TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science
Interactive Media Lab Dresden

Master's Thesis

Towards Collaborative Session-based Semantic Search

Sebastian Straub

A thesis submitted in fulfillment of the requirements for the degree of Master of Science (M.Sc.)

Supervisor:	Prof. Dr.-Ing. Raimund Dachselt
Advisors:	Dr.-Ing. Annett Mitschick
	Dr.-Ing. Anke Lehmann
Date:	October 5, 2017

Declaration of Authorship

Abstract

Towards Collaborative Session-based Semantic Search

by Sebastian Straub

In recent years, the most popular web search engines have excelled in their ability to answer short queries that require clear, localized and personalized answers. When it comes to complex exploratory search tasks however, the main challenge for the searcher remains the same as back in the 1990s: Trying to formulate a single query that contains all the right keywords to produce at least some relevant results.

In this work we want to investigate new ways to facilitate exploratory search by making use of context information from the user's entire search process. Therefore we present the concept of session-based semantic search, with an optional extension to collaborative search scenarios. To improve the relevance of search results we expand queries with terms from the user's recent query history in the same search context (session-based search). We introduce a novel method for query classification based on statistical topic models which allows us to track the most important topics in a search session so that we can suggest relevant documents that could not be found through keyword matching.

To demonstrate the potential of these concepts, we have built the prototype of a session-based semantic search engine which we release as free and open source software. In a qualitative user study that we have conducted, this prototype has shown promising results and was well-received by the participants.

Zusammenfassung

Towards Collaborative Session-based Semantic Search

von Sebastian Straub

Die führenden Web-Suchmaschinen haben sich in den letzten Jahren gegenseitig darin übertroffen, möglichst leicht verständliche, lokalisierte und personalisierte Antworten auf kurze Suchanfragen anzubieten. Bei komplexen explorativen Rechercheaufgaben hingegen ist die größte Herausforderung für den Nutzer immer noch die gleiche wie in den 1990er Jahren: Eine einzige Suchanfrage so zu formulieren, dass alle notwendigen Schlüsselwörter enthalten sind, um zumindest ein paar relevante Ergebnisse zu erhalten.

In der vorliegenden Arbeit sollen neue Methoden entwickelt werden, um die explorative Suche zu erleichtern, indem Kontextinformationen aus dem gesamten Suchprozess des Nutzers einbezogen werden. Daher stellen wir das Konzept der sitzungsbasierten semantischen Suche vor, mit einer optionalen Erweiterung auf kollaborative Suchszenarien. Um die Relevanz von Suchergebnissen zu steigern, werden Suchanfragen mit Begriffen aus den letzten Anfragen des Nutzers angereichert, die im selben Suchkontext gestellt wurden (sitzungsbasierte Suche). Außerdem wird ein neuartiger Ansatz zur Klassifizierung von Suchanfragen eingeführt, der auf statistischen Themenmodellen basiert und es uns ermöglicht, die wichtigsten Themen in einer Suchsitzung zu erkennen, um damit weitere relevante Dokumente vorzuschlagen, die nicht durch Keyword-Matching gefunden werden konnten.

Um das Potential dieser Konzepte zu demonstrieren, wurde im Rahmen dieser Arbeit der Prototyp einer sitzungsbasierten semantischen Suchmaschine entwickelt, den wir als freie Software veröffentlichen. In einer qualitativen Nutzerstudie hat dieser Prototyp vielversprechende Ergebnisse hervorgebracht und wurde von den Teilnehmern positiv aufgenommen.

Acknowledgements

First of all I would like to express my very great appreciation to my advisors Annett Mitschick and Anke Lehmann for their patient guidance and valuable feedback. Your continued support of my work was of vital importance to me.

I am grateful to Prof. Raimund Dachselt for the opportunity to write this thesis about a topic of my own choosing. I know that this is not a matter of course and I hope I didn't blow it for future students who come forward with ideas for their own theses.

My special gratitude goes out to my colleagues at Avantgarde Labs, whom I had the pleasure to work with during my entire MA studies. You provided me with a playground where I could try out anything that I have learned during my studies and so it is not an understatement when I say that this work wouldn't have been possible without that great experience.

And finally, I am particularly grateful to my parents, my brother, my sister and my dear friends here in Dresden and back home for their unfailing support and the highly appreciated distractions that kept me going whenever I was ready to scrap the whole thing. You guys are awesome!

Contents

Abstract. v

Zusammenfassung. vii

Acknowledgements. ix

1. Introduction 1

2. Related Work 5
- 2.1. Topic Models 5
  - 2.1.1. Common Traits 5
  - 2.1.2. Topic Modeling Techniques 6
  - 2.1.3. Topic Labeling 8
  - 2.1.4. Topic Graph Visualization 9
- 2.2. Session-based Search 10
- 2.3. Query Classification 12
- 2.4. Collaborative Search 13
  - 2.4.1. Aspects of Collaborative Search Systems 14
  - 2.4.2. Collaborative Information Retrieval Systems 14

3. Core Concepts 19
- 3.1. Session-based Search 19
  - 3.1.1. Session Data 20
  - 3.1.2. Query Aggregation 21
- 3.2. Topic Centroid 22
  - 3.2.1. Topic Identification 23
  - 3.2.2. Topic Shift 27
  - 3.2.3. Relevance Feedback 29
  - 3.2.4. Topic Graph Visualization 30
- 3.3. Search Strategy 32
  - 3.3.1. Prerequisites 33
  - 3.3.2. Search Algorithms 33
  - 3.3.3. Query Pipeline 35
- 3.4. Collaborative Search 38
  - 3.4.1. Shared Topic Centroid 38
  - 3.4.2. Group Management 39
  - 3.4.3. Collaboration 40
- 3.5. Discussion 42

4. Prototype 45
- 4.1. Document Collection 46
  - 4.1.1. Selection Criteria 46

4.1.2. Data Preparation . . . 46
4.1.3. Search Index . . . 50
4.2. Search Engine . . . 52
4.2.1. Search Algorithms . . . 53
4.2.2. Query Pipeline . . . 55
4.2.3. Session Persistence . . . 58
4.3. User Interface . . . 59
4.4. Performance Review . . . 64
4.5. Discussion . . . 65

5. User Study **67**
5.1. Methods . . . 67
5.1.1. Procedure . . . 68
5.1.2. Implementation . . . 68
5.1.3. Tasks . . . 70
5.1.4. Questionnaires . . . 70
5.2. Results . . . 71
5.2.1. Participants . . . 71
5.2.2. Task Review . . . 72
5.2.3. Literature Research Results . . . 75
5.3. Discussion . . . 76

6. Conclusion **79**

Bibliography **83**
Weblinks . . . 90

A. Appendix **91**
A.1. Prototype: Source Code . . . 91
A.2. Survey . . . 92
A.2.1. Tasks . . . 92
A.2.2. Document Filter for Google Scholar . . . 95
A.2.3. Questionnaires . . . 96
A.2.4. Participant's Answers . . . 100
A.2.5. Participant's Search Results . . . 102

Chapter 1

Introduction

"And how will you inquire into a thing when you are wholly ignorant of what it is? Even if you happen to bump right into it, how will you know it is the thing you didn't know?"

— *Plátōn, Ménōn*

This quote from a fictional dialogue between ancient Greek philosopher Socrates and the politician Meno which was written by Plato has become known as *Meno's Paradox* or *The Paradox of Inquiry*: At its core, it states that there is no need to search for something we already know and there is no way to recognize anything that we don't know, therefore acquiring knowledge should be impossible. For a theist this problem is solved by referring to a higher power which has knowledge of *the* absolute truth and by following the rules of the higher power, even mere mortals can recognize this truth and therefore gain knowledge. Atheists however can only deduce knowledge from what they themselves define as axioms, the suitability of which they can assess by observation of the universe, but in the end there is no way to prove even the most basic axioms due to the lack of an absolute reference frame.

From a practical perspective, the implications of Meno's paradox are of little relevance to us: Humans have very different and often contradictory assumptions about the world, but in most cases these assumptions still allow them to get through their daily lives without getting into too much trouble with their environment. However, the issue of acquiring knowledge about things we don't know anything about is a very practical one which we constantly encounter.

Searching for information has become much more efficient and convenient with the rise of web search engines that allow access to knowledge sources that go far beyond what libraries traditionally used to offer. The popular search engines go through great lengths to offer search results that are localized and even personalized, like a search for "movies" might return a list of movies which are currently running in nearby theaters and which fit your cinematic profile. Yet while these retrieval methods do a great job when we have a clear goal, looking for information about topics we are already well aware of, most search engines fail to support the users in acquiring new knowledge in fields that they are unfamiliar with. This leads to the question how we can support people in exploratory search tasks when the goal is blurry at best and it is unclear how relevant sources that contain the desired knowledge can be discovered.

To address this issue, we present the concept of a session-based semantic search engine. Session-based search implies that we interpret each user's queries in the context of their previous queries from the same search session, which includes all requests that are submitted in order to fulfill a single information need. The designation *semantic* refers to the capability of the system to find relevant search results not just based on the keywords that are part of any query in a session but on other contextual information that has been explicitly or implicitly provided by the user.

So instead of compelling the user to refine a query until the search engine gets all the data it needs in a single request, we save all relevant context information in a temporary search session and interpret subsequent queries under that prior, without the need to build up a user profile first. This way, we can immediately improve the ranking of search results and let the search engine guide the user by suggesting topics that are relevant in the current context.

We are going to use probabilistic topic models to uncover the hidden thematic structure in arbitrary document collections. A topic model allows us to discover the dominant topics of documents that are reviewed by the user in the course of a search session, which we will harness to improve the ranking of search results in subsequent queries and to suggest documents that were previously not considered based on the query terms that the user has submitted. This is an important step to support the user in acquiring knowledge while not being fully aware of the relevant terminology in a field. Through probabilistic inference we can model the most important topics in a search session and recommend documents about topics that are likely to be relevant in the current search context but which the user possibly has not considered before.

In order to enable groups of people to efficiently work on the same research task, we extend this concept to the collaborative setting. Our goal is to create a system that encourages collaborative work by improving the ranking and presentation of search results based on shared context information that has been provided by all group members. While we are going to offer additional functionality that can make use of the shared search context, we will not define a comprehensive alternative to existing collaborative search systems. Instead, the methods we propose are intended as an extension to improve the efficiency of collaborative search in various situations.

To put these concepts to the test, we present the prototype of a session-based semantic search engine which we publish together with this work as free software. This is, to the best of our knowledge, the first open source release of a search engine of this kind. It implements search sessions which serve as a wrapper for all context information provided by the user and allows to retrieve documents using a combined search algorithm based on the user's previous queries in the session as well as a model of the most relevant topics in the current search context. While the prototype is limited to the single-user setting, it has been designed with a possible extension to collaborative search in mind.

In a small-scale, qualitative user study we investigate whether the semantic capabilities of the prototype are beneficial when compared to conventional academic search engines. To this end we let participants perform several literature research tasks using either the prototype or a popular commercial academic search engine. We will compare the search results of the participants as well as their assessment of the usability of both systems to get an impression of the prototype's fitness for exploratory search. To ensure that the search results are comparable, we rely on a full-text index of 1.2 million scientific papers from the *arXiv*, a publicly accessible repository of scientific papers in various disciplines.

The rest of this thesis is organized as follows: We give an overview of related research concerning statistical topic models, session-based search, query classification and collaborative search. In Chapter 3 we present the core concepts that define a session-based semantic search engine in the single-user and the collaborative setting. These include the foundations of session-based search, the topic centroid as a data structure that tracks the most relevant topics in a search session, a comprehensive search strategy that harnesses the context information in a search session and an extension of these concepts to the collaborative setting. Next we introduce the prototype that was built as part of this work: We describe the process that led to the construction of the search index, give an overview of the implementation of the search engine's core functions, provide insights into some of the unique UI features and show the practicality of the prototype in a technical performance review. Finally, we present the results of the qualitative user study we conducted, in which we compare the prototype with a conventional literature search engine.

Chapter 2

Related Work

In this chapter we will give an overview of the related work concerning the main topics we are going to discuss in this work: First, we review statistical topic models, which enable us to uncover the hidden thematic structure in arbitrary document collections and consequently to recommend documents based on these topics instead of just the user's query terms. Next, we discuss session-based search, an information retrieval method that interprets user queries in the context of a search session. This inclusion of the search context relieves the user from the requirement to formulate a single query that addresses all aspects of a complex research task. With query classification, which is the issue of the following section, we can combine the context information from a search session with the knowledge about the topics in the document collection to reason about the searcher's actual intentions, which can help to further improve the ranking of search results. Finally, we review the literature on collaborative search, which will serve as a foundation for our concepts to influence search results based on contextual information that is implicitly shared between all members of a group.

2.1. Topic Models

An important prerequisite for efficient exploratory search is to have knowledge about the latent topics in the document collection which are not inherently visible through keyword matching. The goal is to not only serve documents whose content matches a query string, but to better understand the search interest of the user and to find documents that are most relevant to that search interest. This however requires a consistent annotation of documents based on their contents, which is a huge effort when done manually by humans. However, with statistical topic models it is possible to discover the abstract topics that occur in each document of a collection in an unsupervised learning process.

2.1.1. Common Traits

Topic modeling is the process of uncovering the thematic structure hidden inside a collection of unclassified documents. The methods for topic modeling that we are going to present share a number of assumptions about the data model and how topical information is extracted from that data. The data model consists of a collection of documents $d_1, \ldots, d_m$, where each document can be represented as a list of tokens or words $w_1, \ldots, w_n$. The unique tokens of all documents form the vocabulary V.

A *topic* is defined as a multinomial distribution over a fixed vocabulary. This implies that a topic can be represented as a vector whose length is equal to the size of the vocabulary, where each index position represents the weight of the word for this topic at the same position in the vocabulary. Intuitively, this means that a topic is identified by terms that co-occur more frequently in the documents of this topic than in others. This assumption has its root in the distributional hypothesis [Har54] which states that words that occur in the same context usually carry similar meanings.

Because word order is not relevant for the definition of a topic, each document can be represented using the more memory-efficient bag-of-words model, preserving only the term frequency, but not the order of occurrence. We can further reduce the size of the dictionary without sacrificing precision by removing certain terms. These include stop words and very frequent terms, which will likely occur in most new documents, as well as rare words that probably won't occur in new documents. Stemming or lemmatization can further reduce the vocabulary size by joining different grammatical inflections of the same words. This kind of dimensionality reduction not only enables us to calculate topic models for larger document collections in-memory, it also helps with generalization and reduces the probability of overfitting.

2.1.2. Topic Modeling Techniques

We will now show a selection of techniques for generating statistical topic models.

Latent Semantic Analysis (LSA)

One of the earliest techniques that aim to automatically identify topics in unstructured text is Latent Semantic Analysis (LSA), which was introduced by Deerwester et al. in 1990 [DDF$^+$90]. At its core, LSA is quite similar to contemporary topic models: it relies on the bag-of-words model and defines a topic as a distribution over a fixed vocabulary. The central data structure is the term-document matrix, where rows contain the terms of the vocabulary and columns correspond to documents. Each cell contains the frequency of a term in the corresponding document, or more typically the specific term weight, using *tf-idf* as weighting function.

Topics are discovered by reducing this matrix to a vector space of lower dimensionality, which the authors call the *latent semantic space*. This mapping is created by applying a Singlular Value Decomposition (SVD). A reasonable approximation is usually feasible, because the matrix tends to be sparsely populated, and it helps eliminating noisy data, therefore increasing the significance of individual entries. The rows of the decomposed matrix should now represent a set of linearly independent vectors that can be interpreted as concepts or topics, because terms that carry similar meaning should be mapped to the same direction in the latent semantic space. As a result of the SVD, we can record the weight of the influence of each term on each decomposed topic vector, which gives us insights into the meaning of each topic and allows us to classify new documents using these frequency distributions.

An issue of LSA used to be the high computational cost and the requirement to have the full term-document matrix in memory to efficiently perform the SVD. These limitations however have been overcome in recent years, when an online training algorithm for LSA that is capable of processing an unlimited number of documents with constant memory requirements was introduced by Řehůřek [Řeh11] in 2011 and subsequently implemented in the gensim library [@Řeh]. An intrinsic problem with LSA's approach of dimensionality reduction is the inability to handle polysemy (i.e. disambiguating words that have multiple meanings), due to the fact that topics are directly derived from combined term vectors and each word therefore must always be part of exactly one topic, which does not properly resemble the characteristics of natural language.

Probabilistic Latent Semantic Analysis (pLSA)

While in LSA, topics are directly derived from training data through dimensionality reduction, Probabilistic Latent Semantic Analysis (pLSA) [Hof99] relies on a probabilistic generative model which defines the process by which documents are generated. In this model, the goal is to discover the hidden variables that best explain the observed data, or those which minimize the gap between data generated by the model and the actual observations. The hidden variables can be interpreted as topics, which once again are distributions over words and once uncovered we can use these distributions to assign topics to previously unobserved documents.

In pLSA, the probability of observing a word w in a document d is defined as

$$P(d, w) = \sum_{z \in Z} P(z)\ P(d|z)\ P(w|z) = P(d) \sum_{z \in Z} P(z|d)\ P(w|z)$$

with Z being the set of possible unobserved topics, whose size must be defined in advance as a hyperparameter. This definition allows a document to be generated from several topics, which implies that a single document can be about multiple topics, and because a topic is defined as a distribution over words, pLSA can handle polysemy, unlike standard LSA. The model parameters are trained using an expectation–maximization algorithm which uses variant of simulated annealing in order to escape local maxima. In his work, Hofmann has empirically shown that pLSA is superior to standard LSA, both in terms of precision of the uncovered topics as well as computational performance of the algorithm.

Latent Dirichlet Allocation (LDA)

Blei et al. [BNJ03] criticized that pLSA is not a well-defined generative model, because it learns $P(z|d)$ only for the documents that were observed in the training set and there is no clear way to assign topics and their probabilities to unseen documents. To mitigate these issues, they propose Latent Dirichlet Allocation (LDA), a model which is a generalization of pLSA [GK03]. In the LDA model, each document is generated from a mixture of topics which has a sparse Dirichlet prior. If a uniform Dirichlet prior distribution is used, the LDA model is equivalent to pLSA. In order to estimate the model parameters, Gibbs sampling is commonly applied [Ble12]. While the originally proposed inference algorithm required the entire training set to be available ahead of time, Hoffman et al. [HBB10] developed an online algorithm that can be applied to document collections of arbitrary size.

Hierarchical Topic Models

All topic models we have reviewed so far share the property that they generate a fixed number of topics that have no specific relation to each other. It is not trivial to estimate how many topics are required to generate a reasonable topic model for an arbitrary document collection, which usually involves a costly process of trial-and-error. Furthermore, having a topic model with hierarchical relations between topics and sub-topics could help to organize a document collection on different levels of granularity.

The issue of finding the optimal number of topics is addressed by Teh et al. [TJB+05] with the Hierarchical Dirichlet Process (HDP), a generalization of the LDA model. Like with LDA, topics are represented as mixtures over a fixed vocabulary, but while in LDA documents are mixtures of a predefined number of topics, in HDP this number is generated in a Dirichlet process. The denomination "hierarchical" however refers to the additional layer that is added to the random processes and does not indicate, that the topics themselves have a hierarchical structure. A model that can generate topic models with a hierarchical structure is the nested Hierarchical Dirichlet Process (nHDP) [PWB+12], which is an extension of HDP that relies on techniques for building topic hierarchies based on the nested Chinese Restaurant Process (nCPR) [BGJ10].

2.1.3. Topic Labeling

While topic models can be very useful in identifying relations between documents based on the similarity of an underlying topic, they do not provide us with any means to actually define or name individual topics. Having a name for a topic can help in acquiring additional knowledge about that topic from other sources, which in turn can be used to improve search results. Furthermore it enables us to make judgments about the accuracy and quality of the topic model and it helps humans to understand the relationships between the entities of a domain. A search engine can additionally benefit from such a domain model by adequately visualizing important topics to the user, who can use this aid to navigate through topics and filter results.

Each topic in an generative model like LDA or HDP is represented by a vector that assigns a probability to each term that defines the topic. These marginal probabilities [BNJ03] can be used to find the most significant terms that identify a topic. With some background knowledge, it is often possible to derive a name for the underlying concept from these terms. This however requires manual review by humans with knowledge about the domain of interest, which can be an expensive task.

To mitigate this issue, research has been done in recent years on the issue of automatic topic labeling. The general approaches are to use the terms with the highest marginal probabilities for each topic to either derive labels from relevant terms and documents of that topic [MSZ07; BHX14] or to use external knowledge sources to map the discovered topic to a known concept [LNK+10; LGN+11; HHK+13]. The knowledge-free approach is very flexible and able to derive labels even for new topics that just came up in news articles or social media. The knowledge-backed approach on the other hand allows us to link topics with other knowledge sources, which helps us to map a topic model to existing ontologies and to gain further knowledge of that topic.

Still, the research so far has shown that fully automated topic labeling is not yet feasible, due to high error rates and low-quality labels. If meaningful topic labels are a requirement, the generated labels must be screened by human reviewers with sufficient domain knowledge.

2.1.4. Topic Graph Visualization

For the users of a search engine that ranks documents based on a selection of topics from a hierarchical topic model, it can be helpful to have some kind of visualization of the most relevant topics. It may also be interesting to investigate ways to interact with the topics, in order to adjust the selection of relevant topics and to explore new topics in the taxonomy. A particular challenge is the integration of a suitable visualization tool into a search result page.

Early work in the field of topic visualization has been done by Niwa et al. [NNI+97] in 1997, before the first statistical topic models have been described. Their goal was to build an "interactive guidance mechanism for document retrieval systems" which displays a "visualized map of topics at each stage of [the] retrieval process". Significant terminology was extracted through term frequency analysis from a list of search results and relations between terms are determined through co-occurrence analysis in the document collection. A similar approach was investigated by Joho et al. [JSB04], who built a user interface for interactive query expansion in an information retrieval scenario. While these works give an interesting insight into visualization techniques that do not rely on topic models, they are not directly applicable to our visualization problem.

TopicNets is a "system for interactive visual analysis of large document corpora" [GOB+12]. It relies on LDA topic models with topic labels assigned by human editors and provides tools for graph-based visualization and interaction. The main purpose of the system is visual exploration of documents and the analysis of thematically related document collections. Even though some of the visualization methods can be helpful for topic graph visualization, the scope of TopicNets is the exploration of entire document collections, rather than the visualization of topics in a small subset of the documents.

The issue of subgraph extraction is discussed by Dupont et al. [DCD+06]. They developed a novel method to extract nodes from a connected graph while keeping the relations between these nodes by adding more nodes and edges from the original graph that re-connect the nodes of the subgraph. While this algorithm works efficiently even for large graphs, the tree-like topology of a hierarchical topic model allows us to deploy much simpler algorithms, but it may come in handy when topic models with different topologies are used.

The research on the visualization of relevant topics in information retrieval has so far, to the best of our knowledge, not come up with a practicable solution to the issue, especially not when screen space is scarce.

2.2. Session-based Search

In this work we define session-based search as an information retrieval method that interprets user queries in the context of a search session, which consists of previously submitted queries, search results and user interactions with the search engine. A search session is limited to the queries that have been submitted in order to fulfill a specific information need; other than that, no long-term user profile information is gathered. This distinguishes session-based search from personalized search [PSC$^+$02; GCP03; TDH05], where user profiles are built from long-term search and browsing behavior in order to provide search results that more closely resemble the interests of the user. While this kind of search result personalization has been widely adopted by commercial search engines, it comes with notable privacy implications [STZ07] and fails to support the user in complex exploratory search scenarios [WR09], especially when they are not in line with previous browsing behavior.

Early work that considered the query history in a search session to improve retrieval performance was done by Shen and Zhai in 2003 [SZ03]. They have conducted an experiment using an annotated document collection [VH00] where search queries are expanded to include terms from previous queries in a search session. The result of this simple approach was that retrieval performance consistently increased in their setup. In a follow-up paper [SSZ04], the algorithm was extended to detect session boundaries in the query history and to include metadata from documents that have been reviewed by the user. The designation *session-based search* has been mentioned for the first time in this work, although different nomenclature for the concept has since been in use, like *session search* [LZD$^+$15] or *dynamic search* [ZLY15].

In the following years, researchers attempted to discover more relevant features that can be extracted from search sessions which can in turn be used to improve document retrieval. Shen, Tan and Zhai [STZ05; TSZ06] mainly considered the query history and clickthrough data (i.e. information about which documents on a result page have been reviewed by the user) in their work. They showed that clickthrough data of previous queries was especially useful when it comes to predicting the topic of a query.

A large-scale evaluation of the benefits of search session context information was conducted by White et al. [WBD10], who had access to search logs of Microsoft's web search engine Bing [@Mic], as well as browsing data collected through a browser plugin for more than 100,000 users. Their goal was to predict search interests of users based on their previous interactions with the search engine and pages they have visited in between. Visited web pages were classified in about 200 categories in a two-layer hierarchical model. The model they used was able to predict the searcher's intent (i.e. the categories of documents that the user was looking for) far more reliably when context information like the results of previous queries and the browsing behavior on result pages was included.

In the work of Daoud et al. [DTB$^+$09] the relevant topics of a search session were stored in a graph that resembles both the hierarchical structure of an underlying topic model, as well as relations between topics that cannot be expressed in a strict tree structure. Relevant topics are detected by analyzing the topics of search results that have been reviewed by the user. These topics are then used to influence the ranking of subsequent queries, whose results in turn influence the state of the graph.

While publications before 2010 were rather infrequent, the interest in session-based search gained traction when the first *Session Track* [KCC$^+$10] was held at the Text REtrieval Conference (TREC). Notable work has been done by Yang et al. [LZY14; LDY14; LZD$^+$15; GZY13] who model the search session as a Markov Decision Process with states, actions and rewards. The system is seen as a “dual-agent stochastic game to model the interactions between user and search engine” [LZY14] who have to cooperate in order to fulfill the user’s information need. Another contribution is the classification of existing design choices for session-based search into 12 categories based on the combination of several mutually exclusive approaches to model state, actions and rewards, as well as an experimental comparison of the different methods [LZD$^+$15].

Guan and Yang [GY14a; GY14b] have investigated different techniques to expand queries with terms from the query history, which they called *query aggregation*. The core contribution of this work is an experimental analysis of the optimal weights for queries in a search session based on their position in the query history. Their findings show (to little surprise) that the last query is the most important one, but also that relevance does not strictly decrease for older queries. Instead, the relevance of the first query seems to remain rather constant on a level that is comparable to the second-to-last query, which should be considered when terms from the query history are added to an expanded search query.

The works of Jiang et al. [JHA14; JHS$^+$15] provide insights into user interaction with the search engine and overall satisfaction in the course of a search session. They have identified four kinds of search tasks which resulted in different behavior when it comes to the number of search results that are reviewed, the length of the search session and the time that users take to review documents. Being aware of these differences can help to design search algorithms and user interfaces that work best in each of the specified search scenarios.

After the last Session Track in 2014 [CKH$^+$14], the focus of some of the previous organizers at the Text REtrieval Conference has shifted towards *dynamic search*, which is the topic of the *Dynamic Domain Track* [YS16] that is held at TREC since 2015. Dynamic search, as discussed in this conference, is much more restrictively defined than session-based search: The target document collection comes from a specific domain of interest for which some kind of topic model is already available. Users only submit a single query and then give feedback to the search engine by selecting portions of relevant documents, whereupon the search engine adjusts the results that are displayed based on this feedback. The proceedings of this conference are not as relevant to this work as the now discontinued Session Tracks, but they might still provide some valuable insights.

The research on session-based search so far has shown that there is a large variety of methods available to model search sessions and to improve search result ranking using contextual information. Several methods have been implemented and when applied to query logs of reference data sets like those of the TREC session tracks, it was consistently shown that the precision of search results increased. However, all of these evaluations were based on query logs that were generated from interactions with a regular search engine that does not consider any context information and hardly any of the specified algorithms were incorporated into a working session-based search engine.

To the best of our knowledge, only two session-based search engines which actually rank documents in an interactive session based on previous user input have been built by now: Querium [GD11] and Dumpling [ZLY15]. The authors of Querium did not provide much detail about the search algorithms they have used, the service was never published, no source code has been released to date and no evaluation of the search performance was conducted, which makes it impossible to make assumptions about the performance of the system. Dumpling was developed as a research tool for government agencies to help finding information in the so called "dark web". It implements the Query Change Retrieval Model [GZY13] and the Win-win search algorithm [LZY14]. The search engine features a side-by-side view of search results that were retrieved using just the latest query (left) and the session-based search algorithms (right). The service was online for some time between 2015 and 2016 [@ZLY], but not usable by the public "due to the sensitivity of the data" [ZLY15]. As no source code has been published, this implementation also appears to be unavailable for review.

Our investigation indicates that while there is a variety of algorithms available for session-based search, no implementation of a session-based search engine has been published to date and no user studies have been conducted with data that was gathered from real users of a session-based search engine. The research so far has shown that methods which include the session context are superior to traditional retrieval methods, but due to the lack of an implementation there is no data available that could suggest whether session-based search can actually help users to find more relevant documents with fewer query reformulations.

2.3. Query Classification

A search engine that aims to support its users in exploratory search tasks should be able to find relevant documents for a search query, even if some of these documents do not contain the keywords of that query, as long as they are semantically related. Various approaches have been published under names like *query classification* [SSY+06] or *classification-enhanced ranking* [BSD10] that aim to derive the searcher's intent based on the provided query terms, which in turn can be used to improve the ranking of documents or to discover new documents that previously haven't been selected.

Dumais et al. [DCC01] were among the first to classify documents in order to improve the ranking of results in a web search engine. They used a combination of tags assigned by human editors and automatic text classification to assign class labels to websites ahead of time. These labels were used to generate different visualizations of the search result page. In a user study, they showed that the average time for the completion of a search task was minimal, when the search results were grouped by the category they belonged to, compared to other visualizations where labels were displayed next to each document or where no labels where displayed at all.

A different approach was taken by Shen et al. [SSY+06] who did not classify all documents in advance, which can be very expensive for web-scale indices. Instead, they defined a taxonomy and collected several sample documents for each class in the taxonomy. Then a text classifier was trained on these samples, so it could assign arbitrary text snippets to one or more of the categories in the taxonomy. When a user submits a query, a list of search results is generated using conventional information retrieval methods. The text contents of each search result are fed into the text classifier and the majority of the thereby discovered class labels is defined as the class of the query. This approach of using a predefined taxonomy and a text classifier to determine the query class based on regular full-text search results for the query was shown to work reliably at scale even for large taxonomies with about 6000 classes [BFG+07].

Query classification alone can be helpful to provide additional services like ads or context information, but the relevance of search results can hardly be improved without being able to search for semantically tagged documents in the index. Still, due to the high cost of classifying huge quantities of documents, some improvement can already be achieved through ad-hoc classification of search results which in turn can be used to re-rank the result list after query classification has been applied, as Bennett et al. [BSD10] have shown.

All classification techniques described so far were designed to find the best possible result for a single search query. However, in session-based search more context information is available to classify queries and the query classification results of previous queries can also help to understand the searcher's intent. We had already mentioned the work of of Daoud et al. [DTB+09] in the previous section, who tracked the relevant topics of a search session in a graph that was adjusted with each submitted search query. They avoided the issue of classifying search results by relying on human-labeled data provided by TREC for their experiments, but the ranking algorithm based on the graph of relevant topics was shown to improve the precision of their selection of search results compared to a baseline search algorithm that did not use the topic graph. A similar approach to context-aware query classification has been suggested by Cao et al. [CHS+09].

Research so far has shown that query classification is possible and relatively reliable, even for short queries with little context information and for large taxonomies with thousands of classes. The common approach is to use the topics of a query's search results to classify the query itself, although there are different techniques to achieve this classification. Not much research has been done so far when it comes to query classification in the context of a search session. Furthermore, the classification results in the mentioned works have been used to improve the ranking of search results that were retrieved with traditional information retrieval methods, but not to find new documents that could be matched based on their semantic features.

2.4. Collaborative Search

Collaborative information seeking and retrieval is a research field that concerns the study of "systems and practices that enable individuals to collaborate during the seeking, searching, and retrieval of information" [Fos06]. There is no consensus about the terminology that is commonly used in the field, as well as the exact definitions that distinguish them. Common names include *collaborative information retrieval* (CIR) [HJ05; Fos06], *collaborative information seeking* (CIS) [Sha10a] and *collaborative exploratory search* [PGS+08]. In this work, we use the phrase *collaborative search* to describe the participation of multiple users at the same time in a search session with shared context information.

Morris [Mor08; Mor13] suggests that despite the fact that collaborative search tools do not have any notable market share, people are already searching collaboratively on a regular basis with the help of communication tools like chat applications or social media. In this section we give an overview of common properties of systems that encourage collaboration and we will review several tools and different approaches to facilitate collaborative search.

2.4.1. Aspects of Collaborative Search Systems

According to Rodden [Rod91], collaboration systems can be classified by location and time: The members of a collaborating group can either be co-located or use some application to work remotely on the same task. The interaction between participants can be synchronous (i.e. in real-time) or asynchronous. Each of these properties has a major influence on the design of the tools that aim to support collaboration. Furthermore, we can classify collaborative search systems according to group size. For large groups of users, collaborative web search systems that benefit from the choices of the masses are conceivable [SBB$^+$03], while in small groups the goal is to facilitate collaboration and sharing of information between all group members.

According to Shah [Sha10b], a recurring topic in research on collaboration that is also relevant for CIR systems is the triple of control, communication and awareness. Control concerns the ability to manage and execute tasks in a way that participants do not block each other but instead are supported by the system. This includes the capability for structured message passing between group members and controlled access to shared resources. Communication is key to successful collaborative work; while in a co-located work scenario, it is sufficient to design the system in a way that it does not prevent frequent communication between participants, the main challenge in a remote collaboration task is to enable the participants to reach a sufficient level of communication in order to to efficiently solve the task. This can be achieved by integrating instant messaging or audio chat services into the application. The third aspect, awareness, concerns the activity of all participants of the collaborative task: being aware of what others in the group are currently working on, which topics they research and what they have discovered about certain aspects of the task can have a positive impact on the work of other group members. A system that facilitates sharing of this kind of information without additional effort for the users encourages collaboration during the entire research process instead of just division of labor.

2.4.2. Collaborative Information Retrieval Systems

One of the first tools to enable asynchronous collaboration in a search scenario is the *Ariadne* system [TN96] from 1996. It visualizes the search process of a single user on a timeline which contains the user's queries as well as the documents that have been reviewed for each query. Comments can be added to documents and the entire search log can be shared with other users. The system does not allow for real-time collaboration and the search context has no influence on search results, but the ability to share search logs and search results with other users of the software was a first step towards collaborative search.

Community-based Web Search

In the early 2000s the concept of a search engine was introduced that relies on asynchronous collaboration of a larger community to improve the ranking of search results or to otherwise improve the search experience. An example of this is the *Community Search Assistant* [Gla01], a meta search engine which recommends queries that are related to what the user is currently looking for based on the submitted queries of other search engine users. The relatedness of two queries is defined by a metric which depends on the number of matching documents among the top n search results for both queries. By keeping an index of past queries and URLs of the top n documents for each query, related queries can be efficiently retrieved.

Another community-based search engine is *I-SPY* [SFC+04]. Like the Community Search Assistant, it is a meta search engine that analyzes the search behavior of its community of users, but the goal of I-SPY is to directly recommend more relevant search results based on the search results selected by the community for similar queries. The search engine evaluates the session logs of its users to identify the search results which were most relevant to the user for a given query. When applied to all users of the search engine, popular results can be identified for frequent queries and I-SPY can in turn promote these results when the same query is submitted in the future. In a study, Smyth et al. [SBB+05] showed that the average relevance of search results with community-based promotions was higher than the unchanged results of the underlying search engine.

However, the community-based search approach comes with a number of drawbacks: The system only works with a sufficiently large user base whose session logs can be analyzed by the search engine. Even when a large community is available, recommendations will only work for frequent queries which were already submitted by several other users, which implies that the system won't work for rare queries, which are usually the hardest to answer. Furthermore, the recommendations are worthless, if the user's intent is not in line with the majority of the search community. While the specific meta search engines we mentioned here have not prevailed, the idea to use large query logs and click-through analysis [Joa02] to improve search result ranking has caught on and is in use at every major web search engine today. So it appears that implicit collaborative search on a world-wide scale is the norm today, but there seems to be no benefit in explicitly showing how this kind of feedback contributed to the ranking of search results.

Co-located Collaboration

The requirements to make collaborative search work for small groups in real-time are very different to those of an asynchronous community-based search engine. The authors of *Web Glance* [PAB+04] tried to improve collaboration in a small group of co-located participants. In contrast to the remote scenario, where communication and sharing of contents is the main concern, this is hardly an issue for participants that can talk to each other and present ideas to the group. The idea behind Web Glance is that multiple users should have access to a shared display using their own private devices. Participants can use their own device whenever they want, but access to the shared display is managed by Web Glance so that each user can share information with the group without the need to take exclusive control of the display.

CoSearch [AM08] can be seen as the spiritual successor of Web Glance with a stronger focus on search scenarios. The setup is similar to the one used by Web Glance: A big shared display that is visible to all participants and personal mobile devices. In addition, multiple input devices have been added to give more than one user the ability to directly control the contents of the shared screen. CoSearch brings direct integration of a search engine, to which queries can be submitted through the mobile devices of each participant. With the connected input devices, several users can review results and take notes. In a user study, Amershi and Morris show that this setup eases the division of labor in the group while still preserving the high level of communication and direct collaboration between co-located participants.

Synchronous Remote Collaboration

When it comes to real-time collaboration of remotely connected participants, different challenges must be considered. An early concept in this field is *CoVitesse* [LN02], a groupware application that facilitates collaborative web navigation for users that work from different locations on the same task. It enables small groups to share web contents and to send text messages to the other participants. While CoVitesse does not have a particular focus on search, its core functions make it suitable for collaborative research tasks.

SearchTogether [MH07] is a system that facilitates synchronous collaborative search for groups of remotely operating users by making the search engine the core component of the application. Whenever a user submits a query, it is added to a query log that is visible to all other users, from which they can review the results of that query with a single click. Search results can be shared and users can add rate and annotate these documents. The research process can be coordinated over an integrated messaging client.

This concept has been refined by the authors of *Coagmento* [Sha10a], which has seen continuous development since 2010, with the last major update in 2016 [MS16]. As opposed to the other groupware solutions we have reviewed so far, Coagmento is not designed as a standalone desktop application, but as a combination of browser extension and collaborative web platform. The browser extension comes with all the features of SearchTogether, like the ability to bookmark and annotate pages, to replay queries of other users and to send text messages to collaborators. The web platform *CSpace* allows users to manage their projects and to work collaboratively on the data they have collected with the help of the browser extension. This setup allows the group members to do collaborative work without leaving their web browser, which probably makes it the solution with the best integrated workflow for real-time remote collaboration so far.

Review

We discussed a variety of systems that aim to facilitate collaboration in different scenarios. In community-based web search engines, users collaborate asynchronously to improve the search experience for other users with similar queries. The real-time collaboration systems we have reviewed focus on small groups and aim to support the participants in their research tasks by providing the necessary tools to effectively work either remotely or in the same location. What we have not seen so far is a system that improves the ranking of search results based on context information of users collaborating in real-time. Systems like SearchTogether and CoSearch provide the necessary tools to efficiently manage search tasks in groups, but the ranking of search results is in no way influenced by the behavior of group members. Community-based web search engines like I-SPY do adjust the ranking of results, but only based on large-scale query analysis, which is not suitable to generate more relevant results for project-oriented collaboration scenarios in relatively small groups. In Section 3.4 we will discuss ways to improve the ranking of search results based on implicit context information by a group of collaborators.

Chapter 3

Core Concepts

In this chapter we will lay the foundations for a session-based semantic search engine which is able to support a single user or a group of users working collaboratively on complex exploratory search tasks that cannot be answered with a single query. The concepts we are going to present will not require access to large query logs or extensive user profiles, which are often not available for academic search engines and other IR systems with a relatively small or only sporadically recurring user base like that of many libraries. Instead, the user's queries are interpreted in the context of a search session and the ranking of search results is improved based on previous queries as well as an abstract concept of the session's core topics which we call the *topic centroid*. This approach also helps to avoid the serious privacy implications that come with the collecting of long-term user profiles in contemporary web search engines for the sake of search result personalization and increased revenue from personalized advertising.

The first section is dedicated to the concept of session-based search, which aims to improve the relevance of search results by expanding the user's search query with context information that was collected in the course of a search session.

Next we will introduce the afore-mentioned topic centroid, a data structure that holds the most relevant topics in a search session and which is implicitly built from data that is collected in the course of a search session. The topics are derived from a probabilistic topic model that is generated from the entire document collection.

With these concepts in mind we describe the relevant search strategies for a session-based semantic search engine: A full-text search algorithm that includes terms from the query history with varying weights in combination with a topic search algorithm that relies on the topic centroid to retrieve relevant documents. We embed these algorithms in a query pipeline that defines the information retrieval process from query submission to the generated search result page.

Finally, we extend these concepts to the collaborative setting, with the main goal of improving the relevance of search results based on context information that can be derived from the actions of a group with a common search interest.

3.1. Session-based Search

In this work, a search session describes all interactions of a user with a search engine that are required to fulfill a single information need. This may be a simple question that can be answered with a single query (e.g. getting a weather forecast), or a complex task that requires careful review of the search results and a constant refinement of the search query (like planning a holiday).

Contemporary web search engines improve their search results quite successfully by analyzing large query logs and identifying past search sessions with the goal to predict the most likely information need from an otherwise vague or ambiguous query [Joa02]. The result of this analysis can be used to suggest terms while users are typing in their query or to expand the submitted user query to include more terms that the user likely would have chosen in the next query anyway. A dependable approach to identify and harness search sessions identified in query logs is the query-flow graph [BBC+08]. While large query logs can be a valuable resource for a search engine, there are often no query logs of sufficient size available for many search applications, usually due to a relatively small number of users or the high complexity and/or diversity of topics. Therefore we avoid any dependence on query logs in this work; nevertheless we encourage the use of query logs to improve ranking, should they be available.

Session-based search aims to support the user in her search task by interpreting queries in the context of previous queries and user actions in the same session, without the need to have any knowledge about the user or access to query logs of other users. We can support the user in exploratory search scenarios [WR09] by disambiguating terms based on context information and by providing more relevant search results.

3.1.1. Session Data

A search session S consists of one or more steps. Each step begins after the submission of a new query q by the user and ends when the next query is submitted. A step in a session S is identified by its index position in the list of steps $s_1, \ldots, s_n$. A single step is defined as

$$s_i = (q_i,\, u_i,\, d_i,\, s_{i-1})$$

where q is the query submitted by the user, u defines the user's interaction with the search result page during that step, d is the derived data that was generated by the search engine for that step and s_{i-1} is a reference to the step before the current step (except for the first step s_1 in a session). Similar definitions have already been suggested in previous work [LZD+15; GZY13] with varying levels of detail. The definition used in this work was intentionally held abstract to give implementations of this concept the necessary freedom to choose which data of a session they want to harness. We will use the following definitions of q, u and d in this work:

- Query $q = [(q_1, f_1), \ldots, (q_n, f_n)]$ is a list of tuples that consist of a query string q_i and an associated search function f_i. This function defaults to the regular full-text search, but it allows the user to define more specific filters like the date of publication (before or after) or the name of the author of a publication. The selection of filters depends on the type of data that should be supported.
- User Actions $u = [(r_i, dt_i), \ldots]$, where r_i is the document with index i in the current search result list and dt_i is the dwell time the user took to review this search result. It was shown in previous work [ABD06; FKM+05] that dwell time is a valid measure for implicit relevance feedback. It confirms the intuition

that search results are clicked by users if the preview on the result page seems promising, but the document is only seen as relevant, if the user spends a certain amount of time to read the document instead of immediately returning to the result page to review another item.

- Derived Data $d = (e, r, m)$, where e is the expanded search query, r is the list of search results and m is the list of the currently best matching elements in the topic model along with their scores (see 3.2 Topic Centroid).

While q and u consist of user input that cannot be reconstructed from other data sources, d can be derived at any later point in time, given a session S where at each step d_i has not been saved. Still, it can be useful to store d for various reasons: In session-based search, the state of a session is highly dependent on the state of all previous steps. Recalculating the entire query history during each step would slow down the search engine, while storing the state would cost a certain amount of memory which can be justified by the expected savings in processing time.

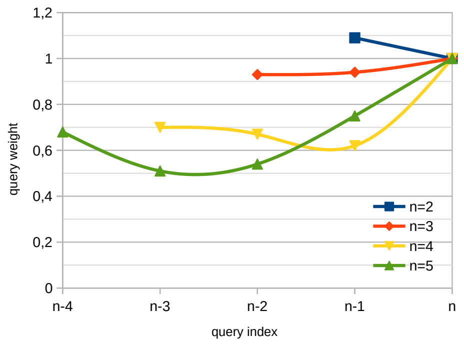
A useful side-effect of saving this additional data is the possibility of efficient navigation in the query history: the user can return to a previous step of the search session, review more results and move from there in a different direction. This is also the reason, why each step has an explicit reference to its predecessor, even though all steps are stored as ordered list in the search session S: When the user is free to navigate in the query history, the order of the steps in the session does not necessarily match the query history at step s_i. By storing the derived data d in each step, the state of the session at that step can be restored at little computational cost and the query history for this step can be reconstructed by recursively following the references up to the first step s_1 in the session.

3.1.2. Query Aggregation

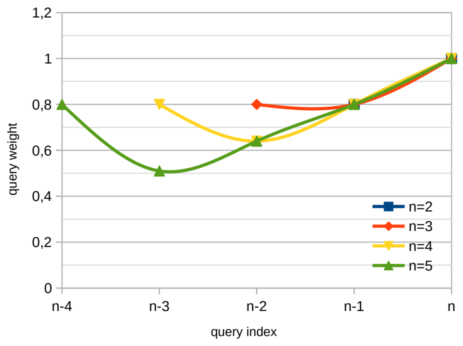
An important feature of session-based search is the inclusion of terms from past queries into the calculation of the search result for the current query. A simple approach to achieve this is to join all past queries and the current one using the OR operator, which was actually done in first attempts at session-based search [SZ03]. This however will likely reduce the correlation of the search results with the most recent search query, which may be unexpected and even frustrating for the user. Therefore, it is advisable to give more weight to the most recent query compared to the previous ones.

This can be achieved by calculating the search results r for each query q in a session individually. To form a unified list of search results r', the results of each search query are joined into a single list by summing up the scores of matching documents. Before the results are joined, the scores of the documents in each result list are multiplied by a weighting factor $w_i \in [0..1]$, whose value depends on the position of the query in the query history. By selecting a larger w for more recent queries, the results matching these queries are preferred over the results of previous queries.

Guan and Yang [GY14a; GY14b] have investigated the optimal weighting factor for a query depending on its position in the query history. They define a function



(A) Optimal query weights as found by [GY14b]



(B) Weights as defined by function $Weight\left(q_i\right)$

FIGURE 3.1.: Query weights depending on the index position of the query and the length of the session

$score\left(q, d\right)$ which calculates the match score of a document d for a query q, which is the result of a regular information retrieval method. The score function of a search session S is therefore defined as:

$$score_S\left(d\right) = \sum_{i=1}^{n} w_i \cdot score\left(q_i, d\right)$$

They call this method *query aggregation*, independent of the actual weight factor and score function that is used. Based on data from TREC 2010-2012, Guan and Yang trained a support vector machine to learn the optimal weight factors for different session lengths. The weights they discovered for search sessions that consist of up to 5 queries are displayed in Figure 3.1a. Their findings show that relevance generally decreases with increasing distance to the latest query, with the exception of the very first query, which always remains about as relevant as the second to last query.

These experimental findings can be roughly approximated by an exponential function with a base $\lambda = 0.8$, with an exception for the first query in a session:

$$weight\left(q_i\right) = \begin{cases} \lambda^{n-i} & for \;\; i > 1 \\ \lambda & for \;\; i = 1 \end{cases}$$

A plot of this function can be seen in Figure 3.1b for comparison. It shall be noted that the experimental results of Guan and Yang are based on query logs that have been obtained from a static search engine and were never tested with a session-based search engine, therefore the suggested value for λ is more of a starting point for further optimization.

3.2. Topic Centroid

In exploratory search scenarios it is to be expected that the user is not fully aware of the specific terminology that is required to find the most relevant documents.

We discussed in the last section, how the ranking of search results can be improved through query aggregation, but this technique still relies on conventional text retrieval methods which will not be sufficient to find relevant documents that do not necessarily contain the keywords the user has entered. In order to find semantically related documents, we are going to use a search index where topic labels have been assigned to all documents. If we find a way to infer the most relevant topics in a search session, we can efficiently search for other documents in the index that are about these topics.

In Section 2.3 we have reviewed a variety of query classification techniques that can help to discover the main topics of a search query by analyzing the text retrieval results of that query. In this section we will introduce *topic identification*, a novel query classification technique that discovers the most relevant topics in a list of documents that have been tagged with topic labels.

A major benefit of session-based search is that we can learn to understand the user's intent in multiple iterations. For each query that the user submits, we can analyze the query and the the search result list to enhance the response to the next search query. The same data can be used to improve the search engine's model of what the user is currently looking for. Therefore we define the *topic centroid*, a data structure that holds the most relevant topics in a search session. It can be stored as a list of tuples, each consisting of a topic t and a score s. Later in this section we present a method to update the topic centroid of a search session with the identified topics of a new search query, which we call *topic shift*.

With this approach we can suggest more documents that are semantically related to the most recent search results, even if they don't match the keywords of the user's query, and we can adjust the scores of matching documents based on the topics they are about.

3.2.1. Topic Identification

We introduce *topic identification*, a novel method to decide, based on a list of search results, which topics are most relevant to the user's current search interest. More generally speaking, we will decide which topics are most relevant in a ranked selection of documents from a document collection, where each document has been assigned at least one topic label.

This task is not related to *topic detection* [All02], which is mainly concerned with text segmentation by recognizing known topics and identifying previously unknown topics, e.g. for news aggregation. Topic identification is related to query classification [SSY+06; BFG+07; BSD10], although the technique we describe in this section is based on pre-calculated topic models instead of ad-hoc text classification and the topic aggregation methods we use are far more fine-grained than those that are used in related work.

Definition

Given a document collection D that consists of documents $d_1, \ldots, d_n$ and a topic model T that consists of topics $t_1, \ldots, t_m$, we assume that each document d has been assigned zero or more topics t (see left panel in Figure 3.2). Furthermore we assume that each topic t that has been assigned to a document d has a certainty

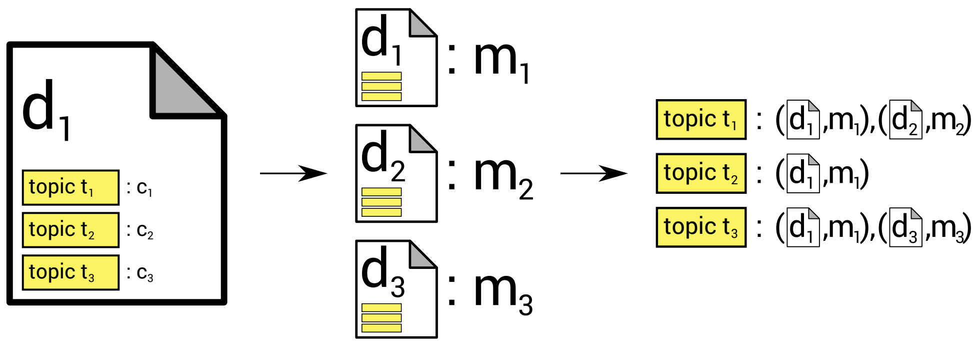


FIGURE 3.2.: Topic identification

score c that is specific to this combination of document and topic. c is an estimation of the probability that document d is actually about topic t. This however does not imply that the sum of all scores c of a document must be 1, because a document can naturally discuss more than one topic. When a topic modeling algorithm has been used to assign topics to documents, the probabilities (and therefore the scores) are usually provided by that algorithm. If however scores are not available (e.g. when topics have been manually assigned by humans), c can be set to 1 for all documents and topics, which just implies that we put the same amount of trust in the validity of all topic assignments.

Our goal is to identify the most important topics based on the search results to a query. A search result is a selection R of documents from the document collection D, where each document has been assigned a match score $m \in \mathbb{R}_{>0}$. The match score is determined by the search algorithm of the search engine and defines the order of the search results. We will identify the most relevant topics among the n highest scoring documents from the search result list. The union of all topics t in the documents d that are among the top n search results defines the candidates for the most relevant topics of this result set:

$$T_n = \bigcup_{d \in R_n} T[d]$$

where R_n is the set of the n highest scoring documents from the search result list R, $T[d]$ is the set of topics that has been assigned to document d and T_n is the union of all topics in these documents. The relevance rating of a topic will be based on three major traits:

1. the prominence of the topic in the search result list
2. the frequency of the topic in the document collection
3. the similarity to other topics, especially in hierarchical topic models

These traits were chosen because each of them solves a different aspect of the relevance problem based on a different data source. The prominence is the most obvious trait, because it derives relevance directly from the occurrence of a topic in the search result list. Still, rating a topic just based on a tiny slice of a document collection will usually lead to skewed results, because topics are in all likelihood not evenly distributed among documents and therefore very frequent topics may dominate the result list while very rare topics will be underrepresented. This is why we look at the frequency of topics in the entire document collection to balance these biases. The third trait that should be taken into account is the semantic reach of a topic: Every domain model that covers a wide variety of topics will have overlaps between different topics. This could result in a situation where several documents that cover the same general topic have different topic labels that are semantically related, which would distort the prominence-based ranking, because the relations between topics are unknown to the ranking algorithm. By analyzing the topic model and joining related topics into virtual super-topics, this issue can be circumvented.

We will examine each of these aspects in detail now.

Prominence in the Result List

The prominence of a topic in the search result list can be defined using several metrics: A topic that frequently occurs in the search result list is probably more relevant than a topic that occurs less frequently. Then again, if a topic occurs in a few high-scoring documents, it is surely more relevant than a topic that occurs more frequently but within documents that have lower match scores. We will try to balance these extremes between high-scoring and frequently occurring topics by defining a metric that considers both properties.

To determine the relevance of each topic $t \in T_n$, the document-score-tuples (d, m) of the search result list R are grouped by the topics of each document (see right panel in Figure 3.2). We now want to reduce each tuple (d, m) to a single value that we can derive the desired metrics from. For this purpose, there are two relevant variables stored in each tuple:

- The match score m of the document, which represents the importance of this document in the context of the current search query
- The certainty score $c_t(d)$ of the topic t under which the document d is grouped, which describes the likelihood that this document is actually about topic t.

Being uncertain about the question whether a document d is actually about a topic t should definitely reduce the rating of topic t, therefore we define the topic match score as the product of m and $c_t(d)$. Now we can define a set of topic match scores for each topic t as

$$M_t = \{ c_t(d) \cdot m \mid (d, m) \in R,\, t \in T[d] \}$$

which contains the match score m of each document d with topic t, multiplied by the certainty score c_t.

Then we calculate three metrics for each topic $t \in T$:

$$\begin{aligned} m_{count}(t) &= |M_t| \\ m_{max}(t) &= \max(M_t) \\ m_{sum}(t) &= \sum_{m \in M_t} m \end{aligned}$$

m_{count} simply counts the number of occurrences of topic t in the result list, which will promote topics solely based on frequency, no matter what score they have. m_{max}

on the other hand relies on the highest match score of all documents that topic t appears in, which completely ignores frequency and only relies the score of the most relevant document with that topic. m_{sum} is defined by the sum of all match scores m of the documents that topic t occurs in, which rewards frequency, but only if the match scores are not negligible. A weighted average of these variables should result in a decent prominence rating.

Before we can reasonably combine these scores, it is necessary to scale them to the same value range. This is due to the fact that the possible values of m may not be under our control and take arbitrary (positive) values, which can result in values for m_{count}, m_{max} and m_{sum} that can vary by several orders of magnitude. To mitigate this issue, we calculate the maximum of each of the metrics and scale the value for each topic relative to the global maximum.

$$p(t) = w_{count} \cdot \frac{m_{count}(t)}{\max\limits_{t \in R} m_{count}(t)} + w_{max} \cdot \frac{m_{max}(t)}{\max\limits_{t \in R} m_{max}(t)} + w_{sum} \cdot \frac{m_{sum}(t)}{\max\limits_{t \in R} m_{sum}(t)}$$

This results in the prominence score p, which is a weighted average of the three metrics m_{count}, m_{max} and m_{sum}, each of which has been scaled to the range [0..1] before the average has been calculated. The sum of the weights w_{count}, w_{max} and w_{sum} must be equal to 1 to get a score that lies within the same range.

Relative Topic Frequency

To avoid the dominance of frequent topics over less frequent ones in a scenario where two topics should be equally important, we add a factor to each topic based on the proportion of the topic's frequency in the result list compared to its frequency in the entire document collection. The calculation of that factor is analogous to *tf-idf* [Spa72], but instead of the term frequency we count the occurrence of each topic in the result list R and compare it to the frequency of the same topic in the document collection D.

The inverse document frequency is defined as the logarithmically scaled inverse fraction of the documents in the document collection D that are linked to a topic t. The resulting score is calculated as

$$\begin{aligned} \operatorname{tf}(t,R) &= |\{d \in R : t \in d\}| \\ \operatorname{idf}(t,D) &= \log \tfrac{|D|}{|\{d \in D : t \in d\}|} \\ \operatorname{tfidf}(t,R,D) &= \operatorname{tf}(t,R) \cdot \operatorname{idf}(t,D) \end{aligned}$$

$\operatorname{tf}(t,R)$ defines the number of occurrences of t in a document d from the result list R, $\operatorname{idf}(t,D)$ is calculated from the size of the document collection $|D|$ divided by the number of occurrences of the topic t in these documents and $\operatorname{tfidf}(t,R,D)$ is the product of the latter two.

Because *tf-idf* does not have a fixed value range, we take the maximum of all calculated *tf-idf* values of the result list and divide all scores by the maximum, which normalizes the results to the [0..1] range and makes them compatible with the prominence score p we discussed in the previous section.

Overlap between Topics

So far we have assumed that each topic is a distinct entity that does not overlap with any other topics. This assumption however is unlikely to hold in reality, especially when hierarchical topic models are involved. This could result in situations where many sub-topics in a very narrow field are counted as distinct and therefore get a lower rating than a less relevant topic that has no sub-topics and was therefore encountered more frequently.

The proposed solution to this issue is to discover a common broader topic for related topics which is then temporarily added to all documents in the result list that contain one or more of the related topics. The discovery process works for (poly-)hierarchical topic models by finding matching ancestors between different topics. Because in a hierarchical model, all topics are related through the root node, there have to be strict limits for the range of discovery. Two important limits should be considered: No common ancestor must be selected from topics that are too close to the root node (the definition of "too close" depends on the size and structure of the topic model) and no common ancestor must be selected through too many recursive visits to a topic's parent (this limit defines which topics are considered to be "related"). The discovery of related topics should be applied recursively, until no more new topics are found or a recursion limit has been reached. If we apply this method to our previous problem, we will see that the more relevant related topics still score lower than the less relevant single topic, but the highest scoring topic will be the common ancestor of the related topics.

Final Score

To bring it all together, the three proposed methods must be combined in order to calculate the final score s which will determine the relevance ranking of the topics. In a first step, we try to discover common ancestors for related topics and add them to the documents in the result list. Then, *tf-idf* and the prominence score p are calculated for each topic t. Finally, *tf-idf* and p are combined. This can be done by either multiplying the scores, which would incur a larger penalty for low *tf-idf* values, or by calculating a weighted average of the two, which gives more control over the influence of each factor on the final score s.

$$s_t = w_{tfidf} \cdot tfidf_t + w_p \cdot p(t)$$

Applied to all topics $t \in T$, this gives us the most relevant topics as a set of topic-score-tuples (t, s).

3.2.2. Topic Shift

The purpose of the topic centroid is to track the most relevant topics in the course of a search session. Now that we have presented a method to identify the most relevant topics in a list of search results, we need a way to update the topic centroid of a session with the identified topics. We define *topic shift* as a method to track changes in the topic centroid in the course of a search session, with the goal to find a decent balance between topicality and stability of the model.

Actions that promote topicality include:

- Adding new topics with high weighting compared to existing topics
- Attenuating topics that are not relevant in the context of the most recent query

On the other hand, actions that facilitate stability of the topic model include:

- Further increasing the weight of frequently appearing topics
- Preserving previously relevant topics if they don't show up in the latest result list anymore

Having a too strong focus on topicality would render the topic centroid redundant, because we would simply replace its state in each iteration with the newly identified topics, while over-emphasizing stability would lead to a topic model that becomes completely static after a few iterations, due to self-reinforcing effects that cause existing topics to dominate the model with no chance for new topics to reach the top.

A First Approach

If there is no topic centroid yet, we can simply initialize it from the topics identified in the first search result list. Otherwise, we join the topic centroid with the identified topics by adding the topics to the centroid and summing up the scores of all matching topics. This would preserve all topics in a session and rank the most frequent and highly scored topics as most important, while new topics could still get ahead of existing ones, if they get higher scores than existing topics over a few iterations.

A similar approach has already been proposed by Daoud et al. [DTB+09], which showed promising results in an experimental evaluation. Experience gained in the course of this work however has shown that in a longer session, many high-scoring topics may accumulate, which will make it harder for new topics to replace existing ones, thereby locking the search results in on the dominant topics of the first few search queries. To achieve a decent balance between stability and topicality, it is a desirable property that new topics can supersede the existing ones not just shortly after the start of a search session.

We can prevent old topics that do not necessarily show up in the search results anymore from further dominating the topic centroid by implementing a measure that decreases the relevance of all topics over time. Topics that have accumulated high scores can therefore remain in the topic centroid for a while, but they can only retain their dominant position, if they frequently appear in subsequent search results.

Another issue is the rating of frequently appearing topics: Topics can accumulate very high scores when the score of an identified topic is simply added to the score of the same topic in the topic centroid. Reducing the rate at which the scores of already existing topics increase will make it easier for new topics to get a hold in the topic centroid.

The Suggested Solution

In a search session, the topic centroid gets updated after the results to the submitted query are available and the most relevant topics for those search results have been computed. If the topic centroid is empty, the topics and scores which are the result of the topic identification are taken as new topic centroid. Otherwise, the topic shift algorithm is executed on the existing topic centroid and the results of topic identification on the last search results. To merge topics into the existing topic centroid, we add new topics, increase the scores of reoccurring topics and decrease the score of topics that did not reappear.

Before the topics are merged, we multiply the score of each topic in the topic centroid by a fixed fraction $f_{cooldown}$. This measure helps to prevent existing topics from dominating the topic centroid, even if they are not relevant for the latest search results anymore. Then we add all topics that are not yet part of the topic centroid with their scores unchanged. The scores of topics that are already part of the topic centroid are updated according to this formula

$$s = \max\left(s_{old},\, s_{new}\right) + w_{shift} \cdot \min\left(s_{old},\, s_{new}\right)$$

where s_{old} is the existing score in the topic centroid, s_{new} is the new score in the topic aggregation and w_{shift} is a value between 0 and 1 that determines the rate at which scores will accumulate over time. The result of the max-function ensures, that the new score will not be lower than the existing score (after being reduced by $f_{cooldown}$), while w_{shift} helps to prevent the accumulation of very high scores, if a value lower than 1.0 is used.

Example

The table below shows an example of the development of a topic centroid (TC) over multiple steps with new topics provided by topic identification (TI). Topic shift is applied with $f_{cooldown} = 0.8$ and $w_{shift} = 0.5$:

step 1		step 2		step 3	
TI	TC	TI	TC	TI	TC
$(t_1, 1.0)$	$(t_1, 1.0)$	$(t_1, 0.7)$	$(t_1, 1.15)$		$(t_1, 0.92)$
		$(t_2, 0.9)$	$(t_2, 0.9)$	$(t_2, 0.9)$	$(t_2, 1.26)$
				$(t_3, 0.6)$	$(t_3, 0.6)$

In the first step, we see that only topic t_1 is identified with score 1.0, which is copied to the topic centroid. From the results of the second query, topics t_1 and t_2 are identified. As t_2 was not part of the centroid before, its value can be copied as well. The current value of t_1 in the topic centroid is multiplied with $f_{cooldown}$, which results in 0.8 and the new score is calculated as $s = \max(0.8,\, 0.7) + 0.5 \cdot \min(0.8,\, 0.7) = 1.15$. Even though the score of t_1 was lower in the new aggregation, its score in the topic centroid remains higher due to its history, but not out of reach for the new topic t_2. After the third request, t_1 does not appear anymore in the result list, therefore its score is just reduced by $f_{cooldown}$, but t_2 shows up again and can take over the position of t_1 after the topic shift algorithm has been applied.

3.2.3. Relevance Feedback

Topic identification, as defined in this section, is based on the assumption that the score of a document in the search result list correctly resembles its relevance for the user's search interest. It is obvious that this assumption will not always hold in practice, because the search engine may fail to rank the most relevant documents first, especially in complex research tasks where the user might not have sufficient background knowledge to properly express her search interest. Previous work has shown that the user's click behavior on the search result page can give valuable implicit feedback from which the relevance of reviewed documents can be derived [ABD06; FKM$^+$05].

We could use this user feedback to adjust the scores of the documents in the result list before topic identification is applied, so the topic centroid more closely resembles the user's perception of relevance rather than the search engine's suggestions. This however comes at a cost: In our current model, topic identification can be executed immediately after the search results are available, so the topic centroid can be updated before the search result page is presented to the user. If we were to include the relevance feedback, we could only update the topic model *after* the next query has been submitted. An immediate update has the advantage that the topic centroid is already adjusted to the current search results, from which other functions of the search engine can benefit, like topic graph visualization or topic-based search results which will be discussed later in this work.

In this work, we decided that the drawbacks of including relevance feedback are too high compared to the possible gains, therefore we continue to use the document scores as returned by the search engine. Finding a way to include relevance feedback into the topic centroid without sacrificing topicality is an interesting problem for future work.

3.2.4. Topic Graph Visualization

For the user of a search engine that relies on the topic centroid for ranking, there is no direct feedback channel to see which topics are currently most relevant. This is usually not an issue, as topic-based ranking does not require explicit feedback or even awareness of the user, but there is a variety of reasons to still provide some kind of visualization of the topic centroid: It can help the user to check whether the topic centroid actually resembles her search interests and to move the search into a different direction, if the topics don't appear to be relevant. This can be achieved either by submitting new queries or directly through interaction with the topic graph. The visualization can also help to explore new topics and to find query terms that could be relevant for the user's search interest. In a multi-user environment (e.g. collaborative search), the visualization can help to get an impression of what other participants are searching for.

Objective

The topic centroid is a selection of topics from a topic model, which can be represented as a directed graph with nodes representing topics and edges indicating relations between topics. These relations can be hierarchical (from topic to sub-topic) or based on semantic similarity between arbitrary topics.

Our goal is to visualize the most important topics of the topic centroid as well as the relations between these topics. We make use of the hierarchical structure of the

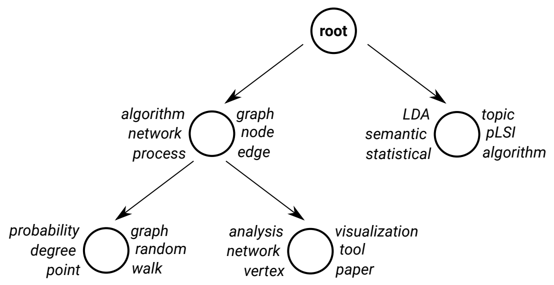


Figure 3.3.: Visualization of the topic centroid. Each node represents a topic, surrounded by the distinctive terminology for that topic. Directed edges indicate hierarchical relations between topics.

topic model to visualize the topic centroid as a subgraph of the topic model in a tree-like structure. The number of displayed topics depends on the space that has been reserved for the topic graph on the search result page. Through interaction with the topic graph, the user should be able to remove topics that are not relevant for the search and to explore new sub-topics of a topic node.

Concept

In order to visualize the topic centroid, a human-readable representation of topics as nodes of a graph is required. Ideally, a descriptive label would be displayed for each topic, but when statistical topic models are used, such labels are not available and the difficulties of automatic topic labeling (see Section 2.1.3) further impede this solution for large topic models. As an alternative, we can display the distinctive terminology that identifies each topic in a statistical topic model, similar to the visualizations of topic models by Paisley et al. [PWB$^+$12]. A node is displayed as circle, around which the six most distinctive terms are arranged, as shown in Figure 3.3. Given a high-quality topic model, the user should be able to identify the topic based on these terms with a short glance.

To build the initial tree structure of the topic graph, the n highest scoring topics from the topic centroid are selected, where n depends on the available screen space to render the graph. We try to find common ancestors for each combination of topics in the topic centroid by recursively visiting each topic's direct ancestor. Whenever a common ancestor for two or more topics is discovered, it is added to the graph and an edge from the ancestor to all of its descendants is drawn. This process is repeated until a single common ancestor is found or the root node is reached. In the worst case, all n topics are only connected through the root node, but in a sound hierarchical topic model, usually a tree with several branches will evolve.

To indicate relations between topics in different positions of the topic hierarchy, we add undirected edges between semantically related topic nodes. Those edges can connect arbitrary nodes, i.e. they do not have to comply with the tree structure that was built so far. We define the relatedness of two topics by a combination of two measures: the co-occurrence of significant terminology and the overlap of documents. The co-occurrence is defined as the number matching terms in the lists of significant terminology of two topics, divided by the length of the shorter list of terms. Analogously, the overlap of documents is defined as the number of documents that are associated with both topics, divided by the number of documents that have at least one of the two topics. The average of both measures is the relatedness score of two topics; if this score reaches a certain threshold, a connection between these topic nodes is drawn in the graph.

Through interaction with the topic graph, the user can directly manipulate the contents of the topic centroid. Each topic node has three control elements that will appear on touch or by hovering with the cursor over the node. These elements are buttons in form of small circles: a button with a yellow star icon that appears in the top left corner, a button with red cross icon that appears in the top right corner and another button with a blue plus icon at the bottom of the topic node. A topic can be promoted by clicking the button with the star icon, which will increase the score of that topic and prevent it from being removed from the topic centroid. By clicking the button with the red cross, a topic is removed from the graph and the topic centroid. When a topic is removed, all of its sub-topics are removed as well, both from the graph and from the topic centroid (therefore the root node cannot be deleted). After one or more topics were removed and less than n topics are currently displayed, new topics from the topic centroid that were previously hidden are added to the graph, for which common ancestors are discovered as described before.

Starting from a topic node, the user can explore new sub-topics of the topic by clicking the plus-icon, if the topic has at least one sub-topic. This will add up to three sub-topics to the graph and connect them to the selected node. The sub-topics are selected by the highest score in the topic centroid, with a fallback to random selection, if not enough topics are not part of the topic centroid. Because adding more topics might make the graph too crammed for the limited screen space, low-scoring topics are automatically removed from the graph, when a certain threshold is reached. These topics will not be removed from the topic centroid however, this will only happen if the user explicitly deletes a topic node. The next topic that will be hidden from view is determined by these conditions: It must be a leaf node, it is not a child node of the topic that is currently being explored and when multiple candidates are available, the topic with the lowest score in the topic centroid is chosen (the score of a topic that is not part of the topic centroid defaults to zero). These rules ensure that the least relevant topic nodes are hidden first and that topics that have many possibly relevant child nodes are not removed, even if they themselves are of little importance.

This concept allows us to visualize the most important topics of the topic centroid and gives the user the ability to selectively remove irrelevant topics as well as to explore new topics and relevant terminology to be used in subsequent queries. A weakness of the method is the lack of proper topic labels and therefore the dependence on the significant terminology that defines a topic, which in some cases can be difficult to understand. The upside is that this solution is applicable to any statistical topic model and it achieves the goal to provide not only a visualization, but also a feedback channel for the otherwise hidden structure of the topic centroid.

3.3. Search Strategy

In this section we describe a search strategy which forms the foundation of a session-based semantic search engine. We will give insights into the search algorithms that incorporate the concept of session-based search and harness the semantic information that is gathered in the topic centroid. Then we give an overview of the process from the point where the user submits a search query until the finished search result page is returned to the user. The strategy described in this section was intentionally held abstract; in Chapter 4 we discuss the details of a prototypical implementation of this concept.

3.3.1. Prerequisites

This search strategy is aimed at text documents from which a reasonable plain text representation can be extracted, like it is the case for PDFs, e-books or HTML documents. The full plaintext representation of the documents as well as relevant metadata like title, authors, date of publication, etc. are stored in an inverted index for efficient document retrieval.

A hierarchical topic model is built from the indexed documents, using one of the algorithms discussed in Section 2.1.2. With this topic model the most likely topics are computed for each document, which are stored in the search index in the respective document's metadata. Each topic t of a document has a certainty score c, which describes the probability that the document is actually about topic t. This is owed to the fact that topic modeling algorithms cannot predict perfect labels for each document, but some of them have a good notion of how likely it is that their estimate is true. Therefore each document in the search index has a list of topic-score-tuples (t, c), which can be harnessed by search algorithms that are not entirely keyword-based.

3.3.2. Search Algorithms

We define a search algorithm for session-based semantic search, which incorporates both full-text search enhanced with session data and topic search, which finds documents that are semantically related to the state of the topic centroid. The combination of these two approaches is controlled by a weighing parameter which allows the search engine to focus on one of the two aspects or to find a decent balance between both approaches.

Full-Text Search

The full-text search algorithm follows the well-proven vector space model [SWY75], which represents documents as real-valued vectors of *tf-idf* weights. Matching and ranking of documents is done by expressing the terms of a query as a vector in the same vector space and then comparing the cosine similarity of the query vector with the documents vectors in the search index.

A major trait of session-based search is the inclusion of terms from the query history H with varying weights into the expanded query, as described in Section 3.1. This can be achieved by retrieving the results for each query $q \in H$, multiplying the scores of the resulting documents with the weight of the query depending on its position in the query history and then combining the result lists by summing up the scores of matching documents.

This approach however would not scale very well for long search sessions, therefore we propose a different solution where a single query is generated which considers the weight of all terms in the query history. This is done by first creating a query vector from the concatenation of all queries in the query history. To incorporate the weights of different queries, the *tf-idf* score of each term in the vector is multiplied by the weight of the most recent query that term occurred in.

Matching of query terms is executed on the full-text index by default, but search results can be further enhanced by additionally matching on other metadata fields like the document's title, authors, abstract or others. For each of these fields, a new inverted index is created and a separate list of search results is generated using the same query vector as for the full-text search. The results are combined by joining the result sets of the different search indices and summing up the scores of matching documents. We can further improve the ranking by applying weights to different fields, in order to capture the intuition that a match on the title field is more relevant than a match on a single term somewhere in a document. To achieve this, the scores of each result list are multiplied by a weight factor that we define for each indexed field before the final rank scores are summed up.

To further improve search performance, a variety of techniques can be applied for document pre-processing, language analysis, indexing, query expansion, parallelization, etc. (see [BR99] for an overview). A description of these standard techniques is beyond the scope of this work and a reasonable selection of these methods is left open to the implementation of the search engine.

Topic Search

While full-text search allows us to reliably retrieve documents that match the terms in the query history, we are going to harness the topic centroid and the topic labels that have been assigned to the documents in the search index to improve the ranking of documents that were matched through full-text search, as well as to discover new documents that could not be matched with conventional information retrieval methods.

Each document d in the search index is associated with a set of topic-score-tuples (t, c), where the score c represents the likelihood that document d is about topic t. The topic centroid follows the same pattern: a set of tuples (t, s), where s represents the relevance of topic t in the context of the current search session, relative to the other topics (meaning there is no upper bound for s).

In order to match the most relevant documents for a topic t, we select all documents from the search index which contain a reference to the same topic. We can order the selection of documents by the certainty c in descending order to get those documents first, that are most likely about topic t.

The topic centroid usually consists of more than one topic, therefore we want to find all documents that match at least one of the topics in the topic centroid, but we would intuitively assume that a document is more relevant if:

- it matches several topics from the topic centroid
- the matched topics from the topic centroid have high relevance scores
- there is little uncertainty that the document is actually about the matched topic

To capture this intuition, we define a custom topic rank function whose results can be used to retrieve the most relevant documents for a specific topic centroid. For each document d, we calculate the intersection I of the topics in the topic centroid and the topics of the document. The rank score for a topic $t \in I$ is given by multiplying the certainty score c for this combination of topic and document with the topic's score s in the topic centroid. The rank score for the entire document d is the sum of the rank scores for each topic $t \in I$:

$$rank\left(d, I\right) = \sum_{t \in I} certainty\left(d, t\right) \cdot centroid\left(t\right)$$

This rank function allows us to get an ordered list of documents that best resemble the topics and their weights in the topic centroid.

Combination

In order to benefit from both the reliability of full-text search and the semantic knowledge that is harnessed through topic search, we combine both approaches by joining the result lists and calculating a weighted average score for all documents. The weighted average is proposed in order to have some influence on the ranking that is used by the search engine. Users tend to expect a strong correlation between the keywords they enter and the search results they receive, therefore a stronger focus on full-text search can be viable.

Because the match scores that are generated by the retrieval algorithms we use do not necessarily map to the same value range, the results of both methods are normalized to the [0..1] range to give equal influence to each score.

3.3.3. Query Pipeline

The query pipeline describes, which steps the search engine takes to calculate the answers for a user's search query in the context of a search session. Figure 3.4 visualizes the steps in the query pipeline and the data structures that are accessed or modified during each step.

The query pipeline relies on two major data structures: the user-specific search session and the search index which is used for document retrieval. As part of the pipeline, two expanded queries are built using the method described in Section 3.3.2 with different parameters: The main query, which provides the documents for the search result list, and the suggestion query, which has a much stronger focus on topic search and whose results can be suggested to the user outside of the main search result list.

When a query is submitted, the search engine loads the user's session context or creates a new search session, if this is the first query of the user or the creation of a new session has been explicitly requested. Then the steps of the query pipeline are executed in sequence, due to data dependencies between each action. We will now take a more detailed look at each step in the query pipeline.

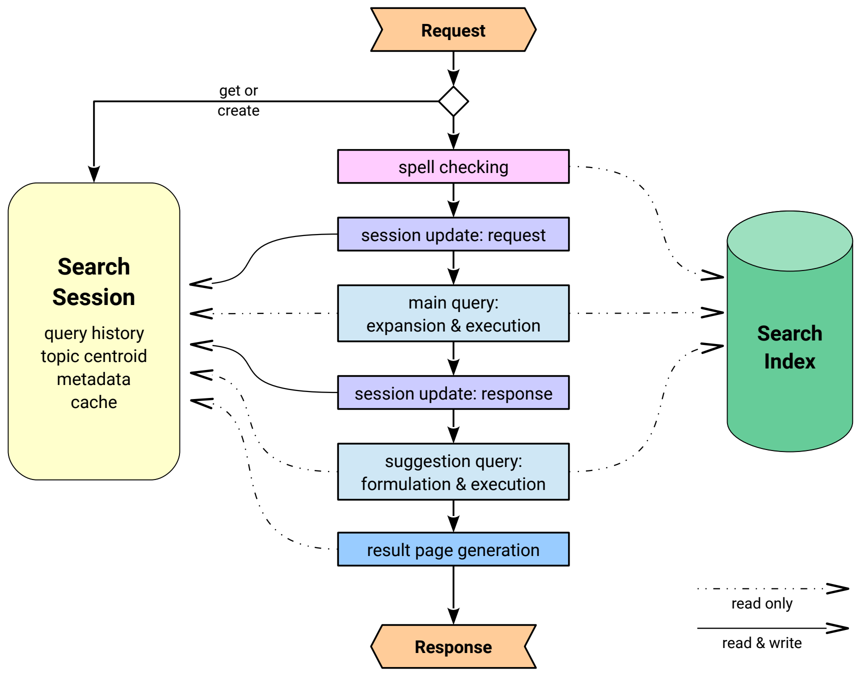


FIGURE 3.4.: From request to response: steps in the query pipeline

Spell Checking

Correcting spelling errors is a standard procedure in modern web search engines, as approximately 10-15% of queries contain spelling errors [CMS10]. The spell checker is at the very top of the pipeline, because it allows us to search for what the user meant, not what might have been typed accidentally, although the user should always be made aware of automatic spelling correction and the search engine should provide an option to search for the original terms in case the spell checker was mistaken.

Spelling errors are conventionally detected by comparing terms to a spelling dictionary and possible corrections are generated by comparing the misspelled term to the words in the dictionary using a similarity measure like the Damerau-Levenshtein distance [Dam64], which counts the number of insertions, deletions, substitutions or transpositions on the character level that are required to get a match. If a term in the dictionary has a sufficiently small edit distance to the apparently misspelled term, it is suggested as correction. With literature search engines however, this approach alone is usually not sufficient, because many queries will contain technical terminology, names and other terms that are not part of any regular dictionary.

To mitigate this limitation, additional data sources like query logs and terminology found in the search index can be used. For this purpose, a dictionary can be built from the document-term-matrix in the search index, which contains all terms that are found in any document. These terms can obviously also contain spelling errors and there are going to be many unique tokens like names and email addresses that are not relevant for spell checking, but decrease the performance of the system. Both issues can be solved statistically by only including terms in the extended dictionary which only appear in at least $n \geq 2$ documents, which will increase the likelihood that the spelling of the term is correct and vastly decrease the number of tokens that will be stored in the dictionary.

This approach allows us to fix common spelling mistakes (using the regular dictionary) and errors that concern the terminology of the documents in the search index, which won't be enough to suggest corrections for arbitrary spelling errors, but it allows us to suggest corrections for nearly all spelling issues that concern the actual contents of the search engine.

Session Update: Request

Before the actual search is performed, the state of the search session (see 3.1.1 Session Data) is updated. A new step is added to the search session which at this point only contains the new query. If recorded user interactions with the previous search result page are available, they are added to the state vector of the previous step.

Main Query: Expansion & Execution

In this step, the main query is generated and the documents for the search result list are retrieved from the search index. We want these documents to strongly correlate with the query terms, therefore we define a higher weight for the full-text search results compared to those of the topic search algorithm. This way, the user still benefits from the semantic capabilities of the search engine, but the ordering of search results is closer to what the users expect from traditional search engines.

Session Update: Response

After the results of the main query are available, the next step is to update the topic centroid based on the response to this query. The 10 highest scoring documents are used to identify the most important topics (see 3.2.1 Topic Identification) and consequently to update the topic centroid (see 3.2.2 Topic Shift).

Suggestion Query: Formulation & Execution

Just like the main query, the suggestion query is a combination of full-text search and topic search (as defined in Section 3.3.2), but this time the parameters were chosen so that topic search largely outweighs full-text search. This configuration will prefer documents that share the highest-scoring topics of the topic centroid, independent of the keywords that have been specified by the user; the results are predominantly selected based on the semantic structure of the documents. Also, the results are derived from the topic centroid after it has been updated by the results of the main query, which makes them consider the latest topics and allows us to display topic-based suggestions starting with the first query in a session. The suggested search results are intended to be displayed outside of the main search result list, for example in a sidebar, to give the user some additional hints about documents that might be relevant, even if they do not necessarily match any of the query terms that were submitted so far.

Result Page Generation

Finally, the search result page is built and returned to the user. The specifics of the user interface are not the main concern of this work and therefore left to the implementation; in Chapter 4 we describe the implementation details of our own prototype.

3.4. Collaborative Search

In the previous sections, we have laid the foundations for a session-based semantic search engine, where queries are interpreted in the context of the user's recent interactions with the search engine and the ranking of search results is enhanced with the help of the topic centroid, a data structure that keeps track of the most relevant topics in a search session. Now we want to extend this concept to the collaborative setting, where multiple users work on the same task at the same time. We want to focus on the question, how we can further improve the ranking of search results based on the additional context information that can be derived from the actions of several users that form a group with a common search interest. For this purpose, we introduce the shared topic centroid, an extension of the topic centroid that was presented in Section 3.2. Around this new data structure, we build a variety of tools that help to support the group members in their collaborative search effort.

3.4.1. Shared Topic Centroid

The shared topic centroid is the central data structure that allows the ranking of search results for all users to be improved, based on context information that has been gathered by all group members. This is an extension of the session-based scenario where the user has a private topic centroid that gets updated with each submitted query in the course of a search session. In the collaborative scenario, all members of the group still have their private topic centroid that gets updated whenever a new query is submitted, but the topic shift process has been expanded: Instead of shifting based on just the topics that were identified in the list of search results, the user's local topic centroid is also influenced by the shared topic centroid and vice-versa.

Mutual influencing of the private and the shared topic centroid happens whenever a group member submits a new query to the search engine, before the query pipeline is executed. We can update the user's private topic centroid TC_u by simply applying the topic shift algorithm (see Section 3.2.2) with the shared topic centroid TC_s as a source of new topics. However, updating the shared topic centroid using the same method would cause several issues: If we assume that multiple users submit queries with similar average frequency, updates to the shared topic centroid happen far more frequently than updates to each user's private topic centroid. To increase the stability of TC_s, the influence of the factor $f_{cooldown}$ must decrease when more users are active. Adding a new high-scoring topic to a private topic centroid is not an issue, because the topic clearly was prominent among the user's last search results, but new topics should never get a prominent position in the shared topic centroid without some level of agreement between the majority of the collaborating users. To mitigate this issue, we evenly divide the weight of all contributions to the shared topic centroid among the active users, so when multiple users agree that a topic is relevant, its score will accumulate over time. Therefore, if only a single user adds a new topic with a high score, it won't reach the top of the topic centroid without some acceptance among the other participants. We can apply the same principle to the weighting factor w_{shift} that determines how fast the score of a recurring topic accumulates.

In summary, these are the adjustments to the topic shift algorithm we propose in order to have a well-balanced shared topic centroid:

$$\begin{aligned} s_{new'} &= \frac{s_{new}}{|users|} \\ w_{shift'} &= \frac{w_{shift}}{|users|} \\ f_{cooldown'} &= f_{cooldown} + (1 - f_{cooldown}) \cdot \frac{|users|-1}{|users|} \end{aligned}$$

where $s_{new'}$ is the score of a new topic that is added to TC_s, $w_{shift'}$ is the adjusted weight factor for the topic shift, $f_{cooldown'}$ is the adjusted cooldown factor and $|users|$ is the number of active users in a session. Both s_{new} and w_{shift} can simply be divided by the number of active users in order to evenly distribute the influence that each user has on the topic centroid. The influence of $f_{cooldown}$ gets adjusted by the number of users so that the reduction of topic scores remains stable between two average query submissions of a user.

So whenever a user submits a new query, the shared topic centroid gets adjusted with the topics from the user's private topic centroid using the adjusted topic shift algorithm we have just described. Then the user's private topic centroid gets updated with the topics from the shared topic centroid using the regular topic shift algorithm. For the update of the private topic centroid, different values for w_{shift} and $f_{cooldown}$ can be used than in the regular query pipeline, which enables more fine-grained control over the influence of the shared topic centroid. By adjusting these parameters, we can create a configuration where the private topic centroid is always a copy of the shared topic centroid (using $f_{cooldown} = 0$) or where the shared topic centroid has no influence on the private topic centroid at all (with $f_{cooldown} = 1$ and $w_{shift} = 0$). However, to achieve a decent balance between personal and group influence on each user's search results, we recommend values for $f_{cooldown}$ between 0.5 and 0.9 and w_{shift} between 0.1 and 0.5, with the sum of both variables being around 1.0 to get stable results.

3.4.2. Group Management

Before multiple users can search collaboratively, they must form a group. For this purpose, each user creates an account with the web service that provides the collaborative search engine (anonymous access would be conceivable, but rather inconvenient for frequent usage). Users can then create groups and invite other users to join them. Within a group, multiple projects can be created. A project serves as container for all data structures that are used in a collaborative search session, including the shared topic centroid and all other information that is shared among the project members. Each user can be part of multiple groups and multiple projects per group, but in a single client session, only one combination of group and project can be active at the same time. This limitation can be circumvented by opening multiple client sessions (i.e. multiple tabs in a web browser) to simultaneously work on several projects, possibly in different groups. The shared topic centroid of a project is initially empty and gets filled when the first project member submits a query. Users that join a project start with a copy of the project's shared topic centroid as their private topic centroid, so that the results to their first query can already benefit from the group's previous work.

Group influence on each user's search results only happens implicitly through topics that are transferred over the shared topic centroid; the queries of individual users are only expanded based on their own query history (as described in 3.1.2 *Query Aggregation*) and not with query terms from other users. This choice was made for several reasons: The core principles of a session-based search engine should not be diluted in the collaborative scenario; the users should still have control of their own query history, which should be the main influencing factor on their search results. The topic centroid already promotes documents that are in line with topics that the group is most interested in, so the benefit of adding more query terms is expected to be rather low.

Privacy in collaborative search scenarios is another issue that has not gained much attention in related work so far. Systems like SearchTogether [MH07] and CoSearch [AM08] show all queries and the names of the persons that submitted them to the entire group. The authors emphasize the benefits of sharing this information which can have a positive influence on the research of other group members, but they largely ignore the issues that arise from this kind of exposure: Individuals behave differently when they know they are being watched in a group, which can entice some to work harder, but it also may impede others to work freely to avoid negative feedback from other group members. Seeing what other group members search for can be an inspiration for the user's own search, but it may also prematurely limit the direction that the group is taking, making it less likely that new ideas are explored. By not showing the queries of other users yet still enabling implicit feedback through the shared topic centroid, we hope to achieve the best of both worlds: the freedom to explore ideas without constant exposure to the group and improved ranking of search results based on the dominant topics among all group members.

3.4.3. Collaboration

One of the greatest challenges when designing a collaborative information retrieval system is to encourage collaboration between users. In the literature review in Section 2.4 we have seen a variety of methods to support collaboration in different scenarios. In this section, we will not present a complete framework for collaborative work in a specific context; instead, we provide the tools that are necessary to benefit most from the concepts we have presented so far. Our contribution is to create a concept that allows to improve the ranking of search results based on the search activity of a small group of collaborators. This is not an alternative to existing systems, but rather a method to enhance the search experience in different scenarios of synchronized collaboration, be it co-located or remote.

Sharing of Results

The ability to easily share search results is particularly important in a collaborative search scenario where users otherwise do not see what the other group members are working on. Each user can save a search result by clicking a star icon that appears in the search result list next to each document. Each document that has been saved this way appears in the user's private list of bookmarks for the current project, as well as in a shared list that is a combination of the bookmarks of all users. In this list, the origin of a bookmark is not displayed, but next to each document is a number that indicates how many users have bookmarked this document so far and another star icon which shows whether the document from the shared list is also in the user's private list. In order to display the most relevant documents first, the shared list is sorted by the number of users that have bookmarked a document. Users can "upvote" a document by adding it to their private list of bookmarks, either by finding the document among their search results or by clicking the star icon in the shared bookmarks list. The star icon always has one of three states: empty (not bookmarked), light (bookmarked by other group members) or dark (bookmarked by the user). Therefore, it is easy to see among the search results to any query, which document has already been reviewed by another user.

Documents that have been added to the shared bookmarks list are obviously relevant for the current project and are therefore suitable as an additional source of influence for the shared topic centroid. We can expand the topic shift algorithm for the shared topic centroid to include the relevant topics of the shared bookmarks list. To find the most relevant topics, the topic identification algorithm (see Section 3.2.1) is applied to the list of saved documents. The score for each document is derived from the number of users that bookmarked that document, so that more weight is given to documents that have been bookmarked by more users. Whenever a new document is added to the shared bookmarks list, the topic shift algorithm can be executed on the shared topic centroid, using the topics of the bookmarked documents as source.

Topic Visualization

Both the private as well as the shared topic centroid can be visualized as described in Section 3.2.4. However, for the shared topic centroid some restrictions apply: Interactions that may alter the shared topic centroid are disabled by default, because a single participant's mistake can potentially harm the ranking of search results for all other users. In order to still allow direct control over the shared topic centroid, it may be reasonable to define privileged user roles which have permission to delete nodes from the shared topic centroid or to boost the score of a topic. The creator of a project would automatically be assigned that role and have the right to transfer this role to other project members.

Topic-based Recommendations

We have introduced the suggestion query (see Section 3.3.3), which finds relevant documents based on the topic centroid in the single user setting. In the collaborative setting, we can still generate these results based on each user's local topic centroid, but the same query can be executed based on the shared topic centroid. Suggestions based on the shared topic centroid are directly derived from the common search context of all group members and therefore likely to be relevant for the project, but they also give some feedback about the accuracy of the topic model. Because the shared topic centroid does not change as fast as the private topic centroids do, we can cache the suggestions from the shared topic centroid for a few iterations and add these documents to the response of each user query without additional computational cost.

3.5. Discussion

We have presented the core concepts that are required to build a session-based semantic search engine which harnesses contextual information that is implicitly generated by the user in the course of a search session to improve the ranking of search results. These concepts are both applicable in single-user environments as well as in a collaborative setting, where multiple participants work in real-time on the same research task.

We defined the search session as the central data structure which serves as a container for all context information of a single user that could be relevant for the ranking of search results and which also contains other data structures like the topic centroid. Furthermore we discussed query aggregation, a method to build an expanded search query that includes terms from a session's query history, yet still returns search results that correlate well with the user's latest query. While most of the ideas we discussed concerning session-based search were already covered in related work (see Section 2.2), we decided to still present them in order to build a coherent foundation for the following concepts.

To get an understanding of the latent topics that the user is looking for, we have specified the topic centroid, a data structure that holds the most relevant topics that were derived from the user's interaction with the search engine in a search session. We introduced topic identification, a novel query classification technique that discovers the most relevant topics in a ranked list of search results. This method aggregates the topics that are associated with each document in the search result list and assigns scores to topics based on the ranking of documents in the result list, the relative frequency of topics in the document collection and the semantic similarity of different topics. Based on the topics that have been identified, the topic centroid is updated using another method we call topic shift, which guarantees a decent balance between topicality and stability of the topic centroid. Furthermore, we have defined a concept to visualize the contents of the topic centroid, even if no human-readable topic labels have been assigned to the topics, and provide ways to explore the topic model as well as to manipulate the contents of the topic centroid.

Based on these concepts we specified the search strategy for the single user scenario. As main search algorithm we have defined a combination of full-text search and topic search: The full-text search algorithm relies on the vector space model and uses an expanded search query to match documents based on weighted terms from the entire query history of the search session. The topic search algorithm on the other hand harnesses the state of the topic centroid to retrieve documents that are relevant in the context of the search session, even if they do not match the user's query terms.

We presented a query pipeline that defines all steps from search request to response, based on the data structures and search algorithms that were described so far.

As a last step, we extended the idea of a session-based semantic search engine to the collaborative setting. Our core contribution is a concept that allows to improve the ranking of search results based on the search activity of a small group of collaborators. For this purpose we introduced the model of a shared topic centroid, an extension of the topic centroid which can gather context information from multiple users. Furthermore, we presented intuitive ways to organize and share search results and provided methods to harness the contents of the shared topic centroid through topic-based recommendations and visualization techniques. This concept is not a comprehensive alternative to existing collaborative systems, but rather an extension that can improve the efficiency of collaborative search in various scenarios that have been discussed in related work (see Section 2.4).

In the next chapter, we are going to show the prototypical implementation of a search engine based on a selection of the concepts we have presented, with a focus on the single-user scenario.

Chapter 4

Prototype

We have built a prototype of a session-based semantic search engine which implements most of the concepts described in Chapter 3, including search sessions and topic-based search backed by a topic centroid. The goal of the prototype is to test whether the new semantic features are beneficial when compared to conventional literature search engines, which we further investigated in a user study (see Chapter 5). The concepts described in *3.4 Collaborative Search* are not in the scope of this work and were not implemented so far.

The prototype consists of a web application which provides the search mask and manages the entire query pipeline, while the heavy lifting of executing search requests is done by Elasticsearch [@Ban], a distributed information retrieval system built on top of Apache Lucene. Furthermore, we provide a set of tools to build the search index, a hierarchical topic model and other relevant data structures. The search index contains the full text of about 1.2 million scientific papers from the arXiv repository [@Gin].

During the design and implementation of the prototype, we went to great lengths to ensure that it becomes a practicable solution which aims to be usable outside of the academic context. The use of Elasticsearch ensures scalability of the resource-intensive parts of the prototype and low latency even for complex search requests that incorporate session data like the user's query history or relevant topics from the topic centroid. The search engine is built as a lightweight web service which manages user sessions, implements the topic centroid, generates the expanded queries for Elasticsearch, interprets the search results and builds the search engine result pages. The tools for building the search index are compatible with arbitrary document collections; the only function that was specifically implemented for the arXiv repository is collecting document metadata, which is hard to generalize due to the lack of a widely adopted standard. This setup should contribute to make this prototype a scalable and generic solution that is applicable to document collections whose size exceed what we will show in this work.

In this chapter we will explain the motivation for selecting the arXiv document collection and illustrate how the data was prepared and the search index was built. We will then give some insights about the core functions of the search engine and how the concepts that were described in Chapter 3 have been implemented. Next we take a closer look at the user interface of the web application and how the semantic features of the search engine are presented to the user. We finish this chapter with a performance review of the prototype and a general assessment of its utility.

4.1. Document Collection

We have built a search index from documents in the *arXiv* [@Gin], an open repository of e-prints in Physics, Mathematics, Computer Science and other fields. The search index contains metadata and the full text of all 1.2 million documents that have been submitted to arXiv until March 2017, as well as a topic model that was built from these documents. In this section, we will explain why we chose arXiv as document collection and how the topic model and the search index were built.

4.1.1. Selection Criteria

Finding a suitable document collection for the search engine was not an easy task due to a number of restrictions we developed after reviewing existing research in the field of session-based search, in order to pave the way for a successful study involving the prototype of a session-based semantic search engine. The major restrictions are:

1. The data set should be reasonably large, so that it allows for realistic search scenarios and to see whether the search engine works at a relevant scale
2. The document collection should be open and not subject to restrictive licensing, so that everyone has the chance to re-build the index and put the results of this work to the test
3. There should be no major access barriers to retrieve the full document collection and the cost to do so should be reasonably low

Access restrictions are a common issue with contemporary research on session-based search, which relies on data sets like those used in the TREC Session Track [CKH+14] or the HCIR challenge [CGK+12]. Unfortunately, those are either cumbersome and expensive to receive (e.g. by shipping of hard drives) or only available to selected researchers under a non-disclosure agreement. The size of the document collection is of concern because it should be suitable for exploratory search, which only works if the available documents are diverse enough that the user has the chance to find more relevant answers, even after several query iterations.

A source that fulfills these requirements is the arXiv repository. All papers submitted to arXiv are freely available to the public and data dumps of the entire document collection (about 1.2 million papers as of 2017) are available from Amazon S3 through a payment model where the requester has to pay a certain fee per GB of bandwidth (about US$ 0.09 as of 2017). While the document collection cannot be retrieved for free, the total cost of less than US$ 100 should be affordable for inclined researchers and the process to get the data is relatively simple. In the most prominent categories, the number of submissions is probably large enough to give a representative sample of the research done in this field, which makes it suitable for exploratory search.

4.1.2. Data Preparation

For the prototype of the search engine, we have built a full-text index of all papers submitted to arXiv until March 2017. It consists of 1,256,873 documents, which account for 737 GB of PDFs or about 38 GB of plain text in utf-8 encoding. To make this raw data usable by the search engine, we have created the *index-builder*, a set of tools written in Python that contains:

- a PDF parser that extracts the content of all papers as plain text
- a crawler for arXiv's metadata-API that allows us to retrieve information that is not part of the PDFs
- a topic model generator that creates hierarchical topic models based on document's metadata and full-text
- a script that initializes and populates the actual search index

Source code and usage instructions for the *index-builder* can be found in Appendix A.1.

PDF Parser

The parser we used to extract plain text from PDF files is based on the *pdfminer* library [@Shi] which allows to process about 25 documents per minute and physical core on a contemporary desktop CPU, based on experience gained during our own effort to build a search index of the arXiv repository. Because parsing all PDFs can take a few days on a single CPU, the script provides a progress logging system that allows to interrupt and resume the task at any time, and there are various built-in options for graceful error-handling.

Extracting text from PDF files is rather slow and not very reliable, as our experience with this task suggests. This is not just an issue of the parsing library that was used in this work (we have tested several other PDF parsers before pdfminer was chosen), but generally a result of the complexity of the Portable Document Format and the diversity of tools that generate PDF documents. Consequently, about 1.5% of documents could not be parsed at all and there are several documents whose plain text representation is corrupt. Still, the parser was able to extract a decent plain text representation for more than 95% of the documents, which is well enough for the search engine, given that we have at least some metadata (including title and abstract) as a fallback for the remaining documents.

Metadata Crawler

arXiv provides metadata for its records through a public API which conforms to the *Protocol for Metadata Harvesting of the Open Archives Initiative* (OAI-PMH) [@LVN$^+$]. This allows us to reliably retrieve information like the title, abstract and list of authors that would otherwise have to be extracted from the PDFs, as well as some additional metadata like the date of submission or a DOI identifier. Because it is not possible to just download a file that contains the records of all documents, the metadata crawler was created, which requests the records chunk by chunk over OAI-PMH, parses the responses and stores all relevant information in a JSON file.

Part of the metadata is also a reference to arXiv's category system, which could be used as an alternative to a topic model generated by an algorithm. We intentionally avoided to use this system however, because the goal of this work is to build a search engine that can work with any document collection and also with combinations of different document collections. To rely on data that is only offered by a specific provider would invalidate this goal. Apart from that, experience we have gained during previous work with arXiv data indicates that this category system is probably not suitable for topic-based document retrieval, as the classification of a paper (which is specified by the submitter) is often too unspecific and not consistent with the classification of other documents.

Topic Modeling

The topic model for the arXiv document collection was built with the help of *gensim* [@Řeh], a Python library that implements a variety of topic modeling algorithms. Originally, we had intended to use the hierarchical topic model nHDP [PWB+12] which was presented in Section 2.1.2, but we were unable to find a viable implementation of the algorithm. To the best of our knowledge, there is currently only one implementation of nHDP available, which is the reference implementation by John Paisley [@Pai], co-author of the nHDP paper. Unfortunately, this implementation is written in Matlab, which makes it hard to integrate with the other components in the tool chain, and it comes with no documentation or usage instructions whatsoever, which is why all attempts to work with this implementation have failed.

Gensim offers a decent implementations of a variety of topic modeling algorithms like LSA, LDA and HDP, but they all lack support for topic hierarchies. Therefore we have implemented a meta modeling framework that allows the use of arbitrary topic modeling algorithms to build a topic hierarchy. It works by first building a (flat) topic model from a set of documents and then using that model to add topic labels to each document. Next, we build a new topic model for each sub-topic with only the documents that are part of that topic. This process is repeated recursively to get topic models of arbitrary depth. If we arrange the computed topic models in a tree structure, we can use it to classify previously unseen documents in one or more of the existing classes, though we won't be able to add new classes (this would require a re-computation of all models).

We have tested the meta modeling framework with gensim's implementations of LDA and HDP on the arXiv document collection. The results using HDP were rather disappointing, given that nHDP would have been the preferred algorithm. The main issue with gensim's implementation of HDP was a very uneven topic distribution. No matter how many topics the algorithm was allowed to create, usually more than 90% of the documents would be assigned to the first topic. Even in a nested topic model, this meant that most of the documents were assigned to the same topic, which makes the model rather meaningless.

The implementation of LDA did not suffer from this issue, which resulted in a much more even distribution of topics and many relatively small groups of a few hundred documents each at the lowest level of the topic hierarchy. We have built two LDA topic models: one based on the parsed plain texts of the PDFs and one that is based on just the title and abstract of each document. While reviewing random samples of topics from each of the two models, it became apparent that the topic model generated from just title and abstract created groups of documents that were more closely related and contained fewer outliers from other fields. We assume that this difference is partly caused by PDF parsing issues, but also influenced by the distribution of relevant terminology in the abstract of a documents, which is usually less noisy than the rest of the document when it comes to the use of distinctive terminology.

To increase the quality of the topic model, two additional measures have been implemented: A language detector was added and lemmatization was applied to all text that is used for topic modeling. The language detector was used to filter out all non-English documents before the model was built. Mixing several languages in a single topic model only blows up the terminology it has to track and causes documents of different languages to be put in different topics, which is a classification that can be solved far more reliably by a language detection algorithm. Removing documents in other languages from the topic model is therefore an important step to improve the quality of the topic model and the negative impact due to missing class labels for documents in other languages in negligible in the case of arXiv, because almost all submissions are in English and multi language support is not in the scope of this work.

Lemmatization was added to further reduce the dictionary size and to allow the topic modeling algorithm to match terms that appear in different inflections. This is a problem that is often solved with stemming, which is computationally much cheaper, but in the case of topic modeling it comes with a disadvantage: each topic is identified by a list of significant terms, which can help us to understand what the topic is about; when stemming is applied, the resulting tokens may be incomprehensible, which is also an issue for the prototype, because we display some of these terms to the users of the search engine (e.g. in the topic centroid visualization; see Section 4.3). Lemmatizing the contents of more than a million documents used to be a costly task, as it requires part-of-speech tagging and in some cases syntactic parsing of sentences, but there have been great advances in NLP software in recent years. For this purpose we use spaCy [@Hon], a highly efficient NLP library which allowed us to calculate the lemmas for the contents of all arXiv documents on consumer hardware in a matter of hours.

The topic model that was finally used in the search index consists of 8968 topics in a four-layer topic hierarchy. The number of topics per layer was limited to 5 for the first layer, 10 for the second and third layer and 30 for the fourth layer. Generating sub-topics was interrupted if there were less than 400 documents left in a branch, and the number of sub-topics to generate was additionally limited to the number of documents in that branch divided by 200. With these settings, a well-balanced hierarchical topic model could be built that would provide a suitable classification of the arXiv document collection.

Index Builder

The last step in the pre-processing pipeline is to create and populate the search index that will be used by the search engine. For this purpose we use Elasticsearch [@Ban], a distributed information retrieval system based on Apache Lucene that solves most of the standard IR tasks that lay the foundation for a session-based search engine. The prototype is not strictly dependent on Elasticsearch as information retrieval system; alternatives that would likely be suitable to achieve similar results include Apache Solr [@Fou] or Sphinx [@Aks], but as these systems do not provide a common interface, a change of the search backend would come at a cost.

The index builder creates a new Elasticsearch index with a custom mapping that defines all fields we want to store, as well as their data types and additional indexing instructions. Then the documents are streamed from the files that were created in the previous steps (full-text, metadata, topic model) and added to the index. The final search index with all 1.25 million documents has a size of about 32 GB.

4.1.3. Search Index

Elasticsearch was selected as information retrieval system for a variety of reasons: It is free software and completely self-contained, which makes it easy to set up and configure. Its horizontal scalability allows us to handle way more data and requests without the need to change algorithms or the data model. The search performance of Elasticsearch is well enough that it can handle the arXiv index on a single node, if it comes with a fast mass storage medium (e.g. a SSD) or enough RAM to store the relevant parts of the index (for a detailed performance review, see Section 4.4). The rich query syntax of Elasticsearch allowed us to implement the search algorithms we described in Chapter 3 within the Elasticsearch ecosystem, which is an important prerequisite if we want to achieve near linear scalability in a distributed setup.

Indexing

The search index consists of a set of documents whose structure is defined by the *mapping* of the index. The mapping specifies which fields can be stored by each document and how each field is indexed. Each field has exactly one data type, but its contents can be indexed using several different methods. There are two basic options for indexing: *raw* and *analyzed*. A raw field stores data in a set structure which only allows for efficient lookup of exact key matches, while the analyzed form enables us to efficiently filter or search the contents of the fields, depending on the data type and *analyzer* that has been used. If a text field is indexed in analyzed form, Elasticsearch creates a document-term-matrix of all documents that have stored some data in that field. The matrix is actually stored in a more memory-efficient data structure in an *inverted index*, which allows us to run full-text queries on the entire document collection and retrieve a ranked list of search results. So while the logical view from outside suggests that documents give access to their fields, the actual implementation is reversed: each field is stored in its own index structure where each entry has a reference to its source document, and by filtering and ranking the contents of one or more fields, Elasticsearch gets a list of document identifiers which are then used to access the raw document data.

Analyzers enable us to define what kind of pre-processing is applied to the contents of a field before they are stored in the index. A text analyzer typically consists of tokenization (splitting a string of text into individual terms) and several normalization steps that concern character encoding, letter case, punctuation and other properties of the text. The standard text analyzer of Elasticsearch splits strings on word boundaries, removes most of the punctuation and converts all terms to lowercase, which is a decent approach that works with documents in many languages. But Elasticsearch also offers a specific analyzer for the English language, which additionally removes a set of common stopwords and transforms words to their base form using the Porter stemming algorithm [Por80]. These measures help to reduce the size of the search index and increase the recall for text search while only slightly reducing precision (e.g. in cases where stemming incorrectly maps two different words to the same reduced form).

This was a short introduction to the core concepts of Elasticsearch that are needed to understand how the search index and the prototype were designed; for more information, please refer to *Elasticsearch: The Definitive Guide* [GT15].

Field	Type	Analyzers
id	keyword	raw
url	keyword	raw
title	text	default, english, raw
authors	text	default, raw
release date	date	default
abstract	text	default, english
contents	text	english
topic	nested	
id	keyword	raw
score	float	default
layer	integer	default

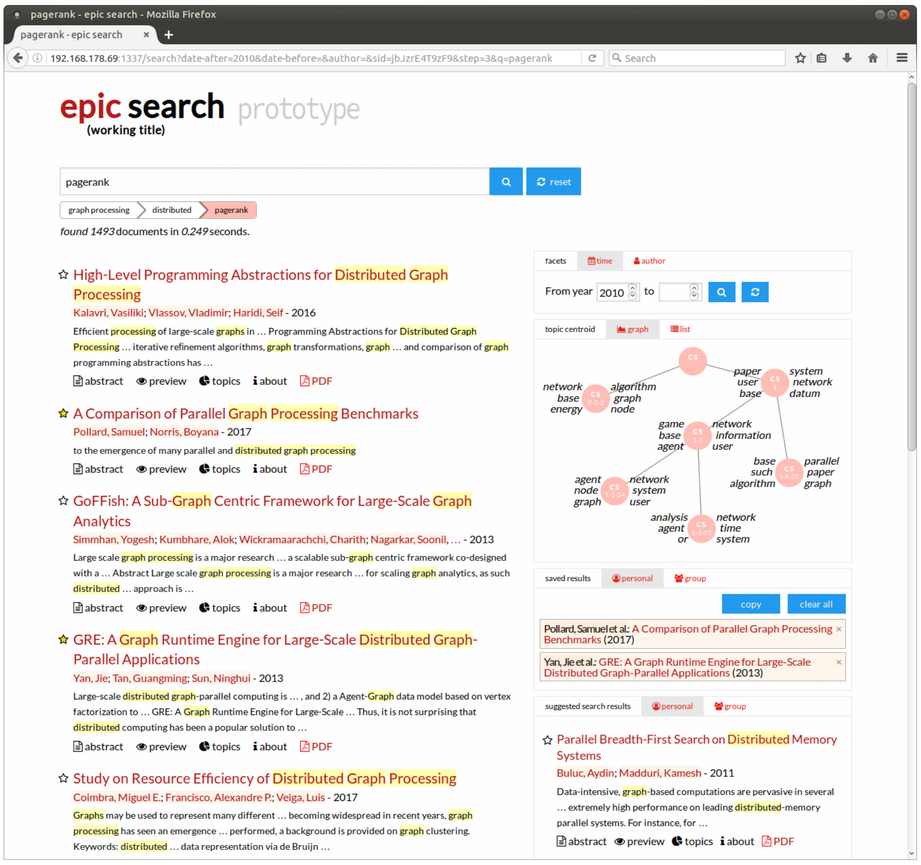
Table 4.1.: Fields, types and analyzers in the search index for the arXiv document collection. Some optional fields have been omitted for clarity.

Schema

For the search index we have built, the mapping consists of the fields that are shown in Table 4.1. The title of each document is a text field that is stored using three analyzers which match different kinds of documents in different search scenarios: The default analyzer helps us to find document titles in any language, while the English analyzer matches English document titles even when the inflection of the search terms is different. The job of the raw analyzer is to boost the match score in case the user enters the exact title of a document. The authors field follows the same pattern, only that we decided to skip the English analyzer here, because language-specific functions like stemming and stop word filters do more harm than good when applied to names. The authors field actually consist of a list of strings, but due to Elasticsearch's strategy to store each field in a separate index structure, there is no need to explicitly define a set data type; the authors of all documents are added to the same index and each author entry simply maps to its source document.

For the document's abstract, the raw form is not required, but it can be useful to apply the default analyzer in addition to the English analyzer to increase the recall, especially when the document was not written in English. For the content field, which contains the complete text of the document, we decided to only use the English analyzer, because additionally applying the standard analyzer would substantially increase the size of the index – and therefore the hardware requirements – for little to no gain.

Some additional metadata is stored for each document, like its ID or an URL where the original document can be retrieved. Those are text fields that use the raw analyzer, because being able to search these fields is not required (the *keyword* type is just a shorthand for text with only the raw analyzer applied). The release date uses the special *date* data type, whose default analyzer allows us to efficiently sort and filter documents by this property. The last field contains the list of topics that have been assigned to a document by the topic model. This field is of type *nested*, which in this case consists of three other fields: topic id, score and the layer in the topic hierarchy.



found 1493 documents in *0.249* seconds.

☆ High-Level Programming Abstractions for Distributed Graph Processing
Kalavri, Vasiliki; Vlassov, Vladimir; Haridi, Seif - 2016
Efficient processing of large-scale graphs in … Programming Abstractions for Distributed Graph Processing … iterative refinement algorithms, graph transformations, graph … and comparison of graph programming abstractions has …
abstract preview topics about PDF

★ A Comparison of Parallel Graph Processing Benchmarks
Pollard, Samuel; Norris, Boyana - 2017
to the emergence of many parallel and distributed graph processing
abstract preview topics about PDF

☆ GoFFish: A Sub-Graph Centric Framework for Large-Scale Graph Analytics
Simmhan, Yogesh; Kumbhare, Alok; Wickramaarachchi, Charith; Nagarkar, Soonil, … - 2013
Large scale graph processing is a major research … a scalable sub-graph centric framework co-designed with a … Abstract Large scale graph processing is a major research … for scaling graph analytics, as such distributed … approach is …
abstract preview topics about PDF

★ GRE: A Graph Runtime Engine for Large-Scale Distributed Graph-Parallel Applications
Yan, Jie; Tan, Guangming; Sun, Ninghui - 2013
Large-scale distributed graph-parallel computing is … , and 2) a Agent-Graph data model based on vertex factorization to … GRE: A Graph Runtime Engine for Large-Scale … Thus, it is not surprising that distributed computing has been a popular solution to …
abstract preview topics about PDF

☆ Study on Resource Efficiency of Distributed Graph Processing
Coimbra, Miguel E.; Francisco, Alexandre P.; Veiga, Luis - 2017
Graphs may be used to represent many different … becoming widespread in recent years, graph processing has seen an emergence … performed, a background is provided on graph clustering. Keywords: distributed … data representation via de Bruijn …

Pollard, Samuel et al.: A Comparison of Parallel Graph Processing Benchmarks (2017) ×

Yan, Jie et al.: GRE: A Graph Runtime Engine for Large-Scale Distributed Graph-Parallel Applications (2013) ×

☆ Parallel Breadth-First Search on Distributed Memory Systems
Buluc, Aydin; Madduri, Kamesh - 2011
Data-intensive, graph-based computations are pervasive in several … extremely high performance on leading distributed-memory parallel systems. For instance, for …
abstract preview topics about PDF

Figure 4.1.: Screenshot of the prototype's search engine results page

The specified indexing options allow us to efficiently access all documents that are about a specific topic and to filter the matched documents by their topic score, which is crucial for efficient topic-based search.

4.2. Search Engine

In this chapter we describe the implementation of the search algorithms and the remaining parts of the query processing pipeline that are necessary to answer a user's search request. This is a direct implementation of the concepts described in Section 3.3. The output of this process is a search result page similar to the one depicted in Figure 4.1, which contains the list of search results for the query in the context of a search session, as well as a visualization of the most important topics in the session and additional search results that are primarily based on the state of the topic centroid. A detailed review of the user interface will be given in Section 4.3; the focus of this segment is to present the steps that are required to calculate the results that will be displayed on the search result page.

The search engine is built as a lightweight web service written in Python [@Ros], which provides a HTTP interface that can answer search requests with human-readable HTML or machine-readable JSON documents. It handles search requests and manages session state, but the actual search algorithms are implemented in Elasticsearch's Query DSL, a declarative language that allows us to search and filter documents on

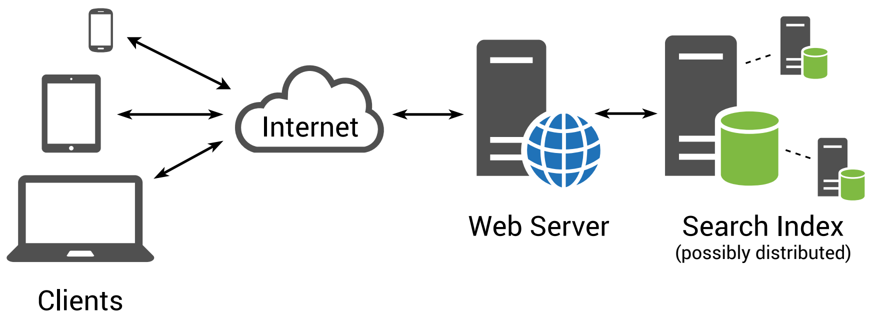


FIGURE 4.2.: Network architecture of the search engine

distributed search indices. A reference to the full source code of the search engine can be found in Appendix A.1.

Figure 4.2 shows the network architecture of the system: A web server manages all user sessions and generates the expanded search queries. The queries are then processed by Elasticsearch, which can run on the same machine as the web server or distributed on several other nodes. In both cases, the internal communication between the web service and Elasticsearch is handled via HTTP. Clients never have direct access to the search index; only the web service exposes a port to the public.

4.2.1. Search Algorithms

The search strategy that is deployed by the prototype has already been discussed in Section 3.3.2; here we will provide some more details about the implementation of the search algorithms, especially with respect to Elasticsearch's Query DSL.

Full-text search

As explained in Section 4.1.3, the search index consists of several analyzed fields (title, abstract, content, etc.), where each field contains a term-document matrix which will help us to efficiently match the query terms against the indexed documents. For full-text search, we use two query types: The simple *match* query will match all documents that contain at least one of the query terms and return higher scores, when multiple query terms are found in the same document. The *phrase match* query on the other hand will only match documents that contain the entire search phrase in that specific term order. This type of query cannot be easily answered by a regular term-document-matrix, but Elasticsearch provides some additional data structures that help to efficiently answer phrase queries (for details on this or other topics related to Elasticsearch, please refer to [GT15]). With the *phrase match* query it is not possible to discover more documents, because it clearly matches only a subset of the documents that were already found by the *match* query, but we can use phrase matching to boost the scores of documents that are more likely to resemble what the user is looking for. Furthermore, different weights are applied to matches in different fields: Phrase matches are generally more important than term matches and a match on a document's title is more relevant than a match in the abstract or some other part of the document. Also, matches in multiple fields and multiple matches in the same field result in higher scores.

In the session-based scenario however it is not enough to answer a single query: The entire query history of the user has to be considered as well. Therefore we have implemented the query weight system that was discussed in Section 3.1.2, using Elasticsearch's *boost* function in combination with a *bool query*. This will ensure that documents matched by more recent queries will receive higher scores. Furthermore, a restriction was implemented so only documents that match at least one term of the most recent query will be displayed in the result list. With this restriction imposed, there is a stronger correlation between the search query and the documents that are displayed, and the user gets instant feedback, if the query did not match any documents, instead of being presented with an average of answers to all previous queries.

The strategy to simply concatenate multiple full-text queries may seem rather expensive at first glance, because the cost seems to increase linearly with the length of a user session, but in practice this assumption does not hold: As multiple independent queries are combined in a single request, they can be executed in parallel by the search engine, and the load caused by other parts of Elasticsearch's query processing pipeline (which is far from negligible) remains constant. Also, the number of queries that are actually included in the full-text search has been limited to a fixed amount, so that unusually long sessions will not at some point cause extraordinary load.

Topic search

With topic search, we rely on the topic centroid rather than the query terms to match and rank documents. The functioning of the topic centroid has been described in detail in Section 3.2 and the algorithm for topic search was presented in Section 3.3.2; both were implemented according to these specifications.

To calculate the rank scores for topic search, we cannot rely on a built-in function of Elasticsearch, because we want to derive the match score from the scores in the topic centroid and the topic score of each document. We could retrieve the documents from the full-text search and adjust their scores afterwards, but this would strictly limit our selection to the first few results that have been returned by the full-text search, making topic search a much weaker tool for document selection. Instead, we want the ranking function to be applied to all matching documents before the final document scores are calculated. This is where Elasticsearch's *function score* query comes into play, which allows us to implement the ranking function based on constant values and variables retrieved from matching documents. The score for a document that matches a certain topic is now calculated from the likelihood score of the topic for this document, multiplied by the topic's score in the topic centroid. To match multiple topics from the topic centroid, we define multiple topic queries and sum up the scores of matching documents, as we did with the full-text search.

Combination

To combine the scores of full-text and topic search, we merge those results again and use another function score query to calculate the weighted average of the full-text and topic scores. This however requires us to have a comparable value range, which is not the case for full-text and topic score. Scaling the values by the highest score of each query is also not an option due to the distributed architecture of Elasticsearch, which does not allow us to use the results of aggregation functions at query time. Therefore the current implementation relies on a heuristic approach that uses logarithms with different bases (which were discovered by trial and error) to map the scores to a comparable value range. This solution is far from ideal, because its parameters must be manually adjusted for each search index, but it works well when the parameters are chosen with care.

Filters

To give the user more control about certain aspects of the search results, certain filter functions are available in the prototype as well. These do not add any documents and they do not require any scores to be calculated or merged. Instead, they simply remove documents from the previous selection when they don't match the desired criteria, which in our case can be the date of publication or the name of an author. Filters in Elasticsearch work analogous to regular databases on indexed tables and therefore come at no major computational cost.

4.2.2. Query Pipeline

In this section we will review the concepts of the query pipeline that was already discussed in Section 3.3.3 and adjust it where necessary to the reality of the prototype that we have built. Figure 3.4 shows the query pipeline in its entirety.

Spell Checking

Spell checking relies on the combination of a conventional spelling dictionary as well as terminology lookup in the search index. We use Hunspell [@Ném] as a traditional spell checker, backed by an English dictionary. Only if Hunspell detects a word with incorrect spelling, the search index is used as an additional source of terminology: Elasticsearch's *suggest* feature allows us to search for terms in the index that match the misspelled word up to a certain Levenshtein distance. The search index may of course contain spelling errors and other mistakes, therefore we do not accept very infrequent terms as spelling corrections. So whenever a query term is not found in the dictionary, suggestions from both Hunspell and the index are collected and the correction with the lowest Levenshtein distance to the misspelled term is accepted. This explicitly includes suggestions from the search index with a Levenshtein distance of zero, which means the term was found in the search index with sufficient frequency, although it was not part of the Hunspell dictionary, and is therefore considered to be correct after all.

In the current implementation, corrections with a Levenshtein distance of two or less are accepted, and suggestions from the search index have to occur in at least two documents to be accepted, though all of these values are configurable. Furthermore, when a possible spelling error is detected, the suggested correction is displayed on the search result page, but it does not automatically replace the originally submitted query. This measure was introduced because we were uncertain whether the spelling suggestions would be correct in most cases, but as it turned out during testing, the number of false corrections was almost negligible, so the automatic correction of spelling errors will probably be enabled by default in an upcoming version of the search engine.

Session Update: Request

In the next step, the session of the user that submitted the query gets updated. In the regular case, this just means that the current query is added to the query history. There are however a few edge cases, that have to be considered:

1. Requesting additional search results: whenever the user requests more search results for the same query, this query won't be added to the query history to avoid duplicates
2. Applying filters to the same query: when the results to the last query are filtered by release date and the query remains otherwise unchanged, the updated query will replace its predecessor instead of being appended to the query history
3. Accepting a spelling correction: when a suggested spelling correction is accepted by the user, it replaces the last (incorrect) entry in the query history.
4. Navigation in the query history: when the user navigates to another point in the query history, either by selecting an element in the breadcrumbs bar or by using the browser's navigation function, the query history at this step gets restored before a new query is added (see Section 4.2.3 for details about session persistence).

Main Query: Expansion & Execution

The main query determines the documents that are displayed in the search result list. It is submitted to Elasticsearch which retrieves the ranked result list, possibly by accessing multiple distributed nodes. The response contains the 10 highest scoring documents (or as many as were requested). If however the user does not submit a new query but merely wants to view more results for the same query, the entire query building process can be skipped. Queries that are submitted to Elasticsearch are cached for a while, and so are the results for that query, so when a user wants to see more of them, we simply replay the last query and get the desired results almost immediately.

For the main query, full-text search is preferred over topic search by a factor of two, which results in a rather strong focus on text, but it allows the user to retrieve results that correlate well with the search terms that have been entered. We have selected this factor because it generated promising search results in the test phase of the prototype, but it is freely configurable and an optimization of parameters like these would be an interesting topic for future work.

Session Update: Response

After the results of the search query are available, the next step is to update the topic centroid based on the response to the query. The 10 highest scoring documents are used to identify the most important topics. The implementation of topic identification is very close to the specification from Section 3.2.1, with one exception: Overlaps between related topics have been ignored, therefore no virtual super-topics have been generated for these topics. This decision was made because the topic model turned out to have far fewer overlaps than expected and by only accepting topics in the centroid that do not have any subtopics themselves (i.e. the most specific topics), overlaps that would otherwise be caused by different positions in the topic hierarchy can be avoided as well.

After the most important topics have been identified, the topic shift, as specified in Section 3.2.2, is executed. First, the scores of the existing topics are multiplied by $f_{cooldown} = 0.7$, then new topics are added and the scores of existing topics are combined with the new ones according to the scoring formula with $w_{shift} = 0.4$. Finally, topics with a score of less than 0.1 are removed from the topic centroid, because they don't contribute to the selection of new documents anymore. Again, all of these parameters are configurable; they have been selected because they generated search results that were in conformance with our expectations.

Suggestion Query: Formulation & Execution

The suggestion query defines the results that will show up in the sidebar of the search result page under *suggested search results.* It uses the same query patterns that we have seen in the search query, with a few notable exceptions:

- the topic query outweighs the full-text query by a factor of three
- all filters (time & author) are ignored
- all query terms are optional

The balance between text and topic search is the exact opposite of the previous approach, where the full-text query had far more weight. However, full-text search is still applied for two reasons: It gives a slight boost to documents that match both the topics in the topic centroid as well as the query terms of the user, and it allows us to use Elasticsearch's highlight feature, which extracts text snippets from the matched documents that contain the query terms. Displaying these snippets on the search result page is important for the user experience, even if the document selection scheme is predominantly based on topics and not on keywords.

The suggestion query is submitted to Elasticsearch as a separate query in a new request after the main query was executed and the topic centroid was updated accordingly. This may seem to be a reason for concern, as two requests potentially double the response time. This time, Elasticsearch's caching system comes to the rescue: As we have already submitted a very similar query before this one, large parts of the query are still cached, which causes a considerable reduction in response time of the suggestion query.

Result Page Generation

Finally, the necessary data structures for the search result page are built. The Elasticsearch query was configured to return the relevant metadata with each search hit, as well as a list of text snippets that contain some of the query terms. Based on these snippets, the preview texts are generated that will be displayed on the search result page. Furthermore the visualization of the current topic centroid is prepared, which we will discuss in Section 4.3. After the required data has been prepared, the HTML template of the search result page is rendered and returned to the user.

4.2.3. Session Persistence

In this section we will describe the concept of session persistence, which is the foundation for efficient query history navigation. The original motivation behind query history navigation was to give the user the option to use the browser's built-in navigation function without breaking the work flow and causing unexpected results, if the state of the session on the server side is out of sync with the current page on the client side. This fix for a usability problem was turned into an additional feature that allows the use of a breadcrumb trail for navigation, which gives the user some control over the contents of the query history. We will discuss the usability implications of this and other features in the next section.

Though the prototype has a HTTP interface, it is not an entirely RESTful [Fie00] application, due to the fact that some session data is stored on the server side, referenced by a session ID, which makes it a stateful service. This was an intentional design choice that was made for reasons of performance and especially security, because sensitive data like the last Elasticsearch query is cached in the session and manipulation of this query by a malicious user could have harmful side effects. However, having a stateful web service creates issues like desynchronized state between client and server, if navigation actions are not properly reflected on the server side.

It should be noted that it is possible to design a session-based search engine as a stateless service, by having the user send only the absolutely required session data with each request, like the query history and the topic centroid. This would result in acceptable request sizes and manipulation of the request parameters would not impose a security risk. However, with this approach server load would increase and query response times would suffer considerably, because with each request the necessary data structures would have to be rebuilt on the server side and a complete evaluation of the query pipeline would be required. This negative impact would be most noticeable with common actions like requesting more search results, which benefit most from the server-side cache.

The alternative is to synchronize the state between client and server at each request, which comes at the cost of additional complexity of the server code and requires more memory for each user session, but brings the benefit of lower response times and reduced processing load on the server due to effective use of caching. In the prototype we implemented session persistence by introducing the *step* parameter, which is a number which uniquely identifies each step in a search session. This step parameter is transferred with every request of the user, and in case it is omitted in a request, it defaults to the last step that was stored on the server side. When the user navigates to an older page of a session, the step parameter from that page is transferred with the next request and the state associated with this step can be restored on the server side before the query is executed.

When a request comes in that would result in a new entry in the query history (i.e. not a request to load more results or filter by date), the relevant content of the current session context is persisted and stored under the current step number. The stored data contains a copy of the query history, the terms and scores of the topic centroid and the last Elasticsearch query. Afterwards, a new step number is assigned to the session and the query pipeline is executed as before. If a request with a step number comes in that does not match the last step number of the current session, the session context that was saved under that step number is restored and the request is interpreted in the context of the restored session. The number of steps that can be persisted this way is limited to 20 per session, so that each user can only reserve a certain amount of memory on the server. This of course implies that query history navigation is actually limited to the last 20 steps, but this should cover most, if not all, use cases and make the service more resilient against potential denial of service attacks.

4.3. User Interface

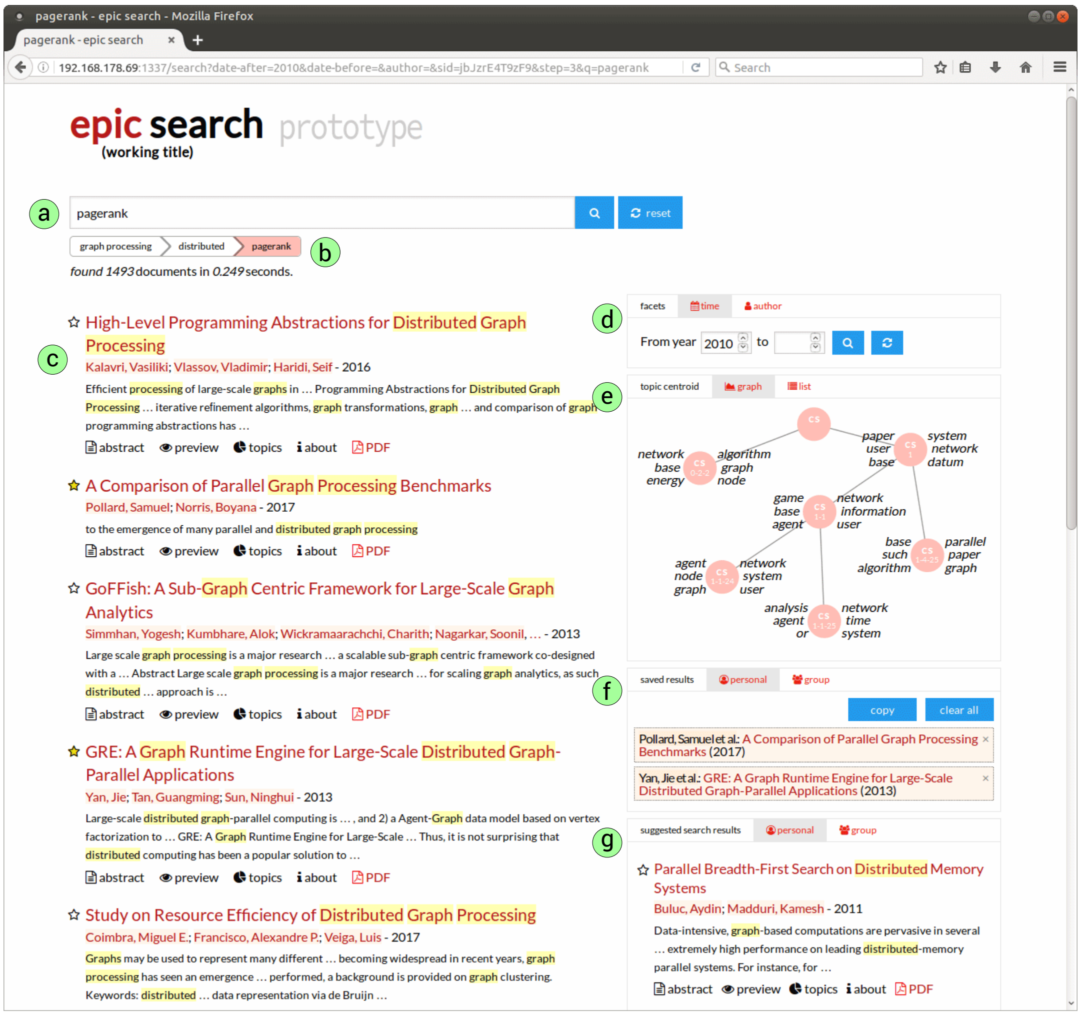
The user interface of the prototype comes in the form of a web application that was built with the help of the web framework *Flask* [@Ron], which allows us to implement all server-side functions in Python, while the contents of HTML pages are controlled with the template engine *Jinja*. The layout of the search engine result page (SERP) [HL09] is in line with contemporary web search engines to make it easily recognizable even for first-time users of the application. It consists of three main elements: The header section with the search box, the search result section on the left and the sidebar with additional information and functions on the right. Figure 4.3 shows the different elements of the search results page, which we will now review in detail.

a) Search Box

The search query input field (a.k.a. search box) allows the users to formulate their search interests. While the search algorithms used by this search engine differ considerably from other search engines, the search box works pretty much the same. The only difference to conventional search engines is the reset button on the right, which allows the user to forfeit the previous query history and to start a new session. Due to the implementation of session persistence, as described in Section 4.2.3, this process is reversible, so the contents of a previous session are not immediately lost if the user wishes to return to the previous state, e.g. by using the browser's navigation buttons.

b) Breadcrumbs

The breadcrumbs trail below the search box visualizes the query history of a search session. Its purpose is to make the user aware, that previous queries are included in the current search effort, and to allow free navigation in the query history. Navigation in the query history is not a requirement to effectively use the search engine, but we decided to still implement it for a variety of reasons: It can help the user to get a feeling for the influence that different combinations of search terms in the query history have on the search results. Also, the option to return to a previous "good" state when the search results do not reflect the users interests anymore can be helpful and may entice users to be more explorative. And some users could be annoyed by the fact that they have no way to remove terms they entered by mistake from the query history, even if they don't impair the search results, which was actually pointed out by participants of the user study.


pagerank - epic search - Mozilla Firefox

pagerank - epic search

192.168.178.69:1337/search?date-after=2010&date-before=&author=&sid=jbJzrE4T9zF9&step=3&q=pagerank

epic search prototype
(working title)

a pagerank reset

graph processing > distributed > pagerank b

found 1493 documents in *0.249* seconds.

c

☆ High-Level Programming Abstractions for Distributed Graph Processing
Kalavri, Vasiliki; Vlassov, Vladimir; Haridi, Seif - 2016
Efficient processing of large-scale graphs in … Programming Abstractions for Distributed Graph Processing … iterative refinement algorithms, graph transformations, graph … and comparison of graph programming abstractions has …
abstract preview topics about PDF

★ A Comparison of Parallel Graph Processing Benchmarks
Pollard, Samuel; Norris, Boyana - 2017
to the emergence of many parallel and distributed graph processing
abstract preview topics about PDF

☆ GoFFish: A Sub-Graph Centric Framework for Large-Scale Graph Analytics
Simmhan, Yogesh; Kumbhare, Alok; Wickramaarachchi, Charith; Nagarkar, Soonil, … - 2013
Large scale graph processing is a major research … a scalable sub-graph centric framework co-designed with a … Abstract Large scale graph processing is a major research … for scaling graph analytics, as such distributed … approach is …
abstract preview topics about PDF

★ GRE: A Graph Runtime Engine for Large-Scale Distributed Graph-Parallel Applications
Yan, Jie; Tan, Guangming; Sun, Ninghui - 2013
Large-scale distributed graph-parallel computing is …, and 2) a Agent-Graph data model based on vertex factorization to … GRE: A Graph Runtime Engine for Large-Scale … Thus, it is not surprising that distributed computing has been a popular solution to …
abstract preview topics about PDF

☆ Study on Resource Efficiency of Distributed Graph Processing
Coimbra, Miguel E.; Francisco, Alexandre P.; Veiga, Luis - 2017
Graphs may be used to represent many different … becoming widespread in recent years, graph processing has seen an emergence … performed, a background is provided on graph clustering. Keywords: distributed … data representation via de Bruijn …

d facets | time | author
From year 2010 to

e topic centroid | graph | list

CS
network base energy CS 0-2-2 algorithm graph node
paper user base CS 1 system network datum
game base agent CS 1-1 network information user
base such algorithm CS 1-4-25 parallel paper graph
agent node graph CS 1-1-24 network system user
analysis agent or CS 1-1-25 network time system

f saved results | personal | group
copy clear all
Pollard, Samuel et al.: A Comparison of Parallel Graph Processing Benchmarks (2017) ×
Yan, Jie et al.: GRE: A Graph Runtime Engine for Large-Scale Distributed Graph-Parallel Applications (2013) ×

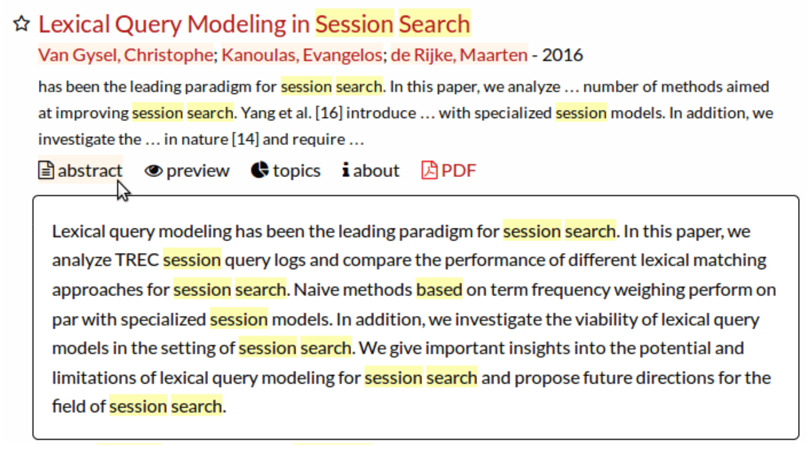
g suggested search results | personal | group
☆ Parallel Breadth-First Search on Distributed Memory Systems
Buluc, Aydin; Madduri, Kamesh - 2011
Data-intensive, graph-based computations are pervasive in several … extremely high performance on leading distributed-memory parallel systems. For instance, for …
abstract preview topics about PDF

FIGURE 4.3.: Screenshot of the prototype's search engine results page

The alternative would have been to not display the query history at all, which would surely reduce some user's urge to over-optimize their query history, but would also make the system less transparent and leave the user with only one option to recover if the search results are getting worse: starting a new session. This however would eliminate the knowledge that the search engine has acquired so far about the user's search intent, which would make the session-based approach less effective and useful. Therefore we decided to give the users the freedom to navigate in the query history, even if this means that some may focus too much on this feature.

c) Search Results

The search result list contains the documents that the search engine deemed most relevant for the user's search query in the current session context. Figure 4.4 shows, how a search hit is formatted on the search result page. The first line shows the title, which links to the source of the document. Next to the title is a star icon which allows the user to set a bookmark for this document in the sidebar when clicked. In the


Lexical Query Modeling in Session Search
Van Gysel, Christophe; Kanoulas, Evangelos; de Rijke, Maarten - 2016
has been the leading paradigm for session search. In this paper, we analyze … number of methods aimed at improving session search. Yang et al. [16] introduce … with specialized session models. In addition, we investigate the … in nature [14] and require …

Lexical query modeling has been the leading paradigm for session search. In this paper, we analyze TREC session query logs and compare the performance of different lexical matching approaches for session search. Naive methods based on term frequency weighing perform on par with specialized session models. In addition, we investigate the viability of lexical query models in the setting of session search. We give important insights into the potential and limitations of lexical query modeling for session search and propose future directions for the field of session search.

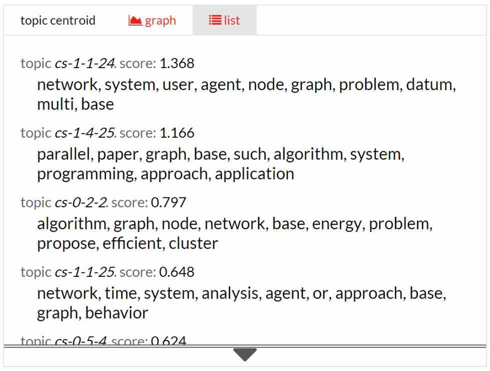
FIGURE 4.4.: A single document on the search result page, with a preview of its abstract

second line the authors of the document are listed, as well as the year of publication. Each author name links to a new query that lists all papers by this author. The following lines consists of a selection of text snippets from the document that match the search terms, highlighted in yellow. In the last line there are five buttons:

- *abstract*: displays the abstract of the paper in a box below the button, with highlighting of all query terms
- *preview*: opens another box that contains more snippets from the document, also with highlighting
- *topics*: lists the topics that have been assigned to this paper, as well as the most significant terms for each topic
- *about*: contains some additional metadata about the document as well as the search result score.
- *PDF*: a direct link to the PDF version of the document, if available

By default, the 10 highest scoring documents are displayed in the result list and more search results can be requested asynchronously by scrolling to the bottom of the search result page.

Compared to other academic search engines, at least two widespread functions are not available on the search result page: exporting citations and displaying other documents that cited this paper. The lack of a citation export function has its root in the nature of arXiv, which is a repository of preprints, i.e. papers that have not yet officially been published in a scientific journal. A citation without a reference to its original publication is usually considered incomplete, therefore we avoided to give the impression that a proper citation is possible, given the information we have. About the issue of determining, which paper has been cited by another: Achieving this goal would only be possible if we had a substantial portion of the world's scientific publications in the index and had the capabilities to parse the publications in a way that allows us to reliably retrieve the bibliography of each paper – none of which is in the scope of this work.


topic centroid graph list

topic *cs-1-1-24*. score: 1.368
network, system, user, agent, node, graph, problem, datum, multi, base

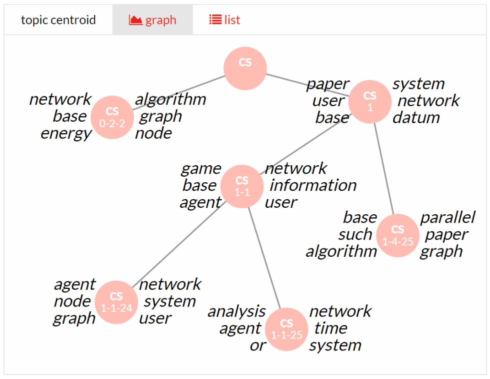
topic *cs-1-4-25*. score: 1.166
parallel, paper, graph, base, such, algorithm, system, programming, approach, application

topic *cs-0-2-2*. score: 0.797
algorithm, graph, node, network, base, energy, problem, propose, efficient, cluster

topic *cs-1-1-25*. score: 0.648
network, time, system, analysis, agent, or, approach, base, graph, behavior

topic *cs-0-5-4*. score: 0.624

(A) The highest scoring topics of the topic centroid, each with its most distinctive terminology



(B) Visualization of the most relevant topics and their defining terminology in a tree structure

FIGURE 4.5.: Visualizations of the topic centroid

d) Filters

The first element in the sidebar is the filter box, which allows the user to restrict the currently search results by release date or author. When a filter is applied, a new Elasticsearch query gets executed, but as long as the query terms remain unchanged, no new elements will be added to the query history in order to avoid duplicate entries.

e) Topic Centroid

In this sidebar element, the highest ranking topics of the topic centroid are visualized. The purpose of this element is to make the user aware of the semantic capabilities of the search engine and to give some insights into which topics were selected to be most relevant in the context of the current search session.

There are two visualization options: *list* and *graph* (see Figure 4.5). In the list view, the 10 highest scoring topics are displayed, along with the most significant terms for each topic. In the graph view, the topics are arranged as nodes in a graph, where each topic node is surrounded by its most distinctive terminology. The edges indicate relationships between topics in the topic hierarchy, and the position of the node on the vertical axis indicates its level in the topic hierarchy. So when viewed from top to bottom, the graph actually resembles the tree structure of the underlying hierarchical topic model.

The graph is an implementation of the concept described in Section 3.2.4 *Topic Graph Visualization*, with the limitation that user interaction with the graph is not part of this prototype. Other than that, the implementation is very close to the concept, e.g. concerning the question how the nodes that are to be visualized are selected, which parent nodes are chosen and on what basis the connections between the nodes are drawn. In future work, the search results could be updated based on the changes to the topic centroid that are caused by the user's interaction with the graph. In the context of collaborative search, a separate visualization of the shared topic model could help the user to get an idea of what the group is currently searching for and incorporate these topics in her own research.

Our own assessment of the topic visualization is that it is not very useful to the end user in its current form for two reasons: it is not easy to understand what a topic is about based on the terms that are displayed and the visualization seems rather complicated. Still, it is a practical tool to understand the influence of the topic model which was especially helpful during the development of the prototype and the suggestions for future work may yet turn this feature into a viable addition to the search engine.

f) Bookmarks

The documents that the user has bookmarked by clicking the star button next to the title will appear in this box. Each bookmark shows the paper's first author, its title, which links to the document's source page, and the year of publication. The documents in the bookmark list can be reordered by drag-and-drop, individual items can be removed by clicking the cross icon on the right and all items can be removed by clicking the *clear all* button. The *copy* button copies the current bookmark list in the specified order into into the user's clipboard, formatted as table, so it can be easily inserted into a spreadsheet.

The saved documents are stored on the client side, in the browser's IndexedDB, which makes it independent of the current session and persistent even if cookies and cache are cleared. This makes the bookmark storage as reliable as it can get without the need to save them in a user profile on the server side. As already illustrated in Section 3.1, the implementation of a user profile was intentionally avoided to show that for the effective use of a session-based semantic search engine, no long-term profile information about the user is required. Still, as soon as collaborative search is involved, there will be a reason to have user profiles and then server-side storage of bookmarked documents could also be added for convenience.

g) Topic-based Suggestions

This sidebar element contains a list of search results that are predominantly selected by their similarity to the most relevant topics in the topic centroid of a search session. The latest query terms and the other terms in the query history have only little influence on these search results, but highlighting of text is still enabled to make it easier to spot relevant documents. The purpose of this approach is to suggest documents that are relevant for the user's current research, but do not necessarily match the keywords of the current or past queries, by harnessing the full potential of the topic model.

The layout of the list of suggested search results is very similar to that of the regular search results. The functions are all the same, but the allowed length of the text snippets has been reduced to make them take no less than three rows, given the lower width of the sidebar. Also, the margin between the buttons in the last row has been reduced. The search engine does not allow any overlap between the main search result list and the suggested search results; duplicates are removed from the suggested search results.

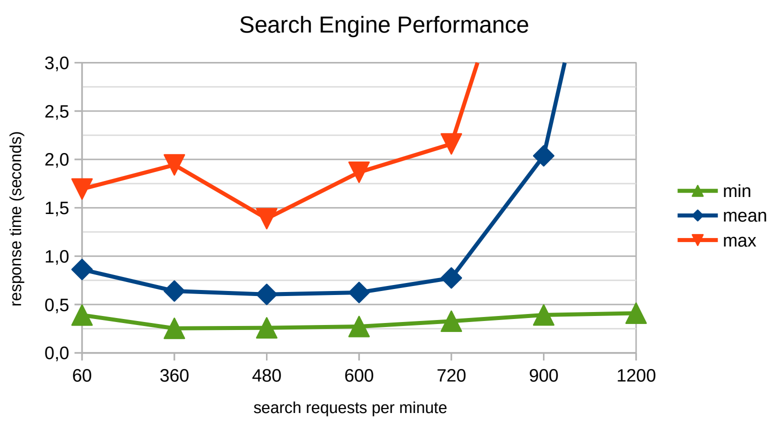


FIGURE 4.6.: Average response time relative to the load on the search engine (number of requests per minute)

4.4. Performance Review

We have conducted a short performance review of the prototype in order to investigate, whether the query response times are within acceptable bounds and what hardware is required in dependence on the number of concurrent users of the system.

The performance test has been performed on a single machine equipped with an Intel Xeon E5-1650 v3 CPU (6 physical cores clocked at 3.50GHz), 64 GB DDR4 RAM and a Samsung SSD PM863 with 480GB of storage space. Before each test, we restart Elasticsearch and the web service and run a few queries to ensure that the relevant index structures were transferred from disk to RAM. During each test, we generate random search requests at a fixed frequency for a time span of 60 seconds. The queries are generated by randomly selecting and combining terms from a pool of technical terminology, which makes it highly unlikely that the same query is submitted twice in a test scenario, therefore benefits from query caching are negligible. For each submitted request, the search engine executes the query pipeline as usual and returns the search result page. We measure the time that passed from the point where the request was sent until the complete HTTP response has been received. The same test is repeated under different server load scenarios.

Figure 4.6 shows the results of this test. In a low load scenario with just 60 requests per minute, the average response time was at 0.86 seconds, with 1.7 seconds in the worst case scenario. This performance remains relatively stable up to 600 requests per minute, with an even lower average of 0.62 seconds. At 900 requests per second, performance starts to degrade and the average response time increases to 2.03 seconds, with sporadic requests taking up to 5 seconds. At 1200 requests per minute, performance degrades fast and requests start to time out, because the search engine cannot process enough requests to keep up with the demand.

The overall performance of the prototype is almost entirely dependent on the search performance of Elasticsearch, which was confirmed by the distribution of CPU times between the two processes that are involved. The Python application which manages the query pipeline and acts as a web server usually claims less than 3% of the total processing time. This means that due to Elasticsearch's horizontal scalability, sufficient performance can be achieved even for large document collections and high load scenarios if enough hardware is added.

We have found considerable differences in throughput and response time when a customary hard disk drive is used instead of a solid state drive. The response times were generally above 5 seconds and performance started to degrade at just 100 requests per minute when the search index was stored on a magnetic disk. This issue could be mitigated by loading the entire search index into memory, but even so the average response times were comparable, but not better than in the scenario where the index was stored on the SSD and only 4 GB of memory were assigned to Elasticsearch. Apparently, adding more RAM to the system does not reduce query response times or increase throughput, as long as the search index is stored on a sufficiently fast SSD.

Besides random queries we have also tested the performance in the context of search sessions. This measurement showed that there is no substantial difference between the response time of the first query and any of the following queries in a search session. This may seem surprising, because the more queries are submitted to the same session, the more terms from the query history must be matched by the search engine. But apparently the bottleneck for query response times is not the number of terms in a search request.

The results of this performance review allow us to estimate the number of users our test system could serve: If we assume that a typical user does not submit more than one new query within a 10 second interval, the search engine can handle at least 120 users working at the same time on this system. This number is probably a conservative estimate, because in a realistic scenario, users often submit requests that can greatly benefit from caching (like requesting more results for the same query or changing the filter settings) and it can be assumed that the average user spends more than 10 seconds to review the documents on the search result page and then to formulate a new query. At the very least, this review has shown that the overall performance of the prototype is more than sufficient to have a small group of people working on the same system, which was an important prerequisite for the user study we will present in the next chapter.

4.5. Discussion

We have built the prototype of a literature search engine which implements the concepts of session-based search and topic search. The search engine is backed by the information retrieval system Elasticsearch, which we used to index the full text and metadata of 1.2 million documents from the arXiv repository. A web application presents the search results as well as some more advanced features of the search engine in a clear and concise fashion.

Search sessions have been successfully implemented, following the specification in Chapter 3 in a way that a decent balance between the influence of the query history and the latest query terms was achieved. The topic centroid was also implemented as laid out in Chapter 3 and both the quality of topic-based search results as well as the query performance were good enough that topic search was added as an influence factor to all regular search queries. Although collaborative search is not part of the current implementation, the prototype was designed with this concept in mind: Collaborative features can be added without major changes to the existing code base.

At first glance, the web interface provides the familiar experience of a literature search engine with a few usability enhancements, but on closer examination it offers a variety of features that are only realizable with a session-based semantic search engine. The suggested search results in the sidebar are a direct result of the topic centroid that is calculated for each search session, as is the visualization of the most relevant topics. The impact of the search session on the search results is clearly visible after the second query in a session, when not just the terms of the last query are highlighted on the result page.

Our performance review has shown that the prototype has a sufficiently low query response time of less than one second on a typical server system, provided that the search index is stored on a solid state drive or in memory. The decent amount of simultaneous requests our test system could handle, combined with the seamless horizontal scalability of Elasticsearch, characterize this prototype as a practicable search engine despite its early stage of development.

Nevertheless, a variety of well-intentioned features and low response times do not necessarily make a great search engine. In the next chapter, we will try to investigate, how the prototype is perceived by a real audience working on different literature research scenarios.

Chapter 5

User Study

We have conducted a small-scale, qualitative user study to investigate whether a session-based semantic search engine is suitable for literature research and how well the prototype that was created as part of this work can be used for this purpose. To this end, we let the participants carry out several literature research tasks with the prototype we described in the last chapter and a conventional academic search engine, namely Google Scholar [@Goo]. The results from the comparison of the two search engines should contribute to the answering of the following research questions:

- Are users able to find relevant search results using a session-based semantic search engine?
- How is the quality of search results in terms of precision and recall, compared to the conventional search? Are users satisfied with the results of their literature research?
- How are the semantic capabilities of the search engine perceived? Does the search engine suggest relevant documents that would otherwise not have been found based on the user's query terms?
- How is the session-based approach perceived? Does the inclusion of previous search terms into the expanded query contribute to the quality of the search results?
- What is the user's perception of the usability of the prototype compared to the competing product?

The participants fill in questionnaires which measure their satisfaction with the search engines they have used. In addition, the results of each participant's literature research are saved. Based on these data sources, we intend to find answers for the research questions.

5.1. Methods

During the study, each participant works on two different literature research tasks. Both search engines rely on the arXiv document collection (the prototype by design, Google Scholar will be restricted to the same data source), which offers access to documents in a limited amount of disciplines. Therefore, research tasks have been created for two of these disciplines: physics and computer science. The topics have been compiled in a way that only basic knowledge of the field is required to solve a task, so that differences in the participants background knowledge will not cause major differences in their performance. For each discipline, two different tasks have been created. In Appendix A.2.1 a full transcript of each task can be found.

In a study session there are two passes under equal conditions: One with the prototype and the other with Google Scholar. The order of the passes is randomized among all participants, to mitigate the influence of learning effects in the course of the study. Participants are subdivided in two groups: those working on computer science tasks (CS) and those working on physics tasks (PH). Each member of a group participates in one of four possible configurations: they either start with Google Scholar (G) or the prototype (P), and they either get task A or task B in their respective fields as their first assignment. The configuration of each participant can be expressed as a tuple defining the search engine and task the participant starts with. For example, (G, A) would mean the participant starts with Google Scholar and task A, from which we can infer that the second assignment of the participant would be (P, B), which stands for the prototype and task B.

5.1.1. Procedure

At first the participants are asked to complete a demographic questionnaire, after which they get some information about the purpose of the study. They are informed that they will be solving two literature research tasks under equal conditions, each one with a different search engine. The procedure for a single pass is as follows:

- Introduction: There is a brief introduction to the core functions of the search engine by the supervisor of the study, so the participants can familiarize themselves with the search engine they are about to use.
- Reading phase (5 minutes): The task sheets are distributed and the participants familiarize themselves with their literature research task.
- Research phase (12 minutes): The participants use the search engine that they have been assigned to find relevant documents in order to solve their task.
- Rating phase (5 minutes): The participants review the search results they've found and rate each result on a scale from 1 (not relevant) to 5 (highly relevant).
- Survey: The participants complete the task review questionnaire.

The total time for a session is expected to be about 60 minutes.

5.1.2. Implementation

Participants get to use a desktop computer or a comparable device with a keyboard and mouse. All participants use the same web browser to access the search engines, which in this case was version 60.0 of the Chromium browser. The workplaces of the participants are arranged in a way that they are not encouraged to look into the screens of other participants or to share information while they are working on the task.

In order to make the results of both search engines comparable, the available documents must be restricted to the same source. The prototype has access to all documents in the arXiv database, which consists of about 1.2 million documents as of 2017. This however is only a tiny fraction of what Google Scholar has in its index.

This disparity can be mitigated by restricting the search results of Google Scholar to a specific domain, using the *site* filter. By appending `site:arxiv.org` to every query, Google Scholar will only return documents found on the arxiv.org domain. We have searched for a random selection of documents using this filter and based on our observations we are fairly certain that Google Scholar has a complete and up-to-date index of the documents hosted by arXiv. Still, having to enter the site filter after each query would reduce the usability of Google Scholar and likely be a common source of error among participants of the study. That's why a browser script has been written as part of this work (see A.2.2 for the source code), which will add the site filter to each query that gets submitted by the user and removes the filter from the search box again after the results have been loaded. With these measures in place, both search engines can operate on the same document collection without imposing noticeable usability restrictions.

In addition to evaluating the answers from the questionnaires we intend to gain insights into the search performance of the participants by reviewing the results of the literature research tasks. In order to rate performance, we first have to define how we are going to measure relevance. Discovering all relevant documents in advance and rating them accordingly is not a feasible solution, because there are usually too many sources that could be at least somewhat relevant for a non-trivial question than any one person can review. Therefore we don't rate any documents in advance and instead let the majority of the participants decide, which documents are relevant. After their research, participants rate all documents they have selected from 1 (not relevant) to 5 (highly relevant). By aggregating the results of several participants, we can compile a list of all documents that got at least two ratings and define the average of these scores as the relevance of that document. Then we can compare the results of each user to this aggregated list and measure, how many of the relevant results were found and how far the participant's rating was off the average. Identifying patterns between different groups of participants, especially if they depend on the search engine that was used, can help us to answer some of the research questions.

In practice, the participants will have to save their results somehow and then decide, which rating to give. This can be done by having the participants copy their search results into a spreadsheet and then leave a rating in the cell next to each selected document. With Google Scholar, this is done by clicking the "cite" link on a search result and copying the citation in a spreadsheet. Users of the prototype can use the built-in bookmarking function (see 4.3f) to save search results and then copy all results into the spreadsheet. In both cases, the spreadsheets are already prepared on the participant's computers so they can easily switch between browser and spreadsheet.

The sessions of the study should be arranged in a way that all participants can get the same introduction to each search engine. It is advisable to have all participants in a session start with the same search engine and then have multiple sessions with different participants to balance the configurations. Also, we recommend to have the same instructor in each session to visually explain the search engine to all participants (e.g. using a video projector), so that everyone gets the same chance to understand the basic functioning of each search engine.

5.1.3. Tasks

The research tasks were designed in a way that allows the participants to explore a complex, versatile topic where they can select the most promising documents from a large variety of titles. Working on tasks that cannot be answered with a single search query is crucial if we intend to measure the impact of the session-based and semantic features of a search engine.

While the topic of a task has to be from one of the research fields that is available in the arXiv data set, it is not helpful to cover highly specialized topics that require lots of background knowledge. Covering topics that can be understood with basic knowledge in the field helps to eliminate differences in participant's search performance that are caused by their educational background and it expands the pool of possible participants.

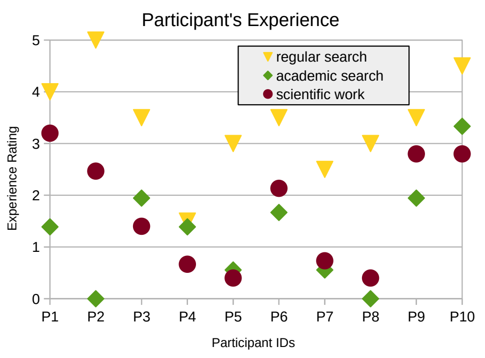
To increase the interest of the participants in their work, all tasks were designed as *simulated work task situations* [BI97], which is a technique that puts the task into a realistic setting that the participants can identify with. A transcript of all tasks that were used during the study can be found in Appendix A.2.1.

5.1.4. Questionnaires

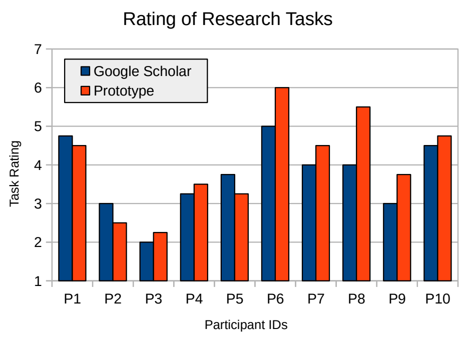
Two questionnaires have been designed for the study: the demographic questionnaire, which asks for some information about the participant's background, and the task review questionnaire, which is filled after each research task. A copy of both questionnaires can be found in Appendix A.2.3.

The purpose of the demographic questionnaire is to discover variables in the participant's background that could influence performance during the study. Therefore the form contains typical questions about age, gender, proficiency in the English language, formal education and work experience in fields related to the research tasks. Another important influencing factor is the participant's experience with search engines in general and academic search engines in particular. Because self-assessments are often not comparable between different participants, we decided to derive experience from objectively measurable values like usage frequency of search engines, usage of advanced functions that search engines typically provide and awareness of common academic search engines. Furthermore, we asked about experience with scientific work, as it is to assume that participants who are adept in writing scientific documents are likely to have some experience in literature research.

The goal of the task review questionnaire is to gain insights into the participant's perception of the overall usability of each search engine as well as the assessment of specific features. To evaluate the usability of the search engine, we use the system usability scale (SUS) [Bro$^+$96], which was shown to be a valid and reliable tool to measure and compare usability between different systems [BKM10]. The SUS consists of ten statements which the participant assess on a five-level Likert scale. The following section of the questionnaire concerns the participant's experience with the task at hand. We ask whether the participants felt they had enough time to solve the task and if their background knowledge was sufficient to efficiently work on the topic. Those are important indicators that can help to explain differences in search performance. In the next part, we ask about specific functions of the search engines



(A) Participant's experience in the use of search technologies and scientific work (0: inexperienced, 5: highly experienced)



(B) Rating of research tasks that have fulfilled with the help of Google Scholar and the prototype (1: poor, 7: great)

FIGURE 5.1.: Experience of participants and task ratings

and the participant's assessment of the search results. We use the same questionnaire for both search engines, although there are a few additional questions about certain features of the prototype which will not be displayed during the review of Google Scholar. Questions that explicitly ask the participants to compare the search engines have been deliberately avoided, to have the participants focus on the search engine they are currently using instead of rating every aspect in comparison to what they have seen before.

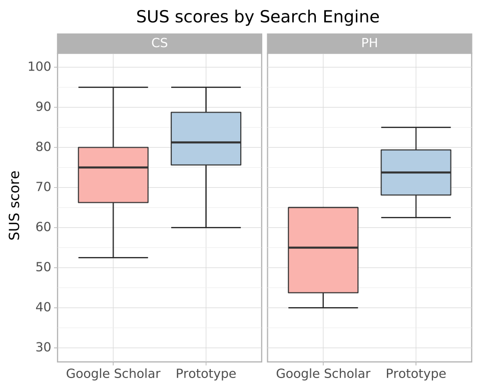
5.2. Results

The user study has been conducted with a small number of participants from a non-representative selection of the public. The qualitative nature of the study allows us to gain insights into the reasons for the choices and behaviors of the participants, and while the results are not generalizable to the public, they may give a first hint about the possible benefits of a session-based semantic search engine.

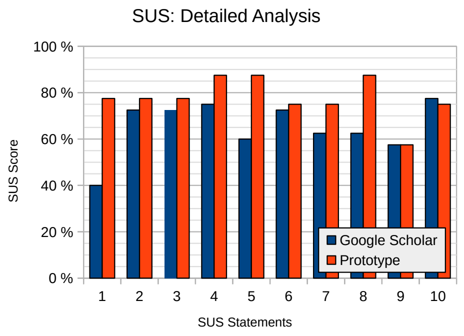
5.2.1. Participants

A total of 11 people participated in the study, in four one-hour sessions with two to four participants each. One result had to be removed due to technical issues, which brings the number of valid results to 10 (the following analysis will not include the invalid result). The participants were between 18 and 34 years old (answers were given in intervals, so the true minimum and maximum are not known), two were female and eight male. All participants were either students or had already finished their studies. Six participants had academic experience in computer science, three in mechanical engineering and one in physics. Six participants already had professional working experience in their field of study.

All participants were native German speakers with intermediate to excellent proficiency in the English language, according to a self-assessment based on the Common European Framework of Reference for Languages (CEFR) [VVT^{+}09]. The average



(A) Boxplot of SUS scores for the prototype and Google Scholar, by participant group (CS and PH)



(B) Average rating per SUS statement (the rating from 0% to 100% indicates, how much each statement contributes to the overall score)

FIGURE 5.2.: Analysis of SUS scores

language skill was between B2 (upper intermediate) and C1 (effective operational proficiency), which is considered to be sufficient for the understanding of scientific publications written in English.

The answers to the demographic questionnaire indicate that 9 out of 10 participants use search engines on a daily basis, but advanced search functions like the use of Boolean expressions or filtering results by time, region and language are primarily used by the computer scientists among the participants, with an average of three functions ever used in practice, compared to other participants who have on average used only one of these functions in the past. When it comes to the usage of academic search engines, five participants answered that they never or very infrequently use them, while only three participants said that they use them at least somewhat frequently. Concerning the advanced functions (e.g. searching for papers that quote a paper, or exporting Bibtex citations), the results are similar to regular search engines: while the computer scientists were aware of two or more of the suggested advanced functions, the other participants have used at most one of these functions so far. Participant's experience with academic search engines seems to correlate with experience in scientific work, as Figure 5.1a suggests. Three participants said they have never used Google Scholar before.

5.2.2. Task Review

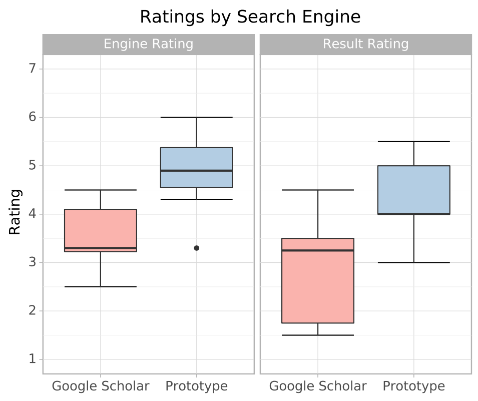
The research tasks that the participants have been assigned were met with mixed feelings. The tasks got an average rating of 4 out of 7, based on the four questions related to the task setting in the questionnaire. Half of the participants said they didn't have enough time to solve the task (which some also moaned in the comments at the end of the questionnaire), though only two participants indicated they felt somewhat overstrained by one of the tasks. Time pressure during the research phase was expected to be an issue, nevertheless it was an intentional decision for two reasons: the participants should be inclined to work quickly, as to see which search engine could help produce better results in a short amount of time, and it should allow to investigate both search engines before the participant's attention span decreases too much.

Figure 5.1b shows the rating of each participant for the tasks that they have been assigned when using the prototype and Google Scholar. While there are substantial differences in the task ratings between participants, the rating for the second task using a different search engine is consistent with the rating of the first task. This indicates that the tasks from the same field are similar in difficulty, that differences in the task ratings are primarily caused by differences in the participant's backgrounds and that the perception of a tasks difficulty is not influenced by the search engine that has been used, which is a decent indicator for the validity of the answers.

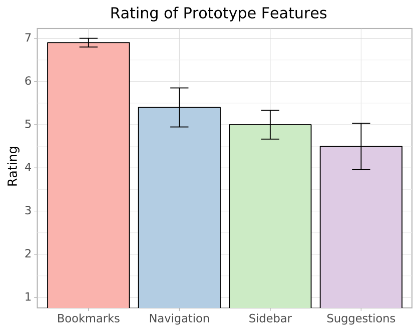
Bangor et al. [BKM09] have investigated the meaning of SUS scores by comparing the results of about 1000 studies that measured SUS for different application user interfaces, most of which were web pages and mobile applications. Their results show that the mean score for the reviewed applications was about 70, scores below 62 were in the first quartile and scores above 78 were in the forth quartile. In this study, Google Scholar got an average of 66 points by the participants, which is lies within the second quartile, while the prototype got an average rating of 78, which is at the border between the third and fourth quartile. Participants from different fields had notably different perceptions of usability, as Figure 5.2a visualizes: Users with a background in computer science generally gave more favorable usability ratings to both search engines and put the prototype just 5 points ahead of the alternative. Participants with a background in physics or mechanical engineering on the other hand had stronger opinions on the subject and put a gap of 20 points between Google Scholar and the prototype, giving the Google product a rating of only 54.

A more detailed look at the ratings gives some insight into the strengths an weaknesses of both systems: Figure 5.2b shows the average score for each of the 10 SUS statements. The raw scores have been transformed into a rating in the $[0, 1]$ range, because SUS consists of both positive and negative statements and if a system is to get the best usability scores, the participants must disagree with negative statements. The diagram shows that the scores for both the prototype and Google Scholar were surprisingly close for 7 out of 10 questions. Major differences were only observed in statements 1, 5 and 8. The first statement ("*I think that I would like to use this system frequently*") has the largest gap and indicates that the prototype was generally better liked by the participants than Google Scholar. The other statements (5: "*I found the various functions in this system were well integrated*" and 8: "*I found the system very cumbersome to use*") suggest that the participants preferred the layout of the prototype's UI and found it more intuitive to use. Both Google Scholar and the prototype scored relatively low on question 9 ("*I felt very confident using the system*"). The variance among the ratings for this statement was higher than usual and low ratings correlated with low experience in scientific work, which indicates that this assessment was probably not only a result of the system's usability.

Figure 5.3a summarizes the participant's perception of each search engine's features and their general satisfaction with the results of their research. The ratings suggest that the participants were more satisfied with the features of the prototype than



(A) Rating of participant's satisfaction with features of the search engine and the results of their research (0: not satisfied, 7: highly satisfied)



(B) Rating of the prototype's unique features: bookmarking documents, navigation in the query history, functions in the sidebar, suggested search results (0: not useful, 7: very useful)

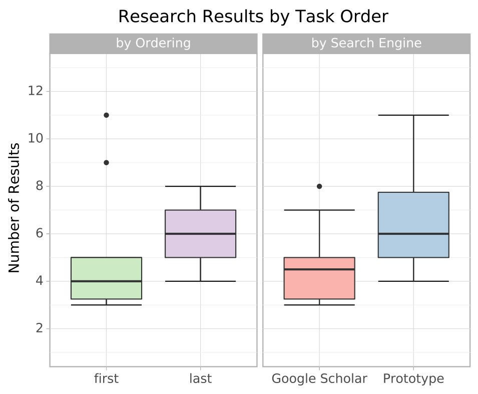
FIGURE 5.3.: Participant's satisfaction with the search engines and their research

those of Google Scholar. The search engine rating is generated from several statements about features and experiences that apply to both search engines. The largest difference concerned the statement "*the presentation of the search results was very clear*": While users of Google Scholar were neutral about this statement, the prototype got strong agreement with 6 out of 7 points. When asked whether they could assess the relevance of a document based on the preview on the search result page, users of Google Scholar slightly disagreed with the statement, but agreed (with a 5 out of 7 rating) when the prototype was used. Reason for concern gave the statement "*among the search results were mostly unsuitable documents*", which was also met with indifference among users of Google Scholar and only caused slight disagreement among users of the prototype. The gist of this is probably that literature research is still a non-trivial task that people struggle with, at least given the tools that we have today.

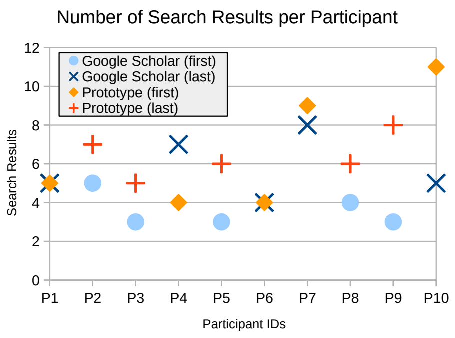
The rating of the search engine is also reflected in the participant's assessment of their own research effort: all participants were generally more satisfied with their result when they were using the prototype and said they got a better overview of the relevant literature.

Several unique features of the prototype were also part of the questionnaire, which are visualized in Figure 5.3b. The results show that the ability to navigate freely in the query history was very well perceived, while the suggested search results and generally the functions provided in the sidebar did not seem to be that important to most participants. This was also reflected in some of the comments at the end of the questionnaire, although the suggested search results and other features of the sidebar did get a favorable rating by those who actually used them. A rather small function that was not an integral part of the prototype achieved critical acclaim among the participants: with an average rating of 6.9 out of 7, the ability to save a search result by clicking the star-icon next to it appeared to be very useful to all participants of the study.

The comments at the end of the questionnaire gave valuable feedback about several functions: The half-baked spell checker has been vastly improved after complaints by



(A) Number of results per participant with a relevance rating ≥ 3. Left panel: Numbers of the first vs. the last task. Right panel: Numbers of Google Scholar vs. the prototype.



(B) Detailed view of the number of search results with a relevance rating ≥ 3 for each participant

FIGURE 5.4.: Selection of search results

participants with dyslexia. Font sizes and the arrangement of some elements have been adjusted to make all text equally well readable, especially in the sidebar. Some of the suggested features were not yet implemented, like the ability to remove specific steps from the query history or a way to preview the abstract of a saved search result, but those are open for discussion.

5.2.3. Literature Research Results

All participants saved the documents they deemed potentially useful and gave them a rating from 1 (not relevant) to 5 (highly relevant). The search results and ratings of each participant are listed in Appendix A.2.5.

It was originally intended to rate the performance of each participant based on the average rating of each document that was selected as result for a task, but due to the small number of participants per task, this approach failed. On average, there were only 5 people working on the same task and the vast majority of documents has been selected by not more than one participant. The proportion of documents that got at least two ratings lies between 13% and 26% per task. This circumstance also prevents us from judging the relevance of the suggested search results, which we intended to compare to the list of relevant documents that have been selected by the participants for each task.

While the planned methods were not applicable under the given circumstances, some insights can still be gained from the data we have. The participants have selected between 4 and 13 documents during the research phase of each task, and gave each topic a rating from 1 to 5. If we assume that each document that got an average rating of 3 or more is at least somewhat relevant, we can count how many relevant documents the participants have found in the limited time they had. Unsurprisingly, learning effects could be observed, as can be seen in Figure 5.4a: During the first pass, an average of 5.1 documents was found by the participants. During the second (last) pass, a participant found on average 6.1 documents. It is notable though, that during the first pass, there are several outliers that performed far better than during the second pass (as indicated by the dots in the leftmost panel). These can be explained if we take a look at the number of relevant search results grouped by the search engine that has been used: When using Google Scholar, the participants found an average of 4.7 documents; the same average is at 6.5, when the prototype was used, and the variance is lower compared to the first/last ranking we reviewed before. The detailed analysis in Figure 5.4b backs this observation: There is only a single case where Google Scholar is ahead of the prototype, and that is in an instance where the prototype was the first search engine that was used. The previously observed outliers are from participants that used the prototype during their first pass and consequently scored lower during the second pass when Google Scholar was used.

5.3. Discussion

The evaluation of the study results is rather intricate under the given conditions: On the one hand, the prototype got a consistently positive rating by the participants of the study and was ahead of the competing search engine under many aspects. On the other hand, the study was too small to make reliable statements and there were methodical issues caused by the low number of participants, which would – strictly speaking – invalidate all conclusions that we might draw from the data. Still, we will attempt to shed some light on some of the research questions that we were trying to answer.

First of all, it was clearly shown that the participants were able to work with the prototype and to use it to find relevant results for the tasks they've been assigned. All participants were able to solve their task using the prototype and there were no technical issues related to the prototype during the study. This might seem like an obvious prerequisite for this research, but given that the prototype was developed in parallel to the study and that there were no major pretests, it was not clear from the beginning whether the software would actually work as intended.

At least for the non-representative group of participants, the usability of the prototype is superior to that of Google Scholar. This is indicated by the results of the System Usability Scale (SUS) and the agreement of participants to statements about properties of the search engine they have used, as well as their satisfaction with the results of their research. Though most participants seemed to prefer the prototype, there were major differences in the usability rating in this relatively small group. How the two search engines would fare in a representative sample is open for discussion in future work.

There are patterns in the data which suggest that users of the prototype work more efficiently and are therefore able to find more relevant results than users of Google Scholar. This is indicated by the average number of relevant results that the participants of the study have found during the research. While there seems to be a learning effect that causes the results of the second literature research generally to be somewhat better than the result of the first pass, the data also shows that users of the prototype usually find more relevant documents and that this effect outweighs the learning effect by a factor of two. Given that the choice which search engine gets used first is randomized, this is a strong indicator that at least the participants of the study were able to work more efficiently with the prototype than with Google Scholar.

The analysis of the participant's research results showed that the overlap between what the participants considered relevant for each task was rather low: More than 80% of the submitted documents were selected by just a single user. While this circumstance made it more difficult to assess the relevance of search results, it also shows how diverse the search behavior of different persons can be. In a collaborative search scenario, this kind of diversity could become a valuable asset, which would allow small groups to efficiently research a topic in parallel. Under such circumstances, the most important documents can be identified by majority vote, yet at the same time a good coverage of all at least somewhat relevant documents can also be achieved.

Assessing which function or combination of functions is responsible for the prototype's success among the participants of the study is an important research question, which unfortunately cannot be answered based on the available data. It is possible that the increased search performance is entirely based on differences in the UI: This hypothesis is supported by the high usability rating and the rating of unique UI-related features of the prototype, like the bookmarking system or the document preview on the search result page. On the other hand, there is overlap between the search results of Google Scholar and the prototype, so both search engines seem to share some ideas about the relevance of documents, although the approaches differ considerably: While Google can rely on usage statistics, the popularity of authors and the number of quotations a paper received, the prototype can only work with the facts that are found in the data and must build all additional data structures (topic model and search index) around these facts. Given the fundamental differences between the approaches of the search engines, a positive reception by the participants of the study seems unlikely if the quality of the search results was much worse than that of Google Scholar. Figuring out how much influence the session-based approach has on search performance, how much the topic centroid contributes to the relevance of search results and how useful the search suggestions are that are primarily generated from the topic centroid must however be deferred to future work.

Chapter 6

Conclusion

We have described session-based semantic search, a concept for information retrieval that improves the selection and ranking of search results by interpreting queries in the context of a user-specific search session. To put this concept to the test, we have built a prototype of a session-based semantic search engine, which we published as free and open source software together with this thesis. In a qualitative, non-representative user study we have shown that the prototype is suitable for exploratory search tasks.

Review

We have defined an information retrieval process that is built around the concept of a search session, which serves as a container for all context information of a single user that could be relevant for the ranking of search results. We relied on previous research in the field of session-based search to specify a full-text search algorithm that incorporates terms from the user's entire query history.

To get a better understanding of the user's search intent we introduced a novel method to track the most relevant themes in a search session: the *topic centroid*. With the help of statistical topic models, we are able to identify the most important topics in a ranked list of search results. Using a method we call *topic shift*, we can iteratively adjust the topic centroid with each submitted query, which allows us to recommend new topics while still interpreting queries in the context of the entire search session. These capabilities are reflected in a search strategy that combines full-text search in the context of the entire search session with a topic-based retrieval method that depends on the state of the topic centroid.

Furthermore, we discussed a concept that extends the presented methods to the collaborative setting. We introduced the model of a shared topic centroid which is influenced by the search behavior of all members of a group and in turn affects the ranking of each user's search results.

For the prototypical implementation of the search concept, we have built a search index that contains the full text of all 1.2 million documents from the arXiv repository as of March 2017. This document collection provided a sufficient basis for exploratory search scenarios and its origin from an open access repository should allow other researchers to rebuild the index with relatively low effort. We published the necessary tools to retrieve the data, prepare the documents, generate a hierarchical topic model and to finally build the search index.

Our prototype implements the core concepts of a session-based semantic search engine, including search sessions, the topic centroid and a combined full-text and topic search algorithm. The user interface of the search engine provides the familiar experience of a conventional literature search engine, but it also offers a variety of advanced functions that make use of contextual information. These include the visualization of important topics in the topic graph, the suggested search results that are generated from the state of the topic centroid and the breadcrumb trail which can be used for navigation in the query history. Building a practicable solution that can also prevail outside of the academic context was a priority during development of the prototype, which is reflected in the decent results of the performance review.

To investigate how well the prototype fares under realistic circumstances, we have conducted a qualitative user study in which we compare the prototype with a conventional academic search engine. Due to the non-representative selection of participants, the results of the study must be treated with caution. However, we were able to make a number of promising observations: All participants were able to work with the prototype and to solve their assigned search tasks after a very brief introduction. Also, the participants gave higher usability ratings and were more satisfied with the results of their assigned tasks when they were using the prototype.

Future Work

By publishing the full source code of the search engine as well as all records regarding the user study (see Appendix) we hope to encourage other researchers to contribute to this work and to the the field of research it comprises. Throughout this thesis we have discussed several prospects for future work:

The effectiveness of the concept of session-based semantic search in general and the prototype in particular could be substantiated in a representative user study that evaluates the performance of participants working with different information retrieval systems. Such an endeavor could greatly benefit from the groundwork that we have laid: The search engine and all the necessary tools to build a suitable search index have been published and we have recorded our experiences during a small-scale user study that can serve as a starting point for a larger follow-up study.

We have proposed a concept for collaborative search that has not been investigated so far. It would be an interesting challenge to create a comprehensive concept for a collaborative search engine where group members share a common search context, to extend the prototype with an implementation of this concept and finally to evaluate the benefits of such an approach in a user study.

An extension of other aspects of the prototype could also be of interest: The accuracy of identified topics and therefore the topic centroid could be improved by evaluating the user's interaction with the documents on the search result page, as was discussed in Section 3.2.3. Furthermore, only part of the concept for topic graph visualization has been implemented so far and the current visualization of topics is often hard to understand; the users would surely benefit from a more intuitive presentation of the core topics in a search session.

Prospects

We have observed that contemporary web search engines have aggressively optimized their information retrieval methods with notable success, but the retrieval model has largely remained unchanged. With the concepts that were presented in this work and the prototype that was built, we have shown that it is possible to support the user in complex exploratory search tasks that require several query reformulations to be solved. It is our hope that the institutions which promote the dissemination of knowledge, like public libraries and open access journals, dare to rethink the way we interact with search engines today and keep an open mind towards novel search concepts that have the potential to make the process of discovering knowledge as exciting as it deserves to be.

Bibliography

[ABD06] Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 19–26, Seattle, Washington, USA. ACM, 2006.

[All02] James Allan. *Introduction to topic detection and tracking.* In *Topic Detection and Tracking: Event-based Information Organization.* James Allan, editor. Springer US, Boston, MA, 2002, pages 1–16.

[AM08] Saleema Amershi and Meredith Ringel Morris. Cosearch: a system for co-located collaborative web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1647–1656, Florence, Italy. ACM, 2008.

[BBC$^+$08] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, Aristides Gionis, and Sebastiano Vigna. The query-flow graph: model and applications. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 609–618, Napa Valley, California, USA. ACM, 2008.

[BFG$^+$07] Andrei Z. Broder, Marcus Fontoura, Evgeniy Gabrilovich, Amruta Joshi, Vanja Josifovski, and Tong Zhang. Robust classification of rare queries using web knowledge. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 231–238, Amsterdam, The Netherlands. ACM, 2007.

[BGJ10] David M. Blei, Thomas L. Griffiths, and Michael I. Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *J. ACM*, 57(2):7:1–7:30, 2010-02.

[BHX14] Amparo Elizabeth Cano Basave, Yulan He, and Ruifeng Xu. Automatic labelling of topic models learned from twitter by summarisation. In *The 52nd Annual Meeting of the Association for Computational Linguistics: Proceedings of the Conference: Volume 2: Short Papers*, pages 618–624. Association for Computational Linguistics (ACL), 2014-06.

[BI97] Pia Borlund and Peter Ingwersen. The development of a method for the evaluation of interactive information retrieval systems. *Journal of Documentation*, 53(3):225–250, 1997.

[BKM09] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: adding an adjective rating scale. *J. Usability Studies*, 4(3):114–123, 2009-05.

[BKM10] Aaron Bangor, Philip T. Kortum, and James T. Miller. An empirical evaluation of the system usability scale. *Int. J. Hum. Comput. Interaction*, 24(6):574–594, 2010-03-09.

[Ble12] David M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, 2012-#apr#.

[BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003-03.

[BR99] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Bro$^+$96] John Brooke et al. Sus - a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[BSD10] Paul N. Bennett, Krysta Svore, and Susan T. Dumais. Classification-enhanced ranking. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 111–120, Raleigh, North Carolina, USA. ACM, 2010.

[CGK$^+$12] Robert Capra, Gene Golovchinsky, Bill Kules, Dan Russell, Catherine L. Smith, Daniel Tunkelang, and Ryen W. White. Hcir 2011: the fifth international workshop on human-computer interaction and information retrieval. *SIGIR Forum*, 45(2):102–107, 2012-#jan#.

[CHS$^+$09] Huanhuan Cao, Derek Hao Hu, Dou Shen, Daxin Jiang, Jian-Tao Sun, Enhong Chen, and Qiang Yang. Context-aware query classification. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 3–10, Boston, MA, USA. ACM, 2009.

[CKH$^+$14] Ben Carterette, Evangelos Kanoulas, Mark M. Hall, and Paul D. Clough. Overview of the TREC 2014 session track. In *Proceedings of The Twenty-Third Text REtrieval Conference, TREC 2014, Gaithersburg, Maryland, USA, November 19-21, 2014*, 2014.

[CMS10] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice.* Addison-Wesley Publishing Company, 2010.

[Dam64] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, 1964-#mar#.

[DCC01] Susan Dumais, Edward Cutrell, and Hao Chen. Optimizing search by showing results in context. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '01, pages 277–284, Seattle, Washington, USA. ACM, 2001.

[DCD$^+$06] Pierre Dupont, Jérôme Callut, Grégoire Dooms, Jean-Noël Monette, Yves Deville, and BP Sainte. Relevant subgraph extraction from random walks in a graph. *Universite catholique de Louvain, UCL/INGI, Number RR*, 7, 2006.

[DDF$^+$90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[DTB$^+$09] Mariam Daoud, Lynda Tamine-Lechani, Mohand Boughanem, and Bilal Chebaro. A session based personalized search using an ontological user profile. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1732–1736, Honolulu, Hawaii. ACM, 2009.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, 2000. AAI9980887.

[FKM$^+$05] Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan Dumais, and Thomas White. Evaluating implicit measures to improve web search. *ACM Trans. Inf. Syst.*, 23(2):147–168, 2005-04.

[Fos06] Jonathan Foster. Collaborative information seeking and retrieval. *Annual Rev. Info. Sci & Technol.*, 40(1):329–356, 2006-#dec#.

[GCP03] Susan Gauch, Jason Chaffee, and Alexander Pretschner. Ontology-based personalized search and browsing. *Web Intelligence and Agent Systems: An international Journal*, 1(3, 4):219–234, 2003.

[GD11] Gene Golovchinsky and Abdigani Diriye. Session-based search with querium. In *Proceedings of the HCIR 2011 Workshop, Mountain View, CA, USA*, 2011.

[GK03] Mark Girolami and Ata Kabán. On an equivalence between plsi and lda. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 433–434, Toronto, Canada. ACM, 2003.

[Gla01] Natalie S. Glance. Community search assistant. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, IUI '01, pages 91–96, Santa Fe, New Mexico, USA. ACM, 2001.

[GOB$^+$12] Brynjar Gretarsson, John O'Donovan, Svetlin Bostandjiev, Tobias Höllerer, Arthur Asuncion, David Newman, and Padhraic Smyth. Topicnets: visual analysis of large text corpora with topic modeling. *ACM Trans. Intell. Syst. Technol.*, 3(2):23:1–23:26, 2012-#feb#.

[GT15] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine.* O'Reilly Media, 2015.

[GY14a] Dongyi Guan and Hui Yang. Is the first query the most important: an evaluation of query aggregation schemes in session search. In *Asia Information Retrieval Symposium*, pages 86–99. Springer, 2014.

[GY14b] Dongyi Guan and Hui Yang. Query aggregation in session search. In *Proceedings of the 3rd Workshop on Data-Driven User Behavioral Modeling and Mining from Social Media*, DUBMOD '14, pages 19–22, Shanghai, China. ACM, 2014.

[GZY13] Dongyi Guan, Sicong Zhang, and Hui Yang. Utilizing query change for session search. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 453–462. ACM, 2013.

[Har54] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

[HBB10] Matthew D. Hoffman, David M. Blei, and Francis Bach. Online learning for latent dirichlet allocation. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems*, NIPS'10, pages 856–864, Vancouver, British Columbia, Canada. Curran Associates Inc., 2010.

[HHK+13] Ioana Hulpus, Conor Hayes, Marcel Karnstedt, and Derek Greene. Unsupervised graph-based topic labelling using dbpedia. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 465–474, Rome, Italy. ACM, 2013.

[HJ05] Preben Hansen and Kalervo Järvelin. Collaborative information retrieval in an information-intensive domain. *Information Processing & Management*, 41(5):1101–1119, 2005.

[HL09] Nadine Höchstötter and Dirk Lewandowski. What users see - structures in search engine results pages. *Information Sciences*, 179(12):1796–1812, 2009.

[Hof99] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 50–57, Berkeley, California, USA. ACM, 1999.

[JHA14] Jiepu Jiang, Daqing He, and James Allan. Searching, browsing, and clicking in a search session: changes in user behavior by task and over time. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 607–616. ACM, 2014.

[JHS+15] Jiepu Jiang, Ahmed Hassan Awadallah, Xiaolin Shi, and Ryen W White. Understanding and predicting graded search satisfaction. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 57–66. ACM, 2015.

[Joa02] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[JSB04] Hideo Joho, Mark Sanderson, and Micheline Beaulieu. A study of user interaction with a concept-based interactive query expansion support tool. In *European Conference on Information Retrieval*, pages 42–56. Springer, 2004.

[KCC+10] Evangelos Kanoulas, Paul Clough, Ben Carterette, and Mark Sanderson. Session track at trec 2010. *Simulation of Interaction*:13, 2010.

[LDY14] Jiyun Luo, Xuchu Dong, and Hui Yang. Modeling Rich Interactions in Session Search - Georgetown University at TREC 2014 Session Track. Technical report, Department of Computer Science, Georgetown University, 2014.

[LGN+11] Jey Han Lau, Karl Grieser, David Newman, and Timothy Baldwin. Automatic labelling of topic models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1536–1545. Association for Computational Linguistics, 2011.

[LN02] Yann Laurillau and Laurence Nigay. Clover architecture for groupware. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 236–245. ACM, 2002.

[LNK$^+$10] Jey Han Lau, David Newman, Sarvnaz Karimi, and Timothy Baldwin. Best topic word selection for topic labelling. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 605–613, Beijing, China. Association for Computational Linguistics, 2010.

[LZD$^+$15] Jiyun Luo, Sicong Zhang, Xuchu Dong, and Hui Yang. Designing states, actions, and rewards for using pomdp in session search. In *European Conference on Information Retrieval*, pages 526–537. Springer, 2015.

[LZY14] Jiyun Luo, Sicong Zhang, and Hui Yang. Win-win search: dual-agent stochastic game in session search. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 587–596. ACM, 2014.

[MH07] Meredith Ringel Morris and Eric Horvitz. Searchtogether: an interface for collaborative web search. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 3–12. ACM, 2007.

[Mor08] Meredith Ringel Morris. A survey of collaborative web search practices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1657–1660. ACM, 2008.

[Mor13] Meredith Ringel Morris. Collaborative search revisited. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 1181–1192. ACM, 2013.

[MS16] M. Mitsui and C. Shah. Coagmento 2.0: a system for capturing individual and group information seeking behavior. In *2016 IEEE/ACM Joint Conference on Digital Libraries (JCDL)*, pages 233–234, 2016-06.

[MSZ07] Qiaozhu Mei, Xuehua Shen, and ChengXiang Zhai. Automatic labeling of multinomial topic models. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 490–499, San Jose, California, USA. ACM, 2007.

[NNI$^+$97] Yoshiki Niwa, Shingo Nishioka, Makoto Iwayama, Akihiko Takano, and Yoshihiko Nitta. Topic graph generation for query navigation: use of frequency classes for topic extraction. *arXiv preprint cmp-lg/9712005*, 1997.

[PAB$^+$04] Tim Paek, Maneesh Agrawala, Sumit Basu, Steve Drucker, Trausti Kristjansson, Ron Logan, Kentaro Toyama, and Andy Wilson. Toward universal mobile interaction for shared displays. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 266–269. ACM, 2004.

[PGS$^+$08] Jeremy Pickens, Gene Golovchinsky, Chirag Shah, Pernilla Qvarfordt, and Maribeth Back. Algorithmic mediation for collaborative exploratory search. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 315–322. ACM, 2008.

[Por80] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[PSC$^+$02] James Pitkow, Hinrich Schütze, Todd Cass, Rob Cooley, Don Turnbull, Andy Edmonds, Eytan Adar, and Thomas Breuel. Personalized search. *Commun. ACM*, 45(9):50–55, 2002-09.

[PWB$^+$12] John Paisley, Chong Wang, David M. Blei, and Michael I. Jordan. Nested hierarchical dirichlet processes, 2012.

[Řeh11] Radim Řehůřek. *Subspace tracking for latent semantic analysis.* In *Advances in Information Retrieval: 33rd European Conference on IR Research, ECIR 2011, Dublin, Ireland, April 18-21, 2011. Proceedings.* Paul Clough, Colum Foley, Cathal Gurrin, Gareth J. F. Jones, Wessel Kraaij, Hyowon Lee, and Vanessa Mudoch, editors. Springer Berlin Heidelberg, 2011, pages 289–300.

[Rod91] Tom Rodden. A survey of cscw systems. *Interacting with Computers*, 3(3):319–353, 1991.

[SBB$^+$03] Barry Smyth, Evelyn Balfe, Peter Briggs, Maurice Coyle, and Jill Freyne. Collaborative web search. In *IJCAI*, pages 1417–1419, 2003.

[SBB$^+$05] Barry Smyth, Evelyn Balfe, Oisin Boydell, Keith Bradley, Peter Briggs, Maurice Coyle, and Jill Freyne. A live-user evaluation of collaborative web search. In *IJCAI*, pages 1419–1424, 2005.

[SFC$^+$04] Barry Smyth, Jill Freyne, Maurice Coyle, Peter Briggs, and Evelyn Balfe. I-spy - anonymous, community-based personalization by collaborative meta-search. In *Research and Development in Intelligent Systems XX*, pages 367–380. Springer, 2004.

[Sha10a] Chirag Shah. Coagmento - a collaborative information seeking, synthesis, and sense-making framework. *Integrated demo at CSCW*:6–11, 2010.

[Sha10b] Chirag Shah. Collaborative information seeking: a literature review. In *Advances in librarianship*, pages 3–33. Emerald Group Publishing Limited, 2010.

[Spa72] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[SSY$^+$06] Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Building bridges for web query classification. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 131–138. ACM, 2006.

[SSZ04] Smitha Sriram, Xuehua Shen, and Chengxiang Zhai. A session-based search engine. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 492–493, Sheffield, United Kingdom. ACM, 2004.

[STZ05] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Context-sensitive information retrieval using implicit feedback. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 43–50. ACM, 2005.

[STZ07] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Privacy protection in personalized search. In *ACM SIGIR Forum*, volume 41 of number 1, pages 4–17. ACM, 2007.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975-11.

[SZ03] Xuehua Shen and Cheng Xiang Zhai. Exploiting query history for document ranking in interactive information retrieval. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 377–378. ACM, 2003.

[TDH05] Jaime Teevan, Susan T Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 449–456. ACM, 2005.

[TJB$^+$05] Yee W Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Sharing clusters among related groups: hierarchical dirichlet processes. In *Advances in neural information processing systems*, pages 1385–1392, 2005.

[TN96] Michael Twidale and David Nichols. Collaborative browsing and visualisation of the search process. In *Aslib Proceedings*, volume 48 of number 7-8, pages 177–182, 1996.

[TSZ06] Bin Tan, Xuehua Shen, and ChengXiang Zhai. Mining long-term search history to improve search accuracy. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 718–723. ACM, 2006.

[VH00] Ellen Voorhees and Donna Harman. Overview of the eighth text retrieval conference (trec-8). *Proceedings of the Eighth Text REtrieval Conference (TREC-8)*, 2000-11.

[VVT$^+$09] Norman Verhelst, Piet Van Avermaet, Sauli Takala, Neus Figueras, and Brian North. *Common European Framework of Reference for Languages: learning, teaching, assessment.* Cambridge University Press, 2009.

[WBD10] Ryen W White, Paul N Bennett, and Susan T Dumais. Predicting short-term interests using activity-based search context. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1009–1018. ACM, 2010.

[WR09] Ryen W White and Resa A Roth. Exploratory search: beyond the query-response paradigm. *Synthesis lectures on information concepts, retrieval, and services*, 1(1):1–98, 2009.

[YS16] Grace Hui Yang and Ian Soboroff. TREC 2016 dynamic domain track overview. In *Proceedings of the Text REtrieval Conference 2016 (TREC 2016)*, 2016.

[ZLY15] Andrew Jie Zhou, Jiyun Luo, and Hui Yang. Dumpling: a novel dynamic search engine. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 1049–1050, Santiago, Chile. ACM, 2015.

Weblinks

[@Aks] Andrew Aksyonoff. Sphinx. URL: http://sphinxsearch.com/ (visited on 2017-09-11).

[@Ban] Shay Banon. Elasticsearch. URL: https://www.elastic.co/products/elasticsearch (visited on 2017-08-11).

[@Fou] Apache Software Foundation. Apache solr. URL: https://lucene.apache.org/solr/ (visited on 2017-09-11).

[@Gin] Paul Ginsparg. Arxiv. URL: https://arxiv.org/ (visited on 2017-08-17).

[@Goo] Google. Google scholar. URL: https://scholar.google.com/ (visited on 2017-08-13).

[@Hon] Matthew Honnibal. Spacy. URL: https://spacy.io/ (visited on 2017-08-20).

[@LVN$^+$] Carl Lagoze, Herbert Van de Sompel, Michael Nelson, and Simeon Warner. The open archives initiative protocol for metadata harvesting. URL: http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm (visited on 2017-08-18).

[@Mic] Microsoft. Bing. URL: https://www.bing.com/ (visited on 2017-09-02).

[@Ném] László Németh. Hunspell. URL: https://hunspell.github.io/ (visited on 2017-08-08).

[@Pai] John Paisley. Implementation of nhdp. URL: http://www.columbia.edu/~jwp2128/software.html (visited on 2017-08-19).

[@Řeh] Radim Řehůřek. Gensim. URL: https://radimrehurek.com/gensim/ (visited on 2017-08-18).

[@Ron] Armin Ronacher. Flask. URL: http://flask.pocoo.org/ (visited on 2017-08-18).

[@Ros] Guido van Rossum. Python. URL: https://www.python.org/ (visited on 2017-08-11).

[@Shi] Yusuke Shinyama. Pdfminer. URL: https://euske.github.io/pdfminer/ (visited on 2017-08-18).

[@ZLY] Andrew Jie Zhou, Jiyun Luo, and Hui Yang. Dumpling. URL: https://web.archive.org/web/*/dumplingproject.org (visited on 2017-09-03).

Appendix A

Appendix

A.1. Prototype: Source Code

We publish the prototype of the session-based semantic search engine that was developed as part of this work under a permissive free software license, in the hope that it may be useful to other researchers and developers. Please fork the project if you're interested in designing your own search engine; contributions to the code base are greatly appreciated.

The prototype of the search engine consists of a web application written in Python that uses the micro web framework Flask to process queries and serve search results. At the time of writing, *epic search* is the working title of the project, though this may change in the future.

The source code and accompanying documentation is located at:

`https://github.com/Klamann/epic-search`

Furthermore, we provide some tools that should help you to get started with the search index. The index builder contains a set of scripts to get metadata from arxiv.org, parse PDFs, create a topic model and to build the Elasticsearch index that can be used by the search engine.

The source code and accompanying documentation is located at:

`https://github.com/Klamann/search-index-builder`

If you are interested in reviewing the state of these projects at the time this thesis was submitted, you may checkout the projects and refer to the revision that has been tagged with *thesis-submission*, or just follow these links:

`https://github.com/Klamann/epic-search/releases/tag/thesis-submission`
`https://github.com/Klamann/search-index-builder/releases/tag/thesis-submission`

A.2. Survey

This appendix contains all materials that were used to execute the survey, as well as the raw results of the survey in anonymized form.

A.2.1. Tasks

During the study, there were two groups of participants: computer scientists and physicists (including mechanical engineers). For each group, two tasks were available (one for the prototype and one for the reference search engine), for a combined total of four tasks.

Crowd Work (CS, A)

This is task A for participants with a background in computer science.

> Michael is as researcher who is currently occupied with the analysis of documents for large archives. Often, the documents he's working with are not available in machine-readable form, which is why he spends a large portion of his working hours with the automated processing of documents, followed by wearisome manual corrections of processing errors.
>
> A while ago he heard about *microwork* or *crowd work*, a sub-area of *crowdsourcing*, where people solve small tasks and get paid directly for each task they solve. As a Wikipedia author, Michael has had good experiences with crowdsourcing and could very well imagine to outsource the lengthy copying and correcting of his documents to "the crowd". He has already asked for a budget to implement the concept, but the committee responsible is skeptical: it wants to see proof that the idea can actually work.
>
> Desperately, Michael turns to you: Because of an important project he is currently working on, he has no time to research the topic, but the budget for the coming quarter will be determined in just a few days. Can you help him and compile the necessary literature to convince the committee of his idea?
>
> These topics are of special interest:
>
> - For which types of work is crowd work suitable, for which ones is it not?
> - How is the quality of the work of crowd workers to be assessed, especially in tasks such as the digitization of documents?
> - How is the handling of non-public or confidential data to be assessed?
> - Are there any legal concerns? (e.g. compliance with minimum wages, pause times and other regulations)
> - Are there any ethical implications? (e.g. working conditions, social situation of workers)

Distributed Graph Processing (CS, B)

This is task B for participants with a background in computer science.

Fabius is a software developer in a medium-sized enterprise, but recently he has brought attention to himself with a project that he has pursued in his spare time: a web search engine that differentiates itself from the competition. At a tech conference, the prototype was so well received that an investor made Fabius an offer: if the search engine works just as well with ten times the amount of data, he will finance the project for a full year.

Fabius doesn't want to miss this opportunity, but he quickly realizes that parts of the search algorithm don't scale with larger amounts of data. Especially the link graph, which is the core of his search engine, is very slow since it doesn't fit into RAM anymore. He realizes that he has to distribute the graph on several machines, even if this requires changes to the algorithms.

A couple of days later Fabius contacts you: He has come up with a concept to efficiently calculate the necessary graph algorithms, especially PageRank, on several machines. However, since he is not sure if he has missed an option, he asks you to do some research for him. What are the currently best approaches to calculate graph algorithms on distributed systems?

These topics are of special interest:

- Fabius only wants the best of the best: The system must be highly scalable, therefore he is ready to test the latest methods and technologies.
- General information on distributed graphs is relevant, but the crucial factor is the performance of the PageRank algorithm.
- Besides specific algorithms and frameworks, basic programming concepts and best practices are also relevant.

Hubble Constant (PH, A)

This is task A for participants with a background in physics or mechanical engineering.

Marie is studying physics and has discovered her special interest in astrophysics during her studies. Recently, she spoke at a conference with an ESA employee who recommended that she should apply for an internship at the European Space Operations Centre (ESOC) in Darmstadt. Upon request, she was offered a position in a research group dealing with the measurement of the Hubble constant.

The Hubble constant H_0 describes the current rate of expansion of the universe and is about 65 to 75 $\frac{km}{s \cdot Mpc}$ (Mpc = Megaparsec). The Hubble constant can be inferred by measuring distances to other objects in the universe at different points in time. Common methods include the measurement of the distance to standard candles such as cepheids or supernovae of type Ia. A relatively new method uses gravitational lenses to calculate the distance to objects behind them. For this purpose, ESA has access to the Hubble Space Telescope.

Marie does not want to miss this opportunity and is therefore making extensive preparations for the interview. To make sure she doesn't miss any important information, she asks you in addition to her own research to look for recent papers covering the current state of research on determining the Hubble constant.

These topics are of special interest:

- An overview of the current state of the research regarding the Hubble constant
- Currently common measuring methods, as well as their advantages and disadvantages
- Reasons for the differences between the measured values when using various methods
- The latest measurements and the currently most promising methods
- Note: when it comes to concrete measurements, papers that are more than 10 years old are probably already obsolete from today's point of view.

Graphene Supercapacitors (PH, B)

This is task B for participants with a background in physics or mechanical engineering.

Simon works as an engineer at a company that manufactures energy storage systems for motor homes and energy-autonomous buildings. For this purpose, photovoltaic systems and wind generators are primarily deployed in combination with lithium-ion batteries. The use of rechargeable batteries is primarily due to the high energy density and relatively low manufacturing costs, but also comes with disadvantages: e. g. due to the low power density, devices with high peak energy requirements cannot be operated directly.

The company is now considering to additionally deploy supercapacitors to compensate for the disadvantages of the accumulators. Simon was entrusted with the task of testing the use of different capacitor types and developing a combined solution of battery and capacitor. Just before his first interim report, his manager tells him that he recently read something about graphene based capacitors that are far superior to the currently available components.

Graphene is the name for a modification of carbon with a two-dimensional structure, i.e. it consists of a single atomic layer of carbon. This provides it with some special properties such as very good electrical conductivity and a high surface-to-volume ratio, which makes it interesting for the use in capacitors. Simon is now supposed to investigate the feasibility of graphene based supercapacitors on the basis of latest research results, but so far he has not been very successful. Can you help him find the literature he needs?

These topics are of special interest:

- An overview of the current state of the research regarding graphene super capacitors
- Physical properties compared to conventional super capacitors

> - Production techniques and electrode materials that are used
> - An assessment of the product maturity of graphene supercapacitors

Note

Every task had the following note at the end of the document:

> A few general hints:
>
> - It is important to get an overview of the literature on the topic - a glance at the abstract is often sufficient for an initial assessment of a paper
> - You do not need to answer the questions. It is all about finding the best possible sources in which will likely contain the answers to the questions.
> - Most papers will only answer one aspect of the question. Nevertheless, all papers that can make a contribution to answering the questions are relevant.
> - In the research phase, about 5 to 20 documents should be selected; a superficial check of relevance is sufficient. In the subsequent evaluation phase, the papers can then be examined in more detail.

A.2.2. Document Filter for Google Scholar

This is a user script that works with browser extensions like Tampermonkey for Chromium/Chrome and Greasemonkey for Firefox. It modifies all queries submitted to Google Scholar to only display results from arxiv.org. This is an important prerequisite to make the search results of Google Scholar comparable to the search results of the prototype, because only then both search engines work with the same document collection.

```
// ==UserScript==
// @name         Google Scholar: arxiv.org filter
// @namespace    google-scholar-arxiv-filter
// @author       Sebastian Straub
// @description  use "site:arxiv.org" for all google scholar queries
// @date         2017-07-08
// @version      0.0.1
// @include      http://scholar.google.de/*
// @include      https://scholar.google.de/*
// @include      http://scholar.google.com/*
// @include      https://scholar.google.com/*
// @grant        none
// ==/UserScript==

(function () {
  // find the relevant form fields
  var searchForm = document.f;
  var inputQuery = document.getElementById('gs_hdr_frm_in_txt');

  // add a note that the arxiv.org filter has been initialized
  var note = document.createElement('span');
  note.setAttribute('style', 'color: #d44;');
  note.innerHTML = 'arxiv.org document filter enabled';
  searchForm.appendChild(note);

  // add the event listener
  searchForm.onsubmit = function () {
    var query = inputQuery.value;
    if (query.indexOf('site:arxiv.org') < 0) {
      query = query + ' site:arxiv.org';
    }
    inputQuery.value = query;
  };

  // check that "site:arxiv.org" was in the query
  var filterIndex = inputQuery.value.indexOf('site:arxiv.org');
  if (filterIndex >= 0) {
    // great! remove the filter to make the displayed result more pretty
    var q = inputQuery.value;
    var userQuery = (q.slice(0, filterIndex) + q.slice(filterIndex + 14)).trim
        ();
    setTimeout(function(){
      inputQuery.value = userQuery;
    }, 100);
  } else {
    // looks like someone is trying to trick us. Submit the query again
    inputQuery.value = inputQuery.value + ' site:arxiv.org';
    document.f.submit();
  }
})();
```

A.2.3. Questionnaires

Two questionnaires have been designed for the study: the demographic questionnaire, which asks for some information about the participant's background, and the task review questionnaire, which is filled after each research task.

Demographic

This is a reproduction of the demographic questionnaire. The titles between the questions have been inserted for convenience and were not part of the original form. Processing instructions are placed within square brackets.

1. What is your gender?
   a) female
   b) male
   c) other
   d) I prefer not to answer
2. What is your age?
   a) 17 or younger
   b) 18 to 24
   c) 25 to 34
   d) 35 to 44
   e) 45 to 54
   f) 55 to 64
   g) 65 or older
   h) I prefer not to answer
3. How would you rate your English skills? (according to the Common European Framework of Reference for Languages, CEFR)
   a) A1: Beginner (Can understand and use familiar everyday expressions)
   b) A2: Elementary (Can understand sentences and frequently used expressions related to areas of most immediate relevance)
   c) B1: Intermediate (Can understand the main points of clear standard input on familiar matters)

   d) B2: Upper Intermediate (Can understand the main ideas of complex text on both concrete and abstract topics)
   e) C1: Advanced (Can understand a wide range of demanding, longer clauses)
   f) C2: Mastery (Can understand with ease virtually everything heard or read.)

Education and field of study

4. What is the highest level of education you have completed so far?
   a) High school diploma
   b) Bachelor's degree
   c) Master's degree
   d) Doctor
   e) I prefer not to answer
5. Are you a student or did you study at a university?
   a) yes
   b) no
6. Please specify your field of study (multiple answers possible) *[only if 5 a) selected, multiple choice]*
   a) Physics
   b) Mathematics
   c) Computer Science
   d) Mechanical engineering
   e) Electrical engineering
   f) Other: __________
7. How many semesters did you study? *[only if 5 a) selected]*
   a) 1-2
   b) 3-5
   c) 6-8
   d) 9-12
   e) 13+
   f) I prefer not to answer

Working experience

8. Please select all fields from the list, in which you have working experience *[multiple choice]*
   a) Physics
   b) Mathematics
   c) Computer Science
   d) Mechanical engineering
   e) Electrical engineering
   f) Other engineering
   g) No working experience in one of these fields
9. How many years of experience do you have in the selected fields? (please use combined experience in case of multiple selection) *[only if one or more of 8 a-f selected]*
   a) 0-1
   b) 2-3
   c) 4-5
   d) 6-10
   e) 11+

Usage of search engines

10. How often do you typically use search engines?
    a) several times a day
    b) daily
    c) several times a week
    d) once a week
    e) less frequently
11. Some search engines provide keywords or filters to help you refine the search results. Which of these filters have you used before? *[multiple choice]*
    a) by language or region
    b) by file type
    c) by release date

   d) by domain name
   e) boolean expressions
   f) none of the above

Experience with scientific work

12. How would you rate your experience with scientific work? (e.g. researching and writing papers)
    a) 7 point Likert scale: Beginner ... Expert
13. Which of these activities related to scientific work have you already done? *[multiple choice]*
    a) wrote a term paper
    b) wrote a thesis to gain a scientific degree
    c) published a scientific paper
    d) held a talk on a conference or workshop
    e) published a doctoral thesis

Usage of academic search engines

14. How often do you use academic search engines (such as ACM Digital Library or Google Scholar) during your research?
    a) 7 point Likert scale: never ... exclusively
15. Which of these academic search engines are you using for literature research? *[multiple choice]*
    a) Google Scholar
    b) Microsoft Academic
    c) IEEE Xplore
    d) ACM Digital Library
    e) arXiv
    f) CiteSeerX
    g) Other: ___________
16. Some academic search engines offer advanced features that generic web search engines typically do not offer. Which of these features do you use? *[multiple choice]*
    a) find other papers of an author
    b) filter by release date
    c) find papers that quote a publication
    d) find papers with similar content
    e) export Bibtex quotation
    f) others

Task Review

This is a reproduction of the task review questionnaire. The titles between the questions have been inserted for convenience and were not part of the original form. Processing instructions are placed within square brackets.

> ### System Usability Scale (SUS)
>
> *[All SUS statements use a 5 point Likert scale from strongly disagree to strongly agree]*
>
> Please rate your experience with the search engine you've been using on a scale from 1 (strongly disagree) to 5 (strongly agree)
>
> 1. I think that I would like to use this system frequently.
> 2. I found the system unnecessarily complex.

3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Task Rating

[For the rest of the questionnaire, a 7 point Likert scale from strongly disagree to strongly agree is used for all statements]

Please rate your experience with the task you've been assigned on a scale from 1 (strongly disagree) to 7 (strongly agree)

11. I had enough time to finish the task
12. I found the task was very simple
13. I was able to draw on a lot of previous knowledge, which helped me with the task
14. I was overwhelmed with the task
15. I was able to gain a very good overview of the topic at hand
16. I am very pleased with the result of my research

Search Engine Features

Please rate your experience with the search engine you've been using on a scale from 1 (strongly disagree) to 7 (strongly agree)

17. The search results fit very well with my search terms
18. The presentation of the search results was very clear
19. The filter functions (e.g. year, author) were very useful
20. I was able to judge the relevance of a search result just by looking at its preview
21. Among the search results were mostly unfitting documents
22. Among the search results were often relevant documents which I would not have expected based on my search terms

Features of the Prototype

[only displayed, if the prototype was used. For questions 23-25, explanatory screenshots were provided]

Please rate the features of the prototype on a scale from 1 (strongly disagree) to 7 (strongly agree)

23. The additional tools in the right column (sidebar) were very helpful to me
24. The terms in the "topic centroid" (sidebar) represented the topic of my research very well
25. The suggested search results (sidebar) were relevant to my search
26. The possibility to save search results by clicking ★ was very useful to me
27. The possibility to navigate freely in the search history was very useful to me

A.2.4. Participant's Answers

Demographic Questionnaire

Participants (1/2)

	P1	P2	P3	P4	P5
1.	male	male	male	female	female
2.	18-24	25-34	18-24	18-24	18-24
3.	C1	C2	B2	B2	B1
4.	Bachelor	Master	High School	High School	High School
5.	yes	yes	yes	yes	yes
6.	Computer Science	-	Computer Science	Mechanical engineering	Mechanical engineering
7.	9-12	13+	6-8	6-8	3-5
8.	Computer science	Computer science, Other engineering	Computer science	-	Mechanical engineering, other engineering
9.	0-1	4-5	0-1		2-3
10.	a)	a)	a)	c)	a)
11.	b), c), d)	a), b), c), d), e)	a), b)	f)	a)
12.	4	3	4	3	1
13.	a), b), d)	a), b)	a)	-	a)
14.	3	1	5	5	1
15.	a)	-	a)	a)	a)
16.	a), d)	-	c), e)	-	b)

Participants (2/2)

	P6	P7	P8	P9	P10
1.	male	male	male	male	male
2.	25-34	18-24	18-24	25-34	18-24
3.	C2	B2	C1	B2	C1
4.	High School	High School	High School	Bachelor	High School

Participants (2/2)

	P6	P7	P8	P9	P10
5.	yes	yes	yes	yes	yes
6.	Computer Science	Mechanical engineering	Physics	Computer Science	Computer Science
7.	6-8	6-8	6-8	9-12	6-8
8.	Physics, Computer science	-	-	Computer science	-
9.	2-3	0-1		0-1	0-1
10.	a)	a)	a)	a)	a)
11.	d), e)	f)	b)	a), b)	a), b), c), e)
12.	5	2	1	4	4
13.	a), d)	a)	d)	a), b)	a), b)
14.	3	2	1	2	6
15.	c), d)	a)	-	a), c), d), f)	a)
16.	a), e)	-	-	a), e)	a), b), c), d), e), f)

Task Review: Google Scholar

Participants

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
					SUS					
1.	5	3	3	3	2	2	2	2	2	2
2.	1	2	2	2	3	1	2	3	2	3
3.	5	4	4	4	3	5	4	2	4	4
4.	1	2	2	2	4	1	1	2	2	1
5.	5	5	4	2	3	3	3	3	4	2
6.	1	1	2	2	4	1	1	4	2	3
7.	4	4	3	4	5	4	3	1	3	4
8.	1	2	2	2	2	3	3	4	1	5
9.	5	4	2	3	1	5	3	3	4	3
10.	2	1	2	2	3	1	2	2	2	2
					Task Rating					
11.	6	3	2	3	5	7	5	2	2	6
12.	5	2	2	4	3	2	3	5	4	4
13.	1	1	1	2	2	4	1	3	2	2
14.	1	2	5	4	3	1	1	2	4	2
15.	2	5	2	2	2	5	4	1	2	4
16.	5	3	4	3	1	4	3	2	1	3
					Search Engine Features					
17.	6	4	3	4	6	3	4	2	2	2
18.	7	5	2	4	4	2	4	2	4	2
19.	4	5	2	6	5	1	3	3	4	4
20.	3	4	2	4	2	1	2	1	5	3
21.	3	5	2	3	3	3	3	4	6	2
22.	2	2	5	2	4	7	2	3	3	1

Task Review: Prototype

	Participants									
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
SUS										
1.	5	4	4	4	5	3	3	4	4	5
2.	1	4	2	2	1	2	2	1	2	2
3.	5	4	4	4	4	4	3	5	4	4
4.	1	3	2	2	1	1	1	2	1	1
5.	5	4	5	4	5	5	4	4	5	4
6.	1	3	2	2	3	1	2	2	2	2
7.	4	3	4	4	5	4	3	4	4	5
8.	1	2	2	1	1	1	3	2	1	1
9.	5	3	3	1	3	5	2	3	3	5
10.	2	2	2	2	2	2	2	2	3	1
Task Rating										
11.	5	3	3	3	3	6	5	6	5	5
12.	5	2	1	4	3	6	4	5	3	5
13.	1	1	2	2	1	5	2	5	1	2
14.	1	4	5	3	2	1	1	2	2	1
15.	2	5	5	3	5	3	5	4	4	4
16.	6	3	5	3	6	5	6	6	4	4
Search Engine Features										
17.	4	2	5	4	6	7	5	5	5	6
18.	7	5	7	6	7	7	3	6	6	6
19.	4	5	6	3	5	4	3	6	4	5
20.	3	2	7	4	6	5	6	3	6	6
21.	1	5	5	2	1	2	2	2	2	5
22.	3	3	2	3	5	5	4	3	6	4
Features of the Prototype										
23.	4	6	6	-	6	4	4	6	5	4
24.	4	2	-	-	-	5	3	4	5	5
25.	4	6	6	-	4	5	2	3	6	-
26.	7	7	7	7	7	7	6	7	7	7
27.	5	3	5	6	7	6	3	6	6	7

A.2.5. Participant's Search Results

The results of the participant's literature research are listed in this chapter. The next table specifies which tasks have been fulfilled by each participant.

Participant-Task-Mapping

Participant	Type	Task	Engine	Order
P1	CS	Graph	Google Scholar	2
P1	CS	Crowd	Prototype	1
P2	CS	Graph	Google Scholar	1
P2	CS	Crowd	Prototype	2
P3	CS	Crowd	Google Scholar	1
P3	CS	Graph	Prototype	2
P4	PH	Hubble	Google Scholar	2
P4	PH	Graphene	Prototype	1
P5	PH	Graphene	Google Scholar	1

Participant-Task-Mapping

Participant	Type	Task	Engine	Order
P5	PH	Hubble	Prototype	2
P6	CS	Crowd	Google Scholar	2
P6	CS	Graph	Prototype	1
P7	PH	Graphene	Google Scholar	2
P7	PH	Hubble	Prototype	1
P8	PH	Graphene	Google Scholar	1
P8	PH	Hubble	Prototype	2
P9	CS	Crowd	Google Scholar	1
P9	CS	Graph	Prototype	2
P10	CS	Crowd	Google Scholar	2
P10	CS	Graph	Prototype	1

The next four tables specify the results that have been saved by the participants for each task, as well as their ratings (ranged from 1 to 5). Documents are defined by their arxiv ID, which can be easily requested by pointing a web browser to `http://arxiv.org/abs/<arXiv-ID>`.

Crowd Work(CS, A)

arXiv ID	Scholar				Prototype	
	`P3`	`P6`	`P9`	`P10`	`P1`	`P2`
`1108.1682`		1				
`1204.3342`		3		3	2	
`1205.3856`				2		
`1301.2774`	4					
`1304.3548`				3	4	4
`1309.3330`					5	
`1310.1672`		5	3	4	3	5
`1310.5142`			4			
`1310.5463`	3					
`1311.2349`			3			
`1403.0783`						3
`1406.7588`				5		
`1501.06313`		4				
`1502.07710`					5	
`1503.05897`						5
`1511.06053`				4		
`1609.01017`						
`1609.04855`						4
`1609.04855`						
`1610.06210`		1				
`1611.01572`		5			3	4
`1611.02682`					1	
`1702.01661`	4					
`1702.04214`						4
`1702.08571`	5					
`1706.10097`		5		2		

Distributed Graph Processing(CS, B)

arXiv ID	Scholar		Prototype			
	P1	P2	P3	P6	P9	P10
0809.3232	4					
0910.2004			4			
0911.2280						4
1002.4482	5					
1006.2880						4
1102.0694						
1110.5844	2					
1202.6158	4					
1205.6343						2
1208.1926			4			
1208.3071				5		5
1210.0530				3		
1211.6159			4	3		
1302.6636						
1305.3178				2		
1309.1049		4				
1310.5603	5	5			3	
1311.6209					5	
1312.3018	3	4			4	
1402.1472				2		
1403.6270		5	4			
1404.3861						
1404.4887						
1407.5107						3
1408.3030			3			
1502.04281						5
1503.03155						5
1507.06702			3	5		
1509.00016						3
1509.01476						3
1510.03145					4	
1511.04925						4
1602.03718			4			
1603.04467		2				
1607.02646					5	
1609.09068						3
1611.01907						3
1611.02663		3				
1701.00503						
1701.00546					4	
1703.10628					4	
1704.02003					3	
cs/0310049	2					
cs/0412002						

Hubble Constant(PH, A)

arXiv ID	Scholar	Prototype		
	P4	P5	P7	P8
0709.3924		2	4	3
0902.4680	3			
1003.2185	3			
1004.1856		3	5	5
1005.0263		2		
1006.5888	3			
1010.2679		3		
1202.4459	4			
1303.2063	3			
1309.4118			2	
1406.1718		3	4	3
1606.07316				4
1607.03537			3	
1607.06256		5	4	3
1704.04784	2			
1706.09149	4			
1707.00715	4			
astro-ph/0305060			4	
astro-ph/0309122			3	2
astro-ph/0407430				2
astro-ph/0603643				3
astro-ph/9602123			2	
astro-ph/9604064			2	
astro-ph/9707101		4		
astro-ph/9801315			3	
astro-ph/9805136				2
astro-ph/9810393			3	
astro-ph/9909076			3	

Graphene Supercapacitors (PH, B)

arXiv ID	Google Scholar			Prototype
	P5	P7	P8	P4
1101.1064				3
1104.3379				2
1108.2331		3		
1109.0978				3
1311.1413	3	4	5	3
1311.1548		4		
1311.7529		2		
1409.6396	2		3	
1410.0767	5		4	
1410.4223		3		
1411.6278		3		
1512.08000		3		

Graphene Supercapacitors (PH, B)

arXiv ID	Google Scholar			Prototype
	P5	P7	P8	P4
1512.08288		3	4	4
1601.05173		4	3	
1603.08320			2	
1702.02031	4			
1704.03227			2	
1704.08405		2		
cond-mat/0509355			1	

List of Figures

3.1. Query weights depending on the index position of the query and the length of the session . . . 22
3.2. Topic identification . . . 24
3.3. Visualization of the topic centroid. Each node represents a topic, surrounded by the distinctive terminology for that topic. Directed edges indicate hierarchical relations between topics. . . . 31
3.4. From request to response: steps in the query pipeline . . . 36

4.1. Screenshot of the prototype's search engine results page . . . 52
4.2. Network architecture of the search engine . . . 53
4.3. Screenshot of the prototype's search engine results page . . . 60
4.4. A single document on the search result page, with a preview of its abstract . . . 61
4.5. Visualizations of the topic centroid . . . 62
4.6. Average response time relative to the load on the search engine (number of requests per minute) . . . 64

5.1. Experience of participants and task ratings . . . 71
5.2. Analysis of SUS scores . . . 72
5.3. Participant's satisfaction with the search engines and their research . . 74
5.4. Selection of search results . . . 75