. `http://junit.org/junit4/`, 2016. [Online; accessed 5-March-2017].

[32] Lingpipe official website. CompleteLinkClusterer documentation. `http://alias-i.com/lingpipe/docs/api/com/aliasi/cluster/CompleteLinkClusterer.html`, 2016. [Online; accessed 5-March-2017].

[33] Lingpipe official website. KMeans Clusterer documentation. `http://alias-i.com/lingpipe/docs/api/com/aliasi/cluster/KMeansClusterer.html`, 2016. [Online; accessed 5-March-2017].

[34] Skyscanner official website. Home page. `https://www.skyscanner.com/`, 2016. [Online; accessed 5-March-2017].

[35] Iadh Ounis. Architecture of retrieval systems. *University of Glasgow - IR4*, 2017.

[36] Jackson Home page. Overview. `https://github.com/FasterXML/jackson`, 2016. [Online; accessed 28-Jan-2017].

[37] Pelleg, Dan, and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. *ICML*, 2001.

[38] Alexandrin Popescul and Lyle H. Ungar. Automatic labeling of document clusters. *University of Pennsylvania*.

[39] Christian Posta. Microservices for java developers. *OReilly*, 2016.

[40] Radlinski, F., Dumais, and S. Improving personalized web search using result diversification. *SIGIR*, pages 691–692, 2006.

[41] Van Rijsbergen. Information retrieval (second edition). *Butterworths, London*, 1979.

[42] Alex Rodriguez. Restful web services: The basics. *IBM developerWorks*, 2008.

[43] Mark Sanderson and Bruce Croft. Deriving concept hierarchies from text. *Department of Information Studies University of Sheffield*, 199.

[44] Rodrygo L.T. Santos, Jie Peng, Craig Macdonald, and Iadh Ounis. Explicit search result diversification through sub-queries. *Department of Computing Science University of Glasgow*.

[45] Hu Shuhua. Akaike information criterion. *Center for Research in Scientific Computation*, 2007.

[46] Louise T. Su. Evaluation measures for interactive information retrieval. *Information Processing & Management*, 1992.

[47] AngularJS Official website. Data Binding. `https://docs.angularjs.org/guide/databinding`, 2016. [Online; accessed 28-Jan-2017].

[48] AngularJS Official website. Developer Guide / Directives. `https://docs.angularjs.org/guide/directive`, 2016. [Online; accessed 28-Jan-2017].

[49] AngularJS Official website. Home Page. `https://angularjs.org/`, 2016. [Online; accessed 28-Jan-2017].

[50] AngularJS Official website. $http Service. `https://docs.angularjs.org/api/ng/service/\$http`, 2016. [Online; accessed 28-Jan-2017].

[51] AngularJS Official website. ngClick Documentation. `https://docs.angularjs.org/api/ng/directive/ngClick`, 2016. [Online; accessed 28-Jan-2017].

[52] AngularJS Official website. $rootScope.Scope. `https://docs.angularjs.org/api/ng/type/$rootScope.Scope`, 2016. [Online; accessed 28-Jan-2017].

[53] AngularJS Official website. $route service. `https://docs.angularjs.org/api/ngRoute/service/$route`, 2016. [Online; accessed 28-Jan-2017].

[54] AngularJS Official website. Scopes and digest cycles. `https://docs.angularjs.org/guide/scope`, 2016. [Online; accessed 28-Jan-2017].

[55] AngularJS Official website. Services. `https://docs.angularjs.org/guide/services`, 2016. [Online; accessed 28-Jan-2017].

[56] Bootstrap Official website. Home Page. `http://getbootstrap.com/`, 2016. [Online; accessed 28-Jan-2017].

[57] Dzone Official website. 7 best java frameworks for 2016. `https://dzone.com/articles/7-best-java-frameworks-for-2016`, 2016. [Online; accessed 28-Jan-2017].

[58] Graphviz Graph Visualization Software Official website. Drawing graphs with dot. http://www.graphviz.org/pdf/dotguide.pdf, 2017. [Online; accessed 27-Feb-2017].

[59] Graphviz Graph Visualization Software Official website. Home Page. http://www.graphviz.org/Home.php, 2017. [Online; accessed 27-Feb-2017].

[60] Graphviz Graph Visualization Software Official website. Output Formats. http://www.graphviz.org/content/output-formats\#dgv, 2017. [Online; accessed 27-Feb-2017].

[61] InfoWorld Official website. Is AngularJS ready for the enterprise? http://www.infoworld.com/article/2890272/javascript/is-angularjs-ready-for-the-enterprise.html, 2016. [Online; accessed 28-Jan-2017].

[62] Jersey Official website. User Guide. https://jersey.java.net/documentation/latest/user-guide.html, 2016. [Online; accessed 28-Jan-2017].

[63] LingPipe Official website. Cluster Tutorial Section. http://alias-i.com/lingpipe/demos/tutorial/cluster/read-me.html, 2017. [Online; accessed 28-Jan-2017].

[64] Martin Fowler Official website. DIP in the Wild. https://martinfowler.com/articles/dipInTheWild.html, 2016. [Online; accessed 28-Jan-2017].

[65] Martin Fowler Official website. GUI Architectures. https://martinfowler.com/eaaDev/uiArchs.html, 2016. [Online; accessed 28-Jan-2017].

[66] Martin Fowler Official website. Inversion of Control Containers and the Dependency Injection pattern. https://martinfowler.com/articles/injection.html, 2016. [Online; accessed 28-Jan-2017].

[67] Martin Fowler Official website. POJO. https://www.martinfowler.com/bliki/POJO.html, 2016. [Online; accessed 28-Jan-2017].

[68] Martin Fowler Official website. UnitTesting. https://martinfowler.com/bliki/UnitTest.html, 2016. [Online; accessed 5-March-2017].

[69] Mozilla Developer Network Official website. XMLHttpRequest. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest, 2016. [Online; accessed 28-Jan-2017].

[70] Terrier Official website. DFRBagExpansionTerms class documentation. http://terrier.org/docs/v4.0/javadoc/org/terrier/querying/DFRBagExpansionTerms.html, 2016. [Online; accessed 28-Jan-2017].

[71] Terrier Official website. Indexing documentation. http://terrier.org/docs/v4.1/configure_indexing.html, 2016. [Online; accessed 28-Jan-2017].

[72] Terrier Official website. Manager class documentation. http://terrier.org/docs/v4.1/javadoc/org/terrier/querying/Manager.html, 2016. [Online; accessed 28-Jan-2017].

[73] Terrier Official website. Overview. http://terrier.org/docs/v4.1/overview.html, 2016. [Online; accessed 28-Jan-2017].

[74] Terrier Official website. QueryExpansion class documentation. http://terrier.org/docs/v3.5/javadoc/org/terrier/querying/QueryExpansion.html, 2016. [Online; accessed 28-Jan-2017].

[75] Terrier Official website. SearchRequest Interface. http://terrier.org/docs/v4.1/javadoc/org/terrier/querying/SearchRequest.html, 2016. [Online; accessed 28-Jan-2017].

[76] Terrier Official website. Quickstart Guide: Using Terrier for Experiments. http://terrier.org/docs/v4.2/quickstart_experiments.html, 2017. [Online; accessed 25-Feb-2017].

[77] TheServerSide Official website. Five drawbacks to choosing JSF as your web application framework. http://www.theserverside.com/feature/Five-drawbacks-to-choosing-JSF-as-your-web-application-framework, 2016. [Online; accessed 28-Jan-2017].

[78] TREC 2014 Web Track Guidelines Official website. TREC 2014 Web Track Guidelines. http://www-personal.umich.edu/~kevynct/trec-web-2014/, 2017. [Online; accessed 25-Feb-2017].

[79] Ryen W. White and Bill Kules. Supporting exploratory search, introduction. *Communications of the ACM*, 49:36–39, 2008.

[80] Ryen W. White and Resa A. Roth. Exploratory search: Beyond the query-response paradigm. *CA: Morgan and Claypool*, 2009.

[81] Zeng, H.J., He, Q.C., Chen, Z., Ma, W.Y., Ma, and J. Learning to cluster web search results. *SIGIR*, pages 210–217, 2004.

[82] Zheng, Bweijunl, Wei Zhang, and Xiaoyu Fu Boqin Feng. A survey of faceted search. *Journal of Web engineering*, 2013.