Semantic Search with Keyword Queries

master's thesis
by
Eugen Sawin

advisor
Prof. Dr. Hannah Bast
Department of Computer Science

University of Freiburg
79110 Freiburg, Germany
February 2014

© 2013-2014 Eugen Sawin
All Rights Reserved.

Acknowledgments

I FEEL VERY FORTUNATE FOR THE FREEDOM that I was granted during the years of my studies. The freedom to follow my passion and choose the topics of my interest and the freedom to develop and apply my method of working with such great support. I thank *Hannah Bast*, who inspired and powered my work on this thesis, *Björn Buchhold* and *Elmar Haußmann* for their general assistance and *Mirko Brodesser* for his critical proof reading of this text. My special thanks go to *my family* and to *Sam* for enduring me in this time of excessive freedom.

Contents

Semantic Search with Keyword Queries

Abstract

We present a semantic search concept, which effectively combines the simplicity of keyword-based search with the sophistication of semantic results. Our bootstrapping process is designed to utilize the efficiency and precision of full-text search to provide reliable intermediate results during the retrieval phase. In the second phase, we perform a deep semantic search, based on the previous results for their refinement. The introduced methods include lexical query analysis, ranking and answer selection algorithms and a variety of filtering solutions. We argue that a simple query interface based on keywords is essential to unlock the potential of semantic search technologies and make them accessible for a greater audience. Finally, we provide Pythia – an open source implementation of the two-phase approach – to support further practical research on this topic.

Semantic Search with Keyword Queries

Zusammenfassung

Wir präsentieren ein Konzept zur semantischen Suche, das die Bedienungsfreundlichkeit von Schlüsselwort-basierter Suche mit der Präzision semantischer Resultate vereint. Das entwickelte Zweiphasenverfahren profitiert im ersten Schritt von der Effektivität der Volltextsuche um verlässliche Zwischenresultate zu generieren. Basierend auf den Ergebnissen der ersten Phase, wird als nächstes eine semantische Suche durchgeführt, deren Resultate zur Verfeinerung des Gesamtergebnisses beitragen. Die vorgestellten Methoden umfassen eine Query-Analyse, ein Ranking-Verfahren und eine Auswahl von Verfeinerungstechniken zur Verbesserung der Ergebnisse. Zur Evaluation der Methode wird die Suchmaschine Pythia entwickelt und zur weiterführenden Forschung zur Verfügung gestellt.

If you don't know where you are going,
any road will get you there.

Lewis Carroll

1 Introduction

This thesis portrays the creation of Pythia – an experimental engine for semantic search with keyword queries. In section 1.1 we show our motivation for combining keyword-based queries with semantic search, give an overview over the main contributions of this work in section 1.2 and conclude this chapter with a structure outline of this paper in section 1.3.

1.1 Motivation

The dawn of the World Wide Web has created a new world of information unprecedented in growth and variety. However, its distributed nature does not support content discovery. Web search providers have established the de-facto centralized access to the Web, enabling exploration and research using simple keyword queries. The full-text web search engines have reached a high level of maturity in the 2000s and find relevant results with high precision and efficiency. Moreover, they have become an integral part of our lives and habits. With ubiquitous access through mobile devices, web search has even started to transform our memory functions and the way we think [7]. What web search engines have not managed to date, is to actually answer the questions we ask. Let us clarify this.

More than 40% of all conducted web searches are *entity searches*. We search for names, products or locations, yet what we get are HTMLs, PDFs or worse. When we search for *"books written by lewis carroll"*, we want to know the *book titles* of the published work by Charles Lutwidge Dodgson, including his popular *"Alice's Adventures in Wonderland"*, the sequel *"Through the Looking-Glass"* and *"The Hunting of the Snark"*. Using the query on a full-text search engine returns a ranked list of *document links*, which may, or may not contain the actual book titles we are looking for, since the documents are only guaranteed to contain the *words "books"*, *"written"*, *"Lewis"* and[1] *"Carroll"* – this is called the *document-centric* approach. On the other hand, a semantic search engine attempts to answer the query by consulting its knowledge base to find all *entities* of *type* book, which stand in the *relation* of being written by Lewis Carroll (Fig. 1.1.1), which, we argue, should be the natural way of answering such a query and is called the *entity-centric* approach.

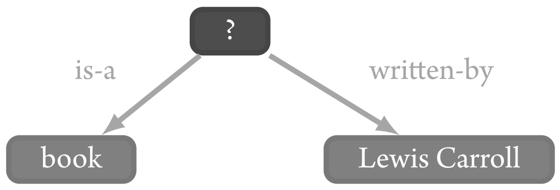


Figure 1.1.1: Query graph for *"books written by Lewis Carroll"*

A more complex example query is *"female computer science professors in Germany"*. Professors rarely specify their gender in the job description, so matching the word *"female"* in relevant documents is problematic. Our only hope is for someone to compile a list of all female computer science professors in Germany and to name it accordingly. Obviously such a solution doesn't scale well. This is where a good semantic search engine can shine, as long as its knowledge base contains the person entities in question and all required relations. Figure 1.1.2 shows an example graph representation of the query, the specific form depends on the types of entities and relations that are contained in the ontology.

Semantic queries can be complex, but the complexity is required to utilize the full potential of the underlying ontology. The developers of semantic search engines do their best at hiding the query complexity or they guide the user to the correct query by visual schemes or auto-completion. However, commercial search engine providers do not adopt such complex queries mechanics for a good reason: *forcing user adaption is a bad business practice.*

Our work in this thesis contributes to the research of providing the best of both worlds –

[1]Depending on the search engine, partial matches can also be supported

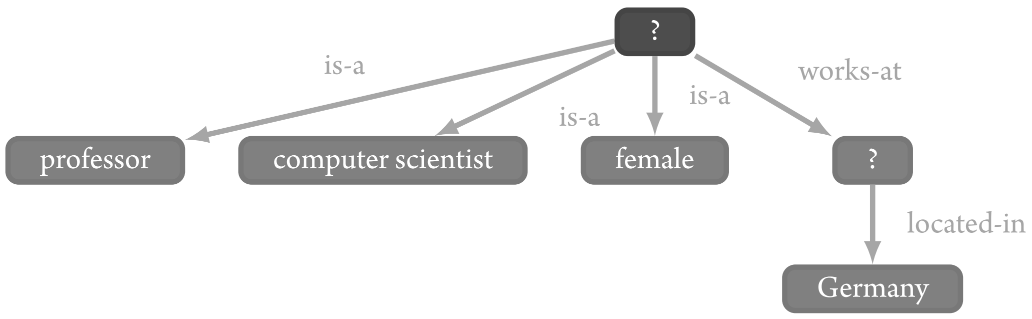


Figure 1.1.2: Query graph for *"female computer science professors in Germany"*

the accessibility of keyword-based queries and the sophistication of semantic search. In our bootstrapping approach, we first perform an inexact web search using keywords to extract candidate answer entities – we call it the *entity retrieval phase*. Next, we use these intermediate results to construct a structured query. In the second phase, we use the structured query on a semantic search engine to retrieve the semantically sound results – this is called the *deep search phase*.

1.2 Contribution

Semantic search provides access to the human knowledge in a rigorous way. For semantic search to work, the knowledge base is translated from natural language to a formal model. Accessing information on that model requires formal query languages, which effectively means that we lose the ability to use natural language to interface with the data.

The success of a technology depends highly on the ratio between the value it provides and the efficiency of its interface. Semantic search provides great value, but requires streamlining of its interface to be adopted by a wider range of audience.

We would like for semantic search to succeed and to make it available to a wider audience by removing the obstacle of a complex and inconvenient interface. With this thesis, we contribute to the research in providing an easy-to-use interface for semantic search, including:

1. Quantitative entity retrieval based on full-text search and semantic processing
2. Rule-based lexical answer type detection
3. Quantitative semantic answer type inference
4. Rule-based translation of keyword queries to semantic queries
5. Open source experimental search engine

For this, we analyse the parts that make keyword-based semantic search possible, provide solution and verify them using our search engine Pythia. The source code of Pythia and all auxiliary materials are available online[2] under the *GNU General Public License* to allow for for a frictionless reconstruction of the conducted experiments and the continuation of this research.

1.3 Thesis Structure

We have structured the thesis as follows.

Chapter 1: *Introduction* is what you are reading now. Here we show our motivation, enumerate our contributions and describe the thesis structure.

Chapter 2: *Related Work* reaches out into the research community and compares related work.

Chapter 3: *Foundations* establishes the theoretical framework and gives an overview of the utilized techniques.

Chapter 4: *Semantic Search with Keywords* is the main chapter describing the proposed methods to enable keyword-based queries for semantic search.

Chapter 5: *Implementation* describes critical details of our experimental search engine Pythia.

Chapter 6: *Evaluation* provides an in-depth performance analysis of our approach using Pythia and narrows down sources of errors for future research.

Chapter 7: *Discussion* addresses the results and proposes ideas for future work.

[2]https://github.com/eamsen/pythia

"But I don't want to go among mad people", said Alice.
"Oh, you can't help that" said the cat. "We're all mad here".
Lewis Carroll

2 Related Work

For the discussion of related work, we compare the two phases of our approach separately – the *entity retrieval phase* and the *deep search phase*. For the first phase, we analyse comparable systems and their performance in section 2.1. The second phase is the *construction of the semantic query* followed by a *deep semantic search*. Since our approach is unique in this regard, we provide a more general discussion of alternative attempts to achieve the same goal in section 2.2 – that is to provide an easy-to-use interface for semantic search.

2.1 Entity Retrieval on the Web

In the first phase of our approach, we search the web for relevant documents to leverage the quality of full-text search to support the analysis and translation of the query in the next phase. The intermediate results are the ranked semantic entities extracted from the documents. There is a variety of comparable systems from different conference tracks, which aim to provide entity search on the web. We decided that the *TREC 2010 Entity Track* provides the most complete results and only minor deviant restrictions compared to our setting. From this track, the *Related Entity Finding Task* is what essentially resembles the goal of this thesis, but we explain the differences to our approach and constraints in 2.1.2. First, we give a brief overview of the Related Entity Finding Task from the Entity Track, followed by an analysis of some of the presented systems.

2.1.1 Related Entity Finding Task

The goal in the task is to automatically find all *answer entities* for each given *query* and identify them uniquely with the entities' *homepages*.

Here is an example query:

```
<query>
  <entity_name>David Bowie</entity_name>
  <entity_url>davidbowie.com</entity_url>
  <target_entity>organisation</target_entity>
  <narrative>
    What record labels sell David Bowie recordings?
  </narrative>
</query>
```

The query contains an *entity name* with its *identifying URL*, the *answer entity type* and the *narrative*, which describes the relation between the requested entities and the given one. The expected result is a ranked list of *entity names* and their *homepages*.

2.1.2 General Method

The most successful systems used similar approaches. The common methods are as follows.

1. Construct a keyword-based query from the given narrative.
2. Use the query on a web search engine to collect relevant documents.
3. Parse the documents using NLP tools to retrieve named entities.
4. Rank the retrieved entities using a linear combination of scores collected during the previous steps.
5. Use web search again to find the entity homepages.

Steps 1-4 describe the outline of our approach, too, which makes the comparison to those systems interesting. What are they doing differently? Which enhancements are effective? What distinguishes the best systems from the others?

To our disappointment, step 5 – which results from the task's requirement for returning *homepages* owned by the entities, not just *entity names* – prohibits any direct comparisons between our evaluation results and the top performers of the track. And we note that the query does provide *additional context* for the search with the explicit entity name and resolved answer entity type. In comparison, our research is concerned with the task of performing the search only on the narrative and *without additional hints*.

2.1.3 Top-Performing Systems

The top-performing system BIT [11] of the track yields a heavy-weight offline search to find the most relevant documents during step 2. For this, they scan all documents for navigational menus, which are expected to be encoded in HTML lists or tables. They extract the hierarchical site maps from the identified menus and use that to enrich the link anchor texts. The Indri search engine relies heavily on link anchor texts for the assessment of a document's relevance and they use this fact to boost the performance during the document retrieval phase. Their precision results are around 30%, which is promising compared to other systems. However, they have missed to provide evidence on the positive effects of the anchor text enrichment. The good precision could be a result of other aspects of the process. We conjecture that the local search on the ClueWeb09[1] – a snapshot of a subset of the English web – using Indri could have made all the difference to the competing systems, which used filtered web search results instead.

The second best system of the track was FDWIM [10], with only slightly worse results than BIT. Its straight-forward approach follows the scheme we have outlined, using Google for web search and Stanford NER for the extraction of named entities. Their experiments have revealed that deeper analysis of certain pages – like the Wikipedia page of the given entity – can be beneficial. Furthermore they conclude that most errors are connected to inaccuracies of the NLP tools and propose to explore methods to reduce the strong dependence on natural language processing in general.

2.1.4 Summary

Entity Retrieval is a hot topic in research and there are some first promising results. However, the conference tracks are still young and require some iterations to become practically relevant and to establish a realistic and consistent environment for a fruitful competition.

[1]http://lemurproject.org/clueweb09

The approach used by the more successful systems in this track essentially resembles the first phase or our method. We think that some of the proposed optimizations could improve the performance of our method for the entity retrieval phase. However, most improvements are based on structural or lexical assumptions, which could decrease the generality and robustness when confronted with unformatted input or non-English corpora.

2.2 User Query Interfaces

There is a variety of ways to provide user interfaces for semantic search engines. We evaluate some interface types for their *usability*, *expressive power* and *robustness* regarding malformed user input.

2.2.1 Natural Language Queries

Taking the first example from the introduction, a possible natural language query would be

"Which books did Lewis Carroll write?"

The query is *easy to read* and the machine has enough context when applying *natural language processing* to extract semantics. With natural language we can compose questions of *arbitrary complexity* with no limit on expressiveness. However, queries in natural language are *verbose* to write and the result precision relies heavily on the *quality of the natural language processing* tools. The AquaLog system [6] fails in 69% of the test cases to translate the question into a proper semantic query because of the inexact natural language processing.

2.2.2 Keyword-based Queries

Finally, let's look at engines accepting keyword queries and how they differ from our implementation. Understanding keyword queries is hard. We are particularly interested in robust semantic keyword interpretation without drawbacks on usability.

SemSearch [5] translates user queries to formal queries using syntax-based mapping of keywords to classes, instances and properties of the underlying ontology. It does not apply natural language processing or quantitative methods to gain robustness in the translation. As a consequence it requires a *custom query format*, which exposes the result type and limits the query to a conjunction or disjunction of keywords to be associated with the result entity. The mapping of keywords to semantic items leads to a combinatorial explosion when more than two keywords are provided. To cope with the exponential growth of the number of semantic mappings, they apply a heuristic-based selection to choose one viable mapping. Even though we consider the custom query format to be lightweight and easy to learn, it still forces the user to adapt to an unnatural way of posing questions and limits the expressiveness of the queries. We also think that the mapping selection heuristic is critical to the success of the method, but at the same time is difficult to optimize and evaluate.

XXploreKnow! [9] is similar to SemSearch as it attempts to map the query keywords to semantic items of the knowledge base in the first step. Additionally, starting from the identified entities and classes, the knowledge base is explored to infer the relations between the identified items. The explored ontology relations are finally mapped to semantic queries to initiate the search procedure. Just like in [5], the approach suffers from the fast growth of possible combinations for the keyword mapping, their solution is to let the user interactively refine the query. Since the details of their evaluation are not exposed in [9], it is unclear how to rate the reported precision values of 69% for fully automated and 85% for user-refined search.

In summary, related keyword-based systems apply rule-based semantic mapping, resulting in high precision variation due to the dependence on full ontology support and complex and therefore error prone mapping mechanics. We see such an approach critical in regards to practicality and robustness.

In the beginning there was nothing, which exploded.

Terry Pratchett

3 Foundations

Semantic search is a huge and diverse domain. This chapter provides a brief overview of the foundations of full-text search in 3.1 and natural language processing in 3.2. Pythia is the experimental semantic search engine we built to test the proposed methods within this thesis. It makes use of existing tools and APIs for full-text search, natural language processing and finally interacts with a semantic search engine. It is important to understand the basic mechanics of these tools and especially to know their performance complexities to validate the practicality of the proposed methods.

3.1 Full-Text Search

Web search is a solved problem. It's a bold statement, but it's backed up by high-precision results on indices of 50 billion pages in sub-second time frames, as provided by commercial web search engines.

During the first phase, Pythia depends on the result quality of full-text search. To guarantee the practicality of this approach, we need to research the efficiency and measures to increase the precision of full-text search. The industry has developed techniques and architectures to battle the exponential growth of the Web. Let us examine the basic principles these techniques build on.

3.1.1 Inverted Index

Assume a huge collection of text documents. Each document may be encoded as an index of words, see example tables 3.1.1-3.1.3.

index	1	2	3	4
word	I	am	the	walrus

Table 3.1.1: A: *"I am the walrus"*

index	1	2	3	4
word	I	am	the	walrus

Table 3.1.2: B: *"I am the egg man"*

index	1	2	3	4	5
word	They	are	the	egg	men

Table 3.1.3: C: *"They are the egg men"*

index	1	2	3
document	A	B	C

Table 3.1.4: Index for A, B and C

The documents can be made accessible by the document index, see table 3.1.4. This is essentially how documents and a folder of documents can be represented. It allows for efficient access by document and word index, but it is considerably slow to search for documents containing a specific word.

More precisely, matching a word in the whole index accumulates to a time complexity of $O(n)$, where n stands for the total number of characters in all documents combined.

Back to our example, with an index holding 50 billion pages, an average web page size of 170 KB and a scanning speed of 1 GB per second, it would take over 93 days to search through all documents for one word.

A more practical solution is the *inverted index*. The idea is to provide efficient access by keywords instead, see table 3.1.5. For example, to find all relevant documents for the keyword *"egg"*, we just need to look up the document list for that word – all relevant documents for the keywords *"egg"* and *"man"* can be computed by the intersection of the two lists, in our example it is $\{B, C\} \cap \{B\} = \{B\}$, just as expected.

keyword	i	am	the	walrus	egg	man	they	are	men
documents	{A, B}	{A, B}	{A, B, C}	{A}	{B, C}	{B}	{C}	{C}	{C}

Table 3.1.5: Inverted index example

The inverted index allows for constant time access for single keyword queries. For multiple keywords we need to intersect the lists – with sorted lists this yields a time complexity of $O(k \log n)$, where k is the combined number of elements in all lists and n the number of lists. The keyword-based lookup can be implemented using a hash table with amortized constant time access. It should be also noted that both words and documents would be addressed using integer ids, not their original strings, for improved time and memory efficiency.

3.1.2 Document Ranking

The inverted index allows us to efficiently find all relevant documents, but the result could contain thousands or millions of documents. To make the results useful for the user, we need a finer notion of relevance. Here are some examples of existing ranking formulae.

Binary

This most basic form ranks all documents containing the keywords equally.

Term Frequency

The document rank depends only on the frequency of the keywords *tf* in the document. This is a good base score, which is used in most ranking approaches with some refinement.

TF-IDF

Answering queries with multiple keywords based only on term frequency will skew the results towards documents containing the most dominant keywords, like *"the"* or common verbs. To suppress the relative document frequency of a keyword, we produce the *tf.idf* score by multiplying the term frequency with the inverse document frequency:

$$tf.idf = tf \cdot \log \frac{n}{df}$$

where *tf* is the term frequency, n is the total number of documents and *df* is the number of documents containing the keyword.

BM25

A refinement of the *TF-IDF* formula is the *BM25*, which is known for outperforming other derived versions in general. For the BM25, we replace the plain term frequency with the parametrized expression tf^*. The parameters b and k can be used for fine tuning, for example, we could emulate TF-IDF by setting $b = 0$ and $k = \infty$. The default BM25 setting is $b = 0.75$ and $k = 1.75$.

$$bm25 = tf^* \cdot \log \frac{n}{df} \quad \text{with} \quad tf^* = \frac{tf(k+1)}{k(1 - b + b \cdot \frac{dl}{avdl}) + tf}$$

where tf is the term frequency, n is the total number of documents, df is the number of documents containing the keyword, dl is the document length and $avdl$ is the average document length.

3.2 Natural Language Processing

Natural language is ambiguous, which makes it difficult to build deterministic machines to process it in a reliable way. For some limited tasks, rule-based systems produce good enough results efficiently. However, such systems are typically tied very close to a specific language and depend on properly formed complete sentences. Statistical methods on the other hand, show robust behaviour for different languages and generally outperform rule-based systems in precision.

To move from the *bag-of-words* processing of full-text search to *semantic reasoning*, we need methods to connect words to meaningful questions. As a base, that means we need to identify the *parts of speech*: nouns, verbs or adjectives. Next, we need to understand higher level fragments of sentences: subjects, objects and predicates. Additionally, we are interested in locating *named entities*: names of locations, products and people.

In our use case, user queries may be short phrases or only a collection of keywords. Probabilistic methods allow us to identify the parts of speech even when only keywords are provided. This is important, because our query analysis (4.3) is based on the part of speech tags.

3.2.1 Part of Speech

The foundation of our semantic query and document analysis are the *part of speech* (POS) tags. POS-tagging classifies each word of a sentence into a category, such as noun, verb, adjective or adverb. Let us examine the example sentence *"The cat grinned when it saw Alice"*.

word	*"The"*	*"cat"*	*"grinned"*	*"when"*	*"it"*	*"saw"*	*"Alice"*
POS	determiner	noun	verb	adverb	personal pronoun	verb	proper noun

Throughout this text, we only show the basic version of the part of speech tags for clarity. Most tagger systems produce UPenn Tree Bank tags, which are more specific – in our example, *"grinned"* and *"saw"* would be tagged as *VBD* (verb, past tense) and *"when"* as *WRB* (wh-adverb). The example shows that POS-tagging is case sensitive: *"The Cat"* would be tagged as a proper noun and depends on the context: *"saw"* can refer to an activity (verb) or a tool (noun).

We claimed that natural language is inherently ambiguous, so let us look at an example for this. The sentence

"flying planes can be dangerous"

could mean that either the *act of flying* planes is dangerous or that *planes that are flying* are dangerous. It's a classical example for *syntactic ambiguity*. In this case *"flying"* could be tagged as a verb or adjective, both is considered to be correct.

Effective POS-tagging algorithms are based on rule-based or statistical methods. An early success story was the rule-based the Brill tagger [2], however it was beaten in terms of precision by Stanford's POS tagger [8] using a maximum entropy approach. A notable system is SENNA [4], which produces competitive results at improved time efficiency utilizing an artificial neural network.

3.2.2 Named Entity Recognition

Named entities are specific persons, locations, organizations, products and everything that has a *unique identity*. In the lexical context, a named entity is a *proper noun* classified into a predefined category, further referred as its *type*.

In our case, semantic search is concerned with returning named entities. The process of identifying named entities in text and annotating such with their type is called *name entity* *recognition* (NER). For example, the named entity in the sentence

"Mars is named after the Roman god of war."

is *Mars*, the fourth planet in our solar system. However, in the sentence

"Can you live on Mars?"

it could be the planet or the Mars chocolate bar. The Mars examples show that entities' names are not unique and therefore not sufficient for their classification – in this case we have one name for three types of entities: a planet, a god and a chocolate bar. Naturally, the name-type combination of an entity does not identify it uniquely either – we can have millions of *persons* named *John Doe*. But this is not an issue for NER – name ambiguity, however, is a major problem for this task of identifying the correct answer entities in semantic search. We address this again in chapter 4.

As with most natural language processing tasks, the main difference between the approaches used is the reliance on human-crafted rules or on stochastic models. Systems achieve near-perfect results in the detection of named entities using both approaches, with different trade offs. The unresolved problem is the classification of the identified entities. There are multiple entity class hierarchies with different grades of granularities. To this date, no de-facto standard has emerged for this, which makes it difficult to unify and compare results of different systems.

State-of-the-art named entity recognition systems like Stanfords' NER achieve great results, closing in on human performance. Again, the SENNA system shows a well balanced solution between precision, recall and performance.

When I use a word, it means just what I choose it to mean – neither more nor less.

Lewis Carroll

4 Semantic Search with Keywords

Technology amplifies human abilities. We don't grow four legs to move faster – we build cars. Web search providers learned that we have two dominant ways of asking questions – *full phrases* when using voice and *keywords* when typing. This behavior is analog to the difference between the complex mechanics of walking and the simple control mechanisms of driving. When driving, we tap the pedals to control acceleration and move the steering wheel to control direction. It is an efficient way of signaling our intention in the same way we use keywords to find relevant documents.

By our nature, we prefer to change our environment rather than having to adapt to it. Typing long phrases is inconvenient, just like adapting to complex query schemes and user interfaces. Our goal in this thesis is to transfer the *user experience* of full-text search to semantic search – to establish methods that *understand* a user query given only *minimal context*. We want semantic search technology to adapt to our way of asking questions and that way provide a seamless transition from conventional full-text search.

The high-level components of a search engine are its *user interface*, *knowledge base* and *answer selection* engine. The emphasis of this thesis is on the quality of the answer selection, with the goal to enable a convenient keyword-based user interface. We are not concerned about the specifics of the user interface besides the outlined principles in 4.2, or the data format of the knowledge base.

When presenting our method, we strive to isolate the problems of *query analysis*, *answer entities selection* and *result refinement* to find system-agnostic solutions.

In this chapter, we propose a method to enable keyword queries for semantic search by combining full-text search results with semantic analysis.

4.1 Two-Phase Approach

In chapter 2 we discussed related systems with a qualitative approach to semantic search, including query analysis. We find that a direct semantic translation of keywords to a formal query is a difficult task to undertake and prone to errors. We refrain from using such bold direct translations and propose a quantitative approach instead.

The two phases of our approach are *entity retrieval* and *deep search*, see figure 4.1.1 for an overview. In the entity retrieval phase, we use traditional full-text search to find relevant documents to the user query and retrieve relevant entities from these unstructured documents. The obtained results – such as the frequency of certain entities in the texts and their types – give us a base for quantitative reasoning. The second phase begins with a semantic analysis of first phase's results. We infer the semantic answer type (SAT) based on type frequency in the retrieved entities. Furthermore, we try to obtain the relation between the requested answer entities and the given keywords. On successful semantic query construction, we use a semantic search engine to produce the final semantic answer entities, which can be used as a replacement or base for verification of the intermediate answer entities from the first phase.

By using this two-phase approach, we strive to gain more robustness in providing the right answers by decreasing the dependence on the knowledge base and natural language processing tools – which are both open research problems without perfectly practical solutions. At its core, the entity retrieval phase is a quantitative replacement for the semantic query analysis of related systems. The first phase is independent from the semantic knowledge base and requires only named entity retrieval, and we foresee purely lexical replacement methods for that, too.

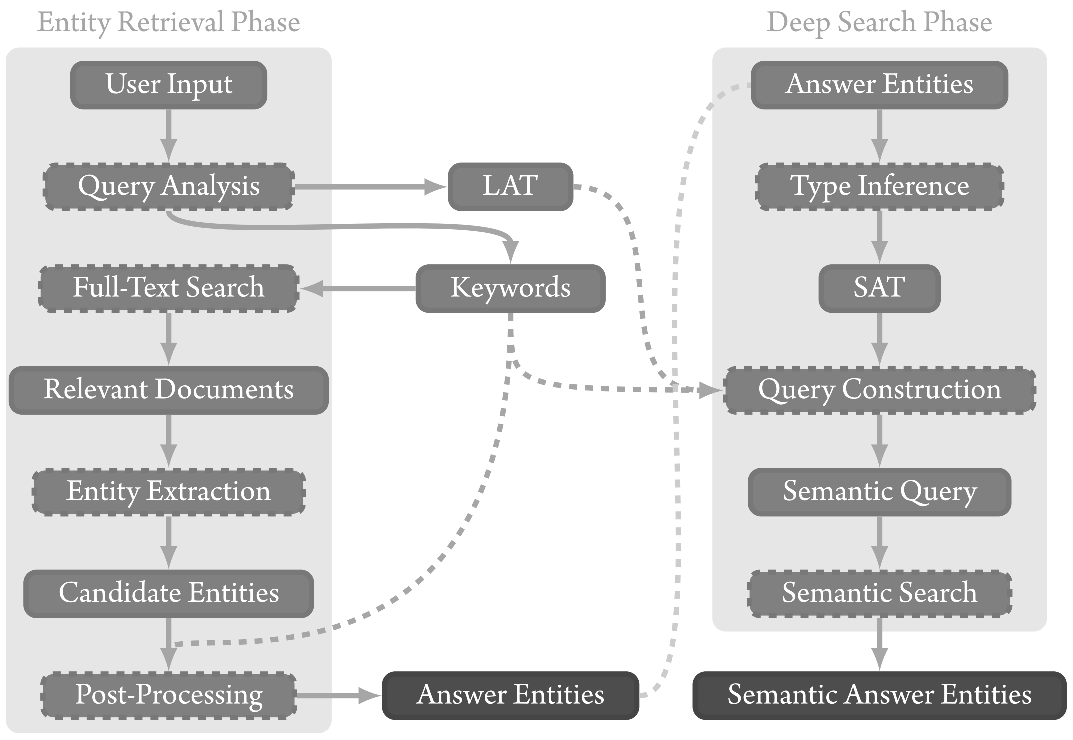


Figure 4.1.1: Overview of the two-phase approach: entity retrieval on the left and the deep search on the right, with shared intermediate results in the center

4.2 User Experience

The goal of semantic search is to answer queries with facts – in our case these facts are limited to named entities. The user interfaces with the search engine via a text input field or using voice. In both cases, the input consists of either keywords or whole phrases. The result is a list of entities; see figure 4.2.1 for an example user interface scheme outline. A more sophisticated result view could provide additional information like the type of the answer entity, reference links and pictures, which helps the user to uniquely identify the entity.

4.2.1 User Input

Our goal for the user experience regarding the type of input is to provide a convenient interface adherent to the following principles:

1. Allow barrier-free user input
2. Allow incomplete and unstructured user queries

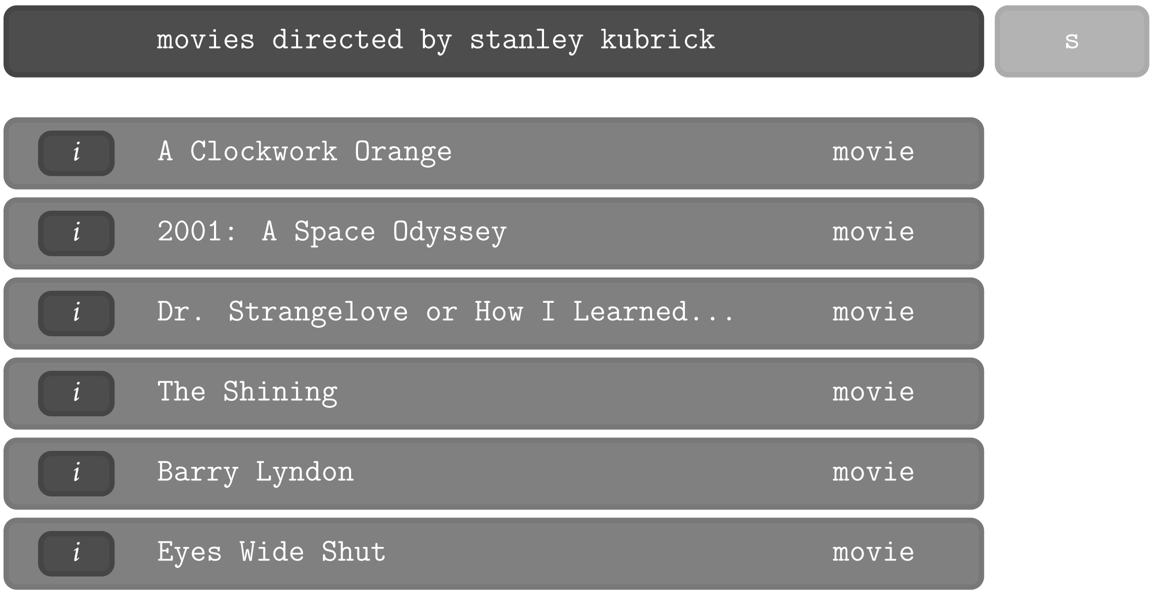


Figure 4.2.1: User interface example: query input field and search button at the top and a result list containing entity name, its type and a link for additional information

3. Do not force user adaptation

By principle 1, we disregard all user input schemes which rely on interactive query refinement. A user query should be submitted by typing into a text field without interference from the system.

Principle 2 assures that any form of user input is allowed – whole questions, short phrases and just keywords should be all supported in the best possible way.

The last principle assures that no hard restrictions of the query language are enforced and no additional user input is required besides the naturally composed user query. We cannot support queries of arbitrary complexity, as this requires *strong artificial intelligence*, but we avoid restrictions of the query language by design and allow for easy extendability.

4.2.2 Result Output

The result output should reference the answer entities in a way to allow for their unique identification. The deep search phase operates on the knowledge base, which is suitable for unique entity identification – contrary to the entity retrieval phase, which is restricted to quantitative methods on the lexical level.

Therefore, for the first phase, we introduce the loose concept of *context-local identification*. The goal here is to produce intermediate result strings, which provide a reasonably clear reference to the answer entities in the given query context. Practically, this means that the string

"Jupiter"

is an acceptable answer for the queries

"Roman god of sky" and *"largest planet"*

without additional references.

4.3 Query Analysis

Given the user query, the goal of the query analysis is to enrich and clean up the query. The output of this phase are the *keywords* and the *lexical answer type (LAT)* of the query. A further useful improvement would be to produce synonyms of the keywords and consider them during the full-text search query construction.

First, we process the user query using *part-of-speech (POS)* tagging. The part-of-speech annotations are required to locate the *nouns* and *important verbs* in the query, which is used to identify the *lexical answer type* and the *relation* between subject and object. For an explanation of part-of-speech tagging, please revisit chapter 3.2.1.

4.3.1 Keywords

Using part-of-speech tags, we can identify the important words – or *keywords* – in the user query. We define keywords as all the nouns, adjectives, adverbs and non-trivial verbs of the query. Trivial verbs like *"is"* and other word types are ignored. The keyword collection is the basis for further query analysis, for the full-text search query and eventually for the entity filtering. See table 4.3.1 for an example of part-of-speech-based keyword tagging.

word	*"books"*	*"written"*	*"by"*	*"lewis"*	*"carroll"*
POS	noun	verb	preposition	proper noun	proper noun
keyword	yes	yes	no	yes	yes

Table 4.3.1: POS and keyword tags for the query *"books written by lewis carroll"*

Removing redundant words from the query is a way of reducing runtime complexity during the filtering phase. It is also useful for the construction of the full-text search query, since the number of keywords has an impact on the performance and result precision.

4.3.2 Lexical Answer Type

The *lexical answer type (LAT)* is a single noun or noun phrase, which provides a hint for the *semantic answer type (SAT)* of the query. We use simple heuristics to identify the LAT within the query based on the provided part-of-speech tags. For example, the LAT for the query *"nicole kidman's siblings"* is *"siblings"* and it is *"targets"* for the query *"first targets of the atomic bomb"* – depending on the ontology used, the SATs could be *person* and *location* respectively.

We apply a set of rules to identify the LAT robustly in a variation of queries. We have identified two ways of posing keyword and phrased queries:

1. [who/what/where/... <verb>] <LAT> <relation> <subject>
   Example: *"[which are the] books written by lewis carroll"*
2. [who/what/where/... <verb>] <subject>'s <LAT>
   Example: *"lewis carroll's books"*

Both cases can be handled by examining the first noun. If the noun is in possessive form, then case 2 holds and the LAT is expected to be the noun which follows next. Otherwise, the first noun is the LAT. The example in table 4.3.2 demonstrates the rule-based LAT detection for a query of case 1 and table 4.3.3 an example for case 2.

word	*"astronauts"*	*"who"*	*"landed"*	*"on"*	*"the"*	*"moon"*
POS	noun	pronoun	verb	preposition	determiner	(proper) noun
keyword	LAT	no	yes	no	no	yes

Table 4.3.2: LAT detection for a type 1 query

word	*"john"*	*"lennon's"*	*"parents"*
POS	proper noun	possessive proper noun	noun
keyword	yes	yes	LAT

Table 4.3.3: LAT detection for a type 2 query

4.4 Full-Text Search

In this phase, we use the keywords from the first phase to construct a full-text *search query*. Using the constructed search query, we use a full-text search engine to obtain the ranked *relevant documents.*

We propose the inverted index for efficient query processing and a ranking based on BM25, or similar; for details revisit chapter 3. The implementation of a full-text search engine is out of scope of this thesis, but it should be noted that high-precision results in this phase are critical for the quality of the final results of our approach.

4.4.1 Query Construction

Full-text search engines perform best when only the key terms are provided in the query. However, redundant keywords generally do not harm the precision, if conditional keyword elimination is supported – that means that relevant documents do not need to contain all keywords necessarily. Additionally, popular terms do not have a negative effect on result quality due to ranking relative to their document frequency.

We construct the full-text search query by including all identified keywords, as described in 4.3.1. If the search engine supports conditionals, we mark the lexical answer type to be unconditional, while all other keywords can be optimally removed, if the resulting documents show more relevance without them. If the LAT is provided in plural form, we deduce its singular version and include it as an alternative to the original term.

For example, the user query

"astronauts who landed on the moon"

would be translated to the full-text search query

"+(astronauts | astronaut) landed moon"

where the *"+"* denotes an *unconditional keyword*, which means that it must be included in all returned documents and *"|"* is an *or-conjunction* between the two spelling versions.

4.5 Entity Extraction

In the previous phase, we showed how to retrieve the syntactically relevant documents. The next step is to extract the candidate entities from the documents using a named entity extractor and to count the frequency of occurrences of each entity in a given document.

Named entity extraction on large documents is time intensive. To reduce the load, we reduce the amount of text of each document by preprocessing. For this, we strip all meta information, which is contained in a HTML, including all HTML tags and their properties. The goal of this preprocessing to reduce the text size without stripping away important context and to do it in the most efficient way.

Regular expressions can be used to parse HTML files and locate tags and properties. However, this can be error prone for malformed HTML files or it amounts to a high complexity in pattern matching to catch all corner cases, which is inefficient. We decided to go with a much simpler approach by scanning for HTML tags beginning with unescaped *""<""* and stripping the tag out by finding the next *"">""*. On uncertain conditions, we revert the text removal, as to avoid destroying potential context for the natural language processing tools. Also, we limit the parsing to the document's *body* element.

4.5.1 Text Snippets

A single document may cover multiple topics. When processing such a document, it is necessary to identify relevant text passages to avoid the extraction of unrelated entities. This can be handled by assigning a window around relevant keywords and only consider the text within a set proximity to those. However, this can have a negative effect for large, in-depth documents covering a specific topic; for example, like they are often found on Wikipedia. Choosing the appropriate window size for each case is non-trivial and sub-optimal selection could degrade the final result quality.

An alternative approach is to include the proximity of the extracted entity to the next keyword into the ranking of the entity. It enables a more fine-grained control over the entity scores, but requires further tweaking to handle the different types of document contents.

Both approaches to avoid the promotion of irrelevant entities require a deep analysis to find a parameter set that behaves robustly across the majority of documents. To our knowledge, no suitable ground truth exists to support such parameter optimization and the construction of such is out of scope of this thesis. Therefore, we have decided to process whole documents with equally ranked entities across a document.

To counteract the inevitable influx or irrelevant entities, we use the short text snippets, as they are provided by full-text search engines. The text snippets are processed as additional documents inheriting the source document rank. This is a compromise between both approaches, but allows for more direct control and does not require complex parameter tweaking. We acknowledge, that the existence of text snippets requires some form of contextual preprocessing, which we would need to be reproduced when implementing the full-text search instead of using a third-party provider.

4.6 Entity Ranking

The goal is to find the most relevant result entities and to achieve this, we need to rank the candidate entities. The final score for each entity *e* from the given entity set *E* is a weighted combination of the subscores and defined as:

$$score(e) = \sum_{s \in Subscores} \frac{w_s \cdot s(e)}{s_{max}} \quad \text{where} \quad s_{max} = \max_{n \in E} s(n)$$

The *Subscores* is a set of scoring function used to control the contribution level of the different occurrence types:

$$Subscores = \{s_C, s_H, s_{CD}, s_{HD}\}$$

More specifically, s_C rates the quantity and quality of the occurrences of an entity within the documents' contents and s_H gives special significance to entities found in *text snippets*. Additionally, we use the s_{CD} and s_{HD} subscores to consider the number of documents and text snippets the entity occurs in for increased confidence of low-frequency terms.

The subscore functions are trivially defined as the weighted linear combination of all occurrences:

$$\begin{aligned} s(e) &= |Occurs(e)| && \text{for } s \in \{s_{CD}, s_{HD}\} \\ s(e) &= \sum_{\langle freq, rank \rangle \in Occurs(e)} \frac{w_{rank} \cdot freq}{\log(cf(e) + cf_{base})} && \text{for } s \in \{s_C, s_H\} \end{aligned}$$

The *Occurs* function returns the set of *frequency-rank tuples* for the given entity's occurrences within the documents. We use the w_{rank} weighting constants to give more significance to top-ranked matches. The corpus entity frequency *cf* is required to yield the proper relevance score for an entity given its general popularity. The cf_{base} constant is in the range $[1, \infty)$ and can be used to weight the dependability of the acquired corpus frequencies. In general, the weighting constants w_s for each subscore and w_{rank} for each individual document rank give us fine-grained control over the scoring results and enables easy experimentation.

4.6.1 Document Rank

The full-text search returns documents ranked by their relevance to the query. The relevance of the source document is a strong indicator for potential candidate entities extracted from them.

The resolution of the document relevance score determines the quality of the derived scores. A linear, discrete ranking scheme does not provide sufficient information to assess the actual relative relevance between two ranks. However, if fine-grained scores are not available, we apply dynamic document rank weighting to counteract the adverse effects.

The linear dampening of discrete ranks can be achieved using the weighting formula:

$$w_{rank} = 1 - \frac{rank}{1 + \lambda \cdot rank_{max}}$$

For example, the relevance factor for a ranking scheme in range $[1, 10]$ between the top rank 1 and the bottom rank 10 is 10. In other words, an entity occurrence in the top ranked document is scored equally to ten entity occurrences in the least ranked document. With a dampening parameter of $\lambda = 1$, the factor decreases to less than 2, which allows for a convenient smoothing of overly popular document sources or inaccurate discrete ranking steps, as in our case.

4.6.2 Document & Snippet Frequency

The document frequency of a given entity is the number of documents the entity occurs in. The occurrence of a candidate entity in multiple sources increases the confidence in the relevance of that entity. In this case, we disregard the actual frequency of the entity within the documents and the document ranks.

Additional evidence for the relevance of an entity is provided, if the entity occurs in multiple snippets. We handle snippets like regular, albeit very short, documents, but provide separate weights from the full documents. This allows us to increase emphasis on snippet frequency, which is generally a good indicator for the relevance of an entity.

4.6.3 Corpus Entity Frequency

Some entities occur more often in texts than others. For example, many documents contain *USA* or other popular locations even when the topic is about some event or person. We need a way to suppress high frequency entities like that and give more relevance to less frequent ones without skewing the results in favour of very rare entities.

TF-IDF and *BM25* scores – see 3.1.2 – put term frequencies relative to the popularity of the term using the total number of documents the term occurs in. Since we rate entities, not documents, we need a way of putting entity frequencies relative to their overall frequency within the corpus.

The English Wikipedia offers a great open text corpus and is well suited for us due to the high information density of its content. We processed all English Wikipedia pages to extract the named entities and have used the resulting frequencies to assess the general popularity of each extracted entity. The resulting corpus frequency *cf* for a given entity *e* are then used as the factor for the subscoring functions:

$$\frac{1}{\log\left(cf(e) + cf_{base}\right)}$$

Although the precision of named entity extraction is high, it is not perfect. That leaves us with some imprecision in the data obtained – some entities are underrepresented relative to the most popular ones. To avoid excessive boosts for such unique or difficult to extract entities, we add the base frequency offset cf_{base} to each frequency count and use a logarithmic scale to flatten the relative differences between the entity scores.

4.7 Answer Entities Selection

Search engines return answers to user queries, in our case the answers are named entities. So far, we have ranked the entities according to their relevance and filtered out improbable entities or entities of wrong types.

The next step is to find the set of answer entities among the top ranked candidates. We propose the *Moving Average Pivot (MAP)* selection method – a statistical approach solely based on the entity scores.

4.7.1 Moving Average Pivot Selection

The selection set E_s is a subset of the set of candidates E_c and contains only entities, whose scores satisfy the minimum score threshold δ. An entity is denoted as $e = \langle e_n, e_s \rangle$ where e_n is the name of the entity and e_s its assigned score.

$$E_s = \{e \in E_c \mid e_s \geq \delta\} \quad \text{where} \quad \delta = S_{avg} + (2\gamma - 1)(S_{max} - S_{avg})$$

The parameter $\gamma \in \mathbb{R}$ is in the range $[0, 1]$ and used to balance between *recall* and *precision*. We have found that higher values in the range $[0.6, 0.7]$ for the first phase and lower values in the range [0.4, 0.5] for the second phase performed well in our experiments, for more details see chapter 6.

The extrema are defined in a natural way:

$$S_{min} = \min\{e_s \mid e \in E_c\} \quad \text{and} \quad S_{max} = \max\{e_s \mid e \in E_c\}$$

The average score is used as a pivot for the selection. We propose using the *exponential moving average (EMA)* instead of the regular average to increase the influence of the top ranked candidates' scores over the less ranked ones'. Additionally, this has the positive effect of dampening top ranked outliers relative to the average score; chapter 6 discusses the reason for such score spikes in more detail. The recursive definition for the EMA score is:

$$S_{avg_r} = \alpha \cdot e_{r-1_s} + (1-\alpha) \cdot S_{avg_{r-1}} \quad \text{with} \quad S_1 = e_1$$

where S_{avg_r} denotes the EMA score up to rank r and e_r the entity tuple at rank r. The α coefficient is the smoothing factor in the range $(0, 1]$. To perform well across a high range of entity set sizes, we determine the factor dynamically depending on the number of entities considered:

$$\alpha = \frac{2}{|E_c| + 1}$$

The goal of the selection is to detect score difference spikes in the higher ranks without isolating outliers. Any numerical method that is capable of detecting value plateaus could be suited for this task. The moving average pivot selection is simple, depends only on entity scores, requires *no additional memory* and is computable in *linear time*.

4.8 Semantic Answer Type

In 4.3 we discussed the importance and extraction method for the lexical answer type (LAT). To provide the correct result entities however, we need to know the actual answer type supported by the ontology. There are multiple ways to infer the answer type, which we will briefly discuss.

4.8.1 Qualitative SAT Inference

A *qualitative analysis* of the LAT can yield a set of candidate answer types by consulting the ontology for abstractions of the LAT. One way is to trace the *is-a* relation paths until we reach a type of a suitable abstraction level. Figure 4.8.1 shows an example for a hypothetical type hierarchy. Based on this example, we would choose the SAT to be *person* for the LATs *"astronaut"* and *"sibling"*. Alternatively, *astronaut* could be used directly as the SAT if supported by the ontology.

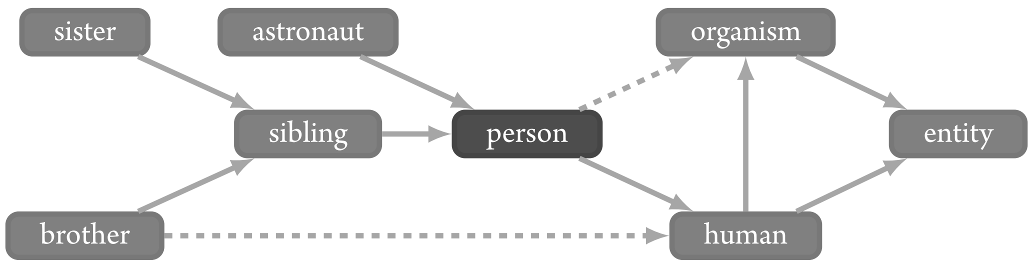


Figure 4.8.1: Hypothetical ontology's *is-a* relations

The qualitative approach can be effective, but is strongly coupled with the ontology. This makes it less flexible and requires manual maintenance when the ontology database is being switched or extended. What is more critical, the qualitative SAT inference fails, if some critical relations are missing or if the LAT is highly ambiguous. For example, the query *"members of U2"*, depends on the *is-a* relation between *member* and *person*. An ontology containing the relation *member* $\rightarrow$ *person* will also contain *member* $\rightarrow$ *state* and others, which yields a huge set of SAT candidates.

We think that this approach does have potential as a refinement of a type inference result, but is impractical when used as the sole method.

4.8.2 Quantitative SAT Inference

The *quantitative approach* infers the type through an analysis of the top scoring candidate entities. For each entity, we identify its types in the ontology. The result is a set of candidate SATs, which are ranked based on the entity score and the abstraction level of the type.

The quantitative approach should be more robust, perform well for abstract and ambiguous types and be more independent from the quality of the underlying ontology. However, it does depend on the quality and more so on the quantity of the retrieved entities, which directly translates into a dependence on the precision of the full-text search and named entity extraction.

4.9 Entity Filtering & Clustering

Entity filtering takes place as a post-processing step after entity extraction. The filtering is a multi-pass procedure with optional clustering and applies the following techniques.

4.9.1 Word Constraints

Entity name sizes must be within a given range and only contain alphanumeric characters, hyphens and apostrophes.

4.9.2 Entity Clustering

The same entity can be referenced in a variety of ways. The *United States of America* may be abbreviated with *USA*, *U.S.* or just *the States* or *America*. Our criteria for entity unification considers the prefix-word distance between each word of two given entities. When the number of similar words exceeds a set threshold, we unify both entity names by choosing a representative name and accumulate the entity scores.

Additionally, we detect simple abbreviations like in our example *USA* and extend it. For efficiency reasons, this feature is reduced to short initialisms – so we would detect *USA*[1] and *NSA*[2], but not *Radar*[3] and *W3C*[4].

4.9.3 Query Similarity

User queries can, and often do, contain named entities. For example, the query *"nicole kidman's" "siblings"* contains the person entity *Nicole Kidman*. Since the full-text search returns relevant documents to the given strings, the documents will frequently contain some variation of the entity's name. Since this entity is part of the question, we need to filter it out of the candidate entities.

For this, we use the entity clustering procedure and remove candidate entities that match a query entity too closely. Our approach is based on word *edit-distances*, which reflect the rate of similarity between two strings, [3] offers an interesting comparison for a variety of algorithms, which are applicable in the case of entity disambiguation.

[1]USA is an initialism for **U**nited **S**tates of **A**merica.

[2]NSA is an initialism for **N**ational **S**ecurity **A**gency.

[3]Radar is an acronym for **Ra**dio **D**etection **a**nd **R**anging.

[4]W3C is an initialism shortcut for **W**orld **W**ide **W**eb **C**onsortium.

4.10 Semantic Query Construction

For the deep search phase, we construct a semantic query based on the extracted keywords and the inferred SAT. Our construction method is best suited as input for a hybrid search engine, such as Broccoli [1]. The advantage of the hybrid search is that we can use the SAT to declare the result type explicitly, while still being able to add the keywords for improved relevancy.

We use a basic formula to construct the query – the SAT is used to provide the class of the answer entities and the keywords are added to a *occurs-with* relation for syntactic full-text matching. For example, the user query

"astronauts who walked on the moon"

could be translated to

"$1 is-a Astronaut; $1 occurs-with moon"

where *$1* refers to the answer entities, the *is-a* relation denotes the type of the requested entities and the *occurs-with* restricts the results to entities, whose reference pages contain the string *"moon"*. Additionally, the verb *walked* can be added in its present form as *"walk*"*. For more details regarding the supported syntax, please refer to [1].

For all the things we have to learn before we can do them,
we learn by doing them.

Aristotle

5 Implementation

The previous chapter described the two-phase approach for semantic search with keywords in an implementation-agnostic way. In this chapter, we depict the details of Pythia – a *lightweight* implementation of the proposed method. Our goal with Pythia is first to test the effectiveness of the two-phase approach and its intermediate steps, and second to create a platform for experimentation on semantic search in general.

After a brief system overview in 5.1, the following chapters describe the realization of each module and we also note differences to the proposed method, which materialized during the development of the platform.

5.1 System Architecture

Pythia is composed of a server application – the *backend* – and a web client – the *frontend*.

5.1.1 Backend

The backend serves all the required markup files via HTTP, corresponding to a regular web server. Additionally, it offers a simple query interface with the results returned in JSON[1] format. When not noted otherwise, JSON is the exchange format used for the server-client communication.

The first iterations of the server implemented the full-text search, the entity extraction, filtering and ranking. To increase the dynamics of the platform, we moved all the critical processing steps out of the backend into the web client. This allows us to quickly evaluate new ranking and filtering concepts and gives us immediate results for an adjusted scoring scheme without recompilation and reinitialization of the server application.

In the first step, the server accepts a query request from the client, analyses the query and uses full-text search to find the most relevant documents for the query. The results contain the document URLs, their titles and some meta data. We retrieve all documents for further processing and assign a score to each based on the result rank.

Next, we extract the named entities from the retrieved documents. The results contain the entity names, their frequencies in the respective documents and additionally the overall popularity of the entities, if provided by the knowledge base. The results are asynchronously sent back to the requesting client.

The backend does also handle semantic entity type queries and ground truth requests, the former are utilized during the inference of the SAT, the latter is required to provide the evaluation results, which are also displayed in the web client.

The server is written in C++11 using the Standard Template Library and following third-party libraries:

- POCO C++[2] for HTTP(S) handling and JSON decoding
- SENNA[3] for natural language processing
- gflags[4] for command line flags processing
- glog[5] for message logging
- gtest[6] for unit testing

[1]http://json.org/
[2]http://pocoproject.org
[3]http://ml.nec-labs.com/senna
[4]https://code.google.com/p/gflags
[5]https://code.google.com/p/google-glog
[6]https://code.google.com/p/googletest

- gperftools[7] for performance profiling
- Flow[8] for serialization, string operations and JSON encoding

The modified cpplint[9] was used for code style checking. The server runs on Linux systems supporting the GNU toolchain.

5.1.2 Frontend

The web client interfaces with the server via its HTTP interface. As noted in 5.1.1, the frontend filters, clusters and ranks the entities returned by the server. Finally, it visualizes the ad-hoc query results and the evaluation statistics for the selected ground truth.

Even without further optimizations, the performance of current JavaScript implementations was sufficient for our cause. Should it be required to speed up the processing – entity clustering is a good candidate for this – then there are options to achieve near-native performance when using a subset of JavaScript in combination with asm.js[10]. For best results, we can develop the performance-critical procedures in C++ and compile them into efficient JavaScript code using Emscripten[11] – a LLVM[12] to JavaScript compiler.

5.2 User Interface

The user interface (UI) is designed using HTML5 and CSS3 based on Bootstrap[13]. The client logic and communication with the server is realized in JavaScript with the help of jQuery[14] for convenient asynchronous message passing and more.

Pythia's UI is composed of a *text field* for the query input and collapsible result views for:

1. Performance overview
2. Query analysis including keyword and LAT tags
3. Semantic answer type (SAT)
4. Semantic query for Broccoli input

[7]http://code.google.com/p/gperftools
[8]https://github.com/eamsen/flow
[9]https://github.com/eamsen/cpplint
[10]http://asmjs.org
[11]https://github.com/kripken/emscripten
[12]http://llvm.org
[13]http://getbootstrap.com
[14]http://jquery.com

5. Result entity selection of the first phase
6. Result entity selection of the second phase
7. Complete candidate entity table
8. Retrieved documents list
9. Evaluation result table
10. Scoring function control
11. Logs for further analysis

The input field accepts free form text for the query. Result areas 1 through 8 show the details for the last manual query. Figure 5.2.3 shows a possible UI configuration with expanded views for 2, 3, 5 and 6 and figure 5.2.4 shows all candidate entity stats for the same query.

The performance view – see figure 5.2.1 for an example – gives us a quick overview of the execution times during the particular processing stages. It should be noted that the performance becomes largely irrelevant for our experiments when result caching comes into effect.

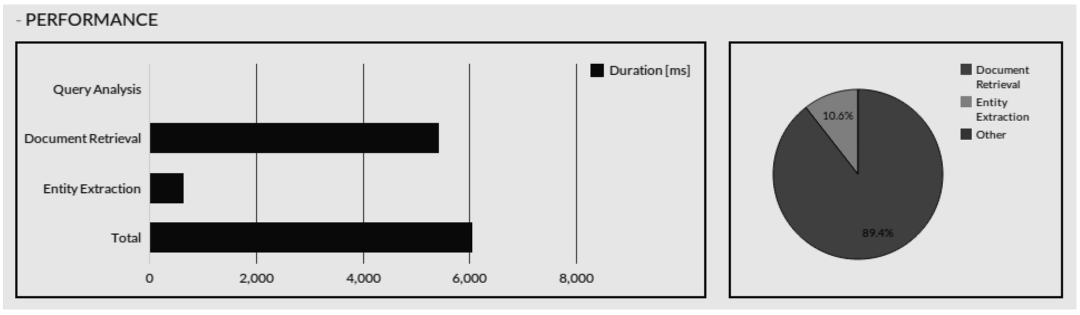
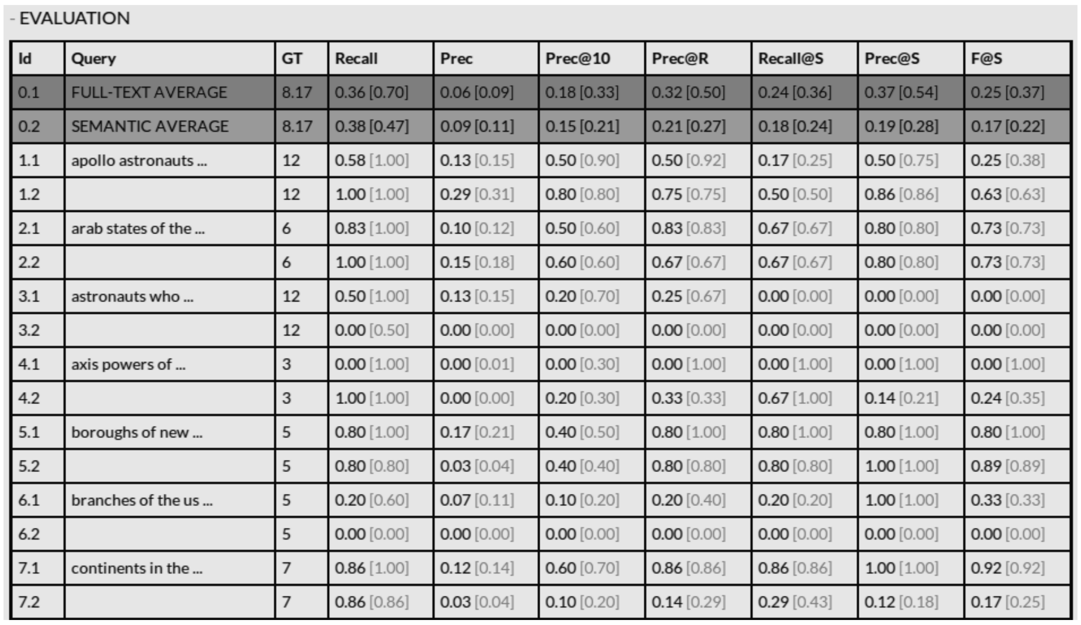


Figure 5.2.1: Pythia performance view for an example uncached query run

The evaluation result table does not depend on the manual queries, instead it processes all the queries of a selected ground truth set in the background and visualizes the average and per-query metrics, see figure 5.2.2.

The scoring function is used for manual and automatic query result ranking. On scoring parameter adjustment the last query results are recalculated and the automatic evaluation restarts in the background to update the metrics on the evaluation result table.


- EVALUATION

Id	Query	GT	Recall	Prec	Prec@10	Prec@R	Recall@S	Prec@S	F@S
0.1	FULL-TEXT AVERAGE	8.17	0.36 [0.70]	0.06 [0.09]	0.18 [0.33]	0.32 [0.50]	0.24 [0.36]	0.37 [0.54]	0.25 [0.37]
0.2	SEMANTIC AVERAGE	8.17	0.38 [0.47]	0.09 [0.11]	0.15 [0.21]	0.21 [0.27]	0.18 [0.24]	0.19 [0.28]	0.17 [0.22]
1.1	apollo astronauts ...	12	0.58 [1.00]	0.13 [0.15]	0.50 [0.90]	0.50 [0.92]	0.17 [0.25]	0.50 [0.75]	0.25 [0.38]
1.2		12	1.00 [1.00]	0.29 [0.31]	0.80 [0.80]	0.75 [0.75]	0.50 [0.50]	0.86 [0.86]	0.63 [0.63]
2.1	arab states of the ...	6	0.83 [1.00]	0.10 [0.12]	0.50 [0.60]	0.83 [0.83]	0.67 [0.67]	0.80 [0.80]	0.73 [0.73]
2.2		6	1.00 [1.00]	0.15 [0.18]	0.60 [0.60]	0.67 [0.67]	0.67 [0.67]	0.80 [0.80]	0.73 [0.73]
3.1	astronauts who ...	12	0.50 [1.00]	0.13 [0.15]	0.20 [0.70]	0.25 [0.67]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]
3.2		12	0.00 [0.50]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]
4.1	axis powers of ...	3	0.00 [1.00]	0.00 [0.01]	0.00 [0.30]	0.00 [1.00]	0.00 [1.00]	0.00 [1.00]	0.00 [1.00]
4.2		3	1.00 [1.00]	0.00 [0.00]	0.20 [0.30]	0.33 [0.33]	0.67 [1.00]	0.14 [0.21]	0.24 [0.35]
5.1	boroughs of new ...	5	0.80 [1.00]	0.17 [0.21]	0.40 [0.50]	0.80 [1.00]	0.80 [1.00]	0.80 [1.00]	0.80 [1.00]
5.2		5	0.80 [0.80]	0.03 [0.04]	0.40 [0.40]	0.80 [0.80]	0.80 [0.80]	1.00 [1.00]	0.89 [0.89]
6.1	branches of the us ...	5	0.20 [0.60]	0.07 [0.11]	0.10 [0.20]	0.20 [0.40]	0.20 [0.20]	1.00 [1.00]	0.33 [0.33]
6.2		5	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]
7.1	continents in the ...	7	0.86 [1.00]	0.12 [0.14]	0.60 [0.70]	0.86 [0.86]	0.86 [0.86]	1.00 [1.00]	0.92 [0.92]
7.2		7	0.86 [0.86]	0.03 [0.04]	0.10 [0.20]	0.14 [0.29]	0.29 [0.43]	0.12 [0.18]	0.17 [0.25]

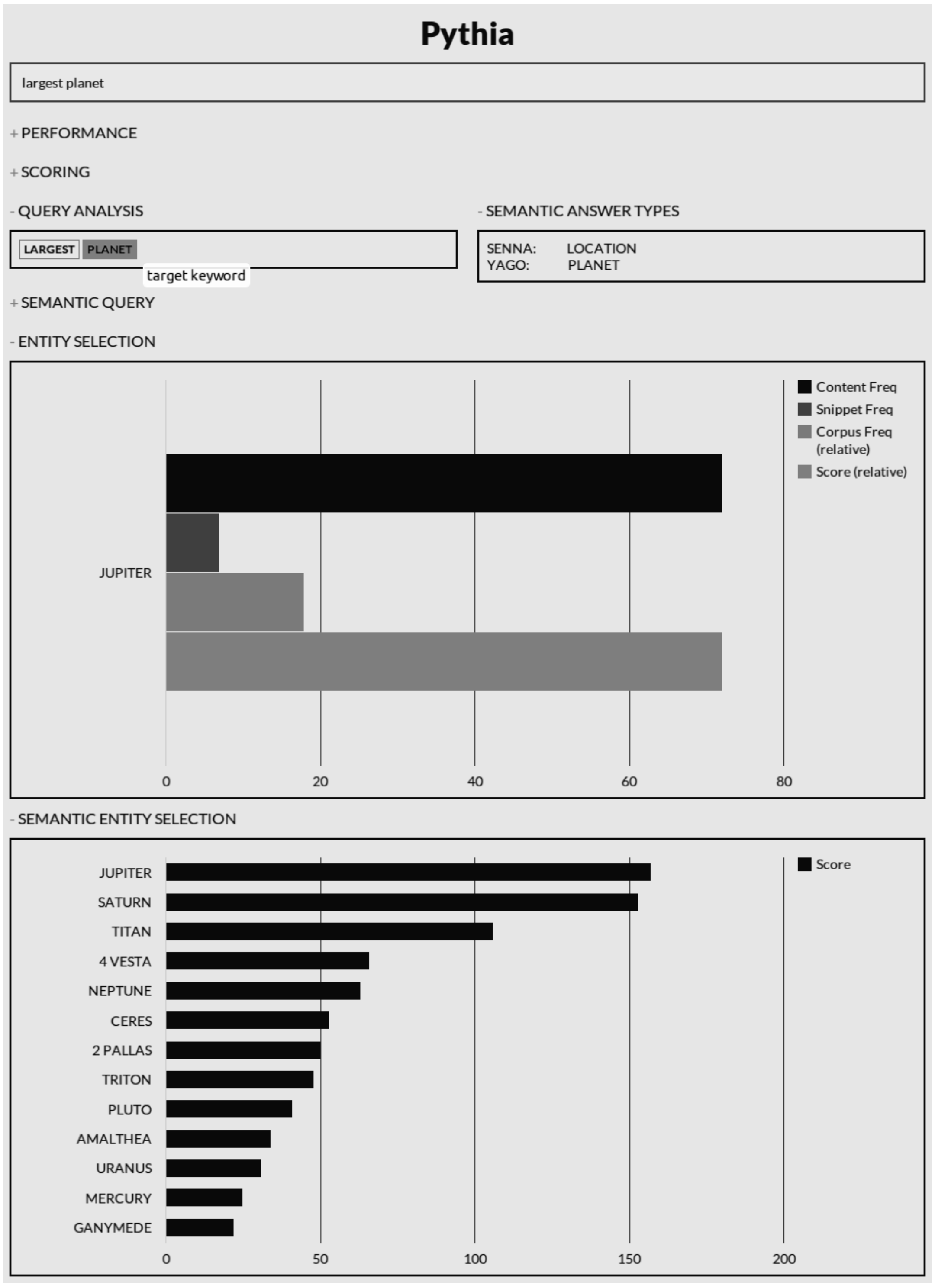
Figure 5.2.2: Pythia evaluation view excerpt: averaged results on top; two rows (one for each phase) per query; *GT* shows the number of expected answer entities; approximate match results are in brackets

5.3 Full-Text Search

A complete index of the Web and an accommodating search engine requires immense resources and development time. Since it is not our goal to prove the effectiveness of web search, we use the Google Custom Search API for *ad-hoc* search results.

Google's search API provides high-quality results, but it also comes with some limitations – it returns only the top 10 results, has a limited daily query quota and does not provide the document scores.

The API does not reveal the actual scores in the results, which would be valuable for the ranking process. This is an issue that we address by a more flexible scoring scheme, which is explained in 4.6. The daily quota limit is not a big concern for us, since we cache all results persistently. However, the restrictive result set, which contains only the top 10 documents, can have a negative effect on the recall metric, since our quantitative approach gains stability with larger sets of documents.


Pythia

largest planet

+ PERFORMANCE

+ SCORING

- QUERY ANALYSIS

LARGEST PLANET

target keyword

- SEMANTIC ANSWER TYPES

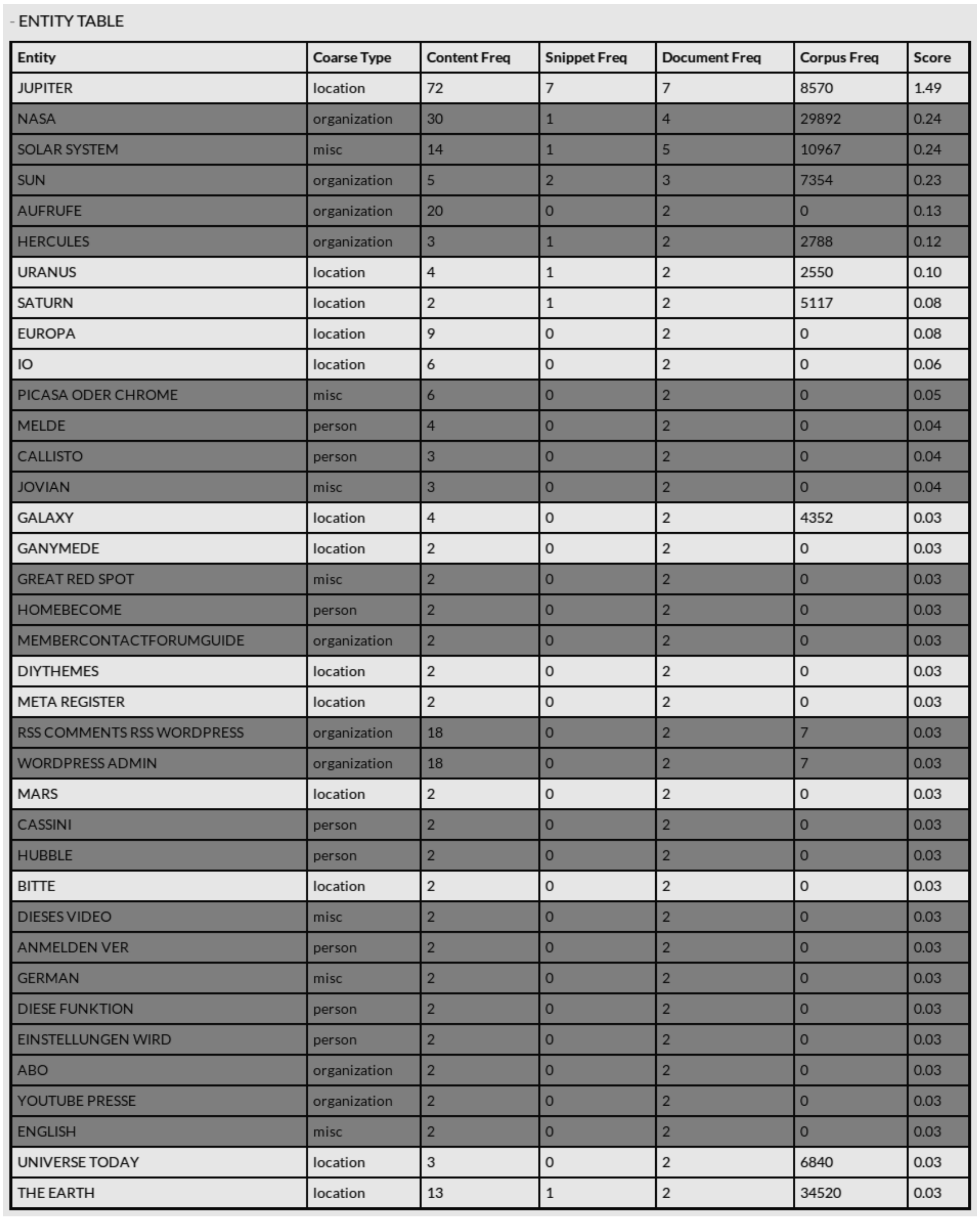
SENNA:	LOCATION
YAGO:	PLANET

+ SEMANTIC QUERY

- ENTITY SELECTION

- SEMANTIC ENTITY SELECTION

Figure 5.2.3: Pythia UI excerpt: query input at top; expanded views for query analysis, SAT, entity selection (first phase) and semantic entity selection (second phase); other views are collapsed or out of screen



- ENTITY TABLE

Entity	Coarse Type	Content Freq	Snippet Freq	Document Freq	Corpus Freq	Score
JUPITER	location	72	7	7	8570	1.49
NASA	organization	30	1	4	29892	0.24
SOLAR SYSTEM	misc	14	1	5	10967	0.24
SUN	organization	5	2	3	7354	0.23
AUFRUFE	organization	20	0	2	0	0.13
HERCULES	organization	3	1	2	2788	0.12
URANUS	location	4	1	2	2550	0.10
SATURN	location	2	1	2	5117	0.08
EUROPA	location	9	0	2	0	0.08
IO	location	6	0	2	0	0.06
PICASA ODER CHROME	misc	6	0	2	0	0.05
MELDE	person	4	0	2	0	0.04
CALLISTO	person	3	0	2	0	0.04
JOVIAN	misc	3	0	2	0	0.04
GALAXY	location	4	0	2	4352	0.03
GANYMEDE	location	2	0	2	0	0.03
GREAT RED SPOT	misc	2	0	2	0	0.03
HOMEBECOME	person	2	0	2	0	0.03
MEMBERCONTACTFORUMGUIDE	organization	2	0	2	0	0.03
DIYTHEMES	location	2	0	2	0	0.03
META REGISTER	location	2	0	2	0	0.03
RSS COMMENTS RSS WORDPRESS	organization	18	0	2	7	0.03
WORDPRESS ADMIN	organization	18	0	2	7	0.03
MARS	location	2	0	2	0	0.03
CASSINI	person	2	0	2	0	0.03
HUBBLE	person	2	0	2	0	0.03
BITTE	location	2	0	2	0	0.03
DIESES VIDEO	misc	2	0	2	0	0.03
ANMELDEN VER	person	2	0	2	0	0.03
GERMAN	misc	2	0	2	0	0.03
DIESE FUNKTION	person	2	0	2	0	0.03
EINSTELLUNGEN WIRD	person	2	0	2	0	0.03
ABO	organization	2	0	2	0	0.03
YOUTUBE PRESSE	organization	2	0	2	0	0.03
ENGLISH	misc	2	0	2	0	0.03
UNIVERSE TODAY	location	3	0	2	6840	0.03
THE EARTH	location	13	1	2	34520	0.03

Figure 5.2.4: Pythia candidate entities view for the query *"largest planet"*: coarse type is detected by SENNA during entity extraction; darkened rows are filtered out entities

5.4 Entity Extraction

Before entity extraction, we remove all HTML tags from the documents using a custom implementation. Then we extract the named entities using SENNA – a natural language processing toolset based on artificial neural networks. SENNA provides a good balance between precision and performance [4], which makes it better suited for Pythia than more precise tools like the Stanford Named Entity Recognizer.

The result of the entity extraction are tuples containing the entity name and its type as recognized by SENNA. The type is important during the filtering phase; we refer to this type as the *coarse type*, since it is limited to the broad categories *organization*, *person*, *location* and *misc* for the rest.

5.5 Entity Filtering

Entity filtering takes place on both, the backend and the frontend. The backend filtering should reduce the load for the frontend, while the filtering in the frontend has the goal to refine and improve the result quality.

5.5.1 Soft Filtering

The named entity extraction does produce some false positives, which need to be identified. To reduce the number of entities before proceeding with the more complex filtering phase, we apply a *soft filtering* technique on the backend. The soft filtering discards entities, which do not pass the required word constraints from 4.9.1 and also removes entities, which have less than two supporting documents. On average, the soft filtering discards 50% of the extracted entities, without negative effects on the recall metric.

5.5.2 Multi-Pass Filtering

The soft filtering of the backend does improve the signal-to-noise ratio, but passes through a high percentage of non-answer candidates. The frontend applies a multi-pass filtering procedure to remove such entities using a variety of techniques.

Pythia offers course type filtering, which is based on SENNA's type classification, query similarity filtering, ontology-based type filtering and a lexical clustering method. The implementation applies the filtering at different stages – before the ranking and then again during post-processing. Such a multi-pass approach makes it possible for the filtering to use ranked type information to deduce the probable SAT, which is only available with ranked results. The pre-filtering is based on lexical methods, which do not require semantic type information.

5.6 Semantic Search

As already mentioned in the previous chapter, our semantic query construction requires a hybrid semantic search interface. In the deep search phase, we query the JSON-based API of Broccoli[15]. The results are provided unfiltered, but with our selection method applied to reduce the amount of answer entities and increase precision.

[15]http://broccoli.cs.uni-freiburg.de

"What is the use of a book", thought Alice,
"without pictures and conversations?".

Lewis Carroll

6 Evaluation

EMPIRICAL EVIDENCE IS THE FOUNDATION FOR PRACTICAL RESEARCH. In this chapter, we evaluate the effectiveness of the two-phase approach using our Pythia search engine.

The base for a statistical evaluation is a ground truth, which we describe in 6.1 and suitable performance measures, as laid down in 6.2. We analyse the performance of the entity retrieval phase in 6.3, review the MAP selection technique in 6.4 and provide the results of the deep search phase in 6.5. The analysis includes all aspects of the process, including entity filtering and ranking schemes.

6.1 GROUND TRUTH

The selection of a ground truth for our task was difficult. We have reviewed some benchmarks for semantic search quality of and have found that no benchmark provides a consistent way to compare systems, which are based on different methodologies. It is especially difficult to compare our two-phased approach with other systems, which are mostly constrained by the specific conference rules.

But finally, we selected the *Yahoo Semantic Search* (SemSearch) queries, see A.1, for the evaluation of the two phases of our approach. SemSearch queries were gathered from actual user input, which makes them practically relevant and they provide a good variety of different query types, which helps evaluating the introduced methods in regards of robustness and generality.

6.2 Performance Measures

The quality of a search engine is measured by recall and precision – for each test query, we analyse the results and compare them with the expected results from the ground truth.

$$recall = \frac{|Results \cap Expected|}{|Expected|} \qquad precision = \frac{|Results \cap Expected|}{|Results|}$$

In our case, the ground truth establishes the notion of relevancy through a set of relevant entities for each test query. More formally, let an entity be a tuple $e = \langle e_n, e_t, e_d, e_h, e_p, e_s, e_r \rangle$, where e_n denotes the name of the entity, e_t its type, e_d the set of frequency-document-rank tuples $\langle f, r \rangle$, e_h the set of frequency-snippet-rank tuples $\langle f, r \rangle$, e_p its general popularity score, e_s is the score assigned by our system and e_r the resulting rank. Furthermore, let E_C be the candidate set of retrieved entities (see chapter 4), E_S the selection set (see 4.7) and E_R the set of relevant entities according to the ground truth. Based on this, we compute recall R and precision P for our results:

$$R = \frac{|E_C \cap E_R|}{|E_R|} \qquad P = \frac{|E_C \cap E_R|}{|E_C|}$$

More general, the recall and precision for a given entity set E are defined as:

$$R(E) = \frac{|E \cap E_R|}{|E_R|} \qquad P(E) = \frac{|E \cap E_R|}{|E|}$$

A large candidate set E_C skews the measures in favor of recall, resulting in low precision. To paint a more realistic image of the performance, we can restrict a given set E to the top k ranks, with $E|_k = \{e \in E \mid e_r \leq k\}$. Similarly, we generalize the recall and precision measures:

$$R(E|_k) = \frac{|E|_k \cap E_R|}{|E_R|} \qquad P(E|_k) = \frac{|E|_k \cap E_R|}{k}$$

Additionally, we define as a special case the dynamic rank value, namely $P@R$ – also known as *R-precision* – where $P@R = P(E_C|_k)$ with $k = |E_R|$.

High precision can be obtained by sacrificing recall, which can be reasonable in specific scenarios, but is generally unpractical. *F-score* is the weighted average of precision and recall and is useful to compare the quality of results by a single value. It is defined as:

$$F_\beta(E) = \frac{(\beta^2 + 1) \cdot P(E) \cdot R(E)}{\beta^2 \cdot P(E) + R(E)}$$

For our measurements we use the *harmonic mean* F_1, or just F for short; and $F_{0.5}$, which weights precision twice as high as recall. The latter is useful to measure the precision of our selection method, without favoring selections, which only return the single best ranked entity.

We note that $P@R = R@R = F@R$ by definition and therefore it is sufficient to document one representative value for these measures.

Here is an overview of the used quality measures abbreviations throughout this chapter:

R is the overall *recall* ratio.

R@S is the *recall* ratio for the retrieved entities within the *selection*.

P@S is the *selection precision*.

P@10 is the *precision* value for the top ten results.

P@R is the *precision* value for the top R results, where R is the number of total relevant entities.

F@S is the *f-measure* for the *selection* results.

6.2.1 Approximate Matching

Named entities may not be uniquely identified by their name and type. The expected answer entities for the query

"Axis powers of World War II"

according to the ground truth are

"Nazi Germany", *"Empire of Japan"* and *"Kingdom of Italy"*

as these are the official handles produces by the YAGO ontology. However, historic texts use the short forms *Germany*, *Japan* and *Italy* within the context, when referring to the aforementioned entities. The result is that Pythia produces the following answer selection

"Germany", *"Japan"* and *"Italy"*

which yields $R@S = P@S = 0$. As we do not identify entities uniquely, an evaluation based on strict matching would skew the results towards short entities or entities with unique and context-independent handles. To counteract this, we provide an additional measure for approximate matchings based on the prefix edit-distance. When reasonable, we provide the approximate matching measures in addition to the strict matching delimited by a dash:

$$measure = strict/approximate$$

For our running example, this gives us $R@S = P@S = 0.0/1.0$ as the results resemble a perfect approximate matching with the ground truth. We note that $strict \leq approximate$ holds for all cases with the strict matchings being a subset of the approximate matchings.

6.2.2 Evaluation Procedure

As previously noted, we strive to evaluation each step in isolation, but doing so is a great challenge. There is a strong interplay between each component – when optimizing filtering, ranking and selection cutoff, it is required to make multiple iterations to let the results propagate and to find the right balance with the adjustable parameters.

For example, we evaluate the filtering results on the previously determined optimal ranking scheme. At the same time, the ranking results do strongly depend on the filtering. The side effect of this iterative evaluation procedure is that some results may not match perfectly when compared between different components, since they were collected at different stages of the process.

6.3 Entity Retrieval

Our bootstrapping approach involves two phases, first we use the user query to conduct entity retrieval on the Web and then we use the intermediate results to construct a formal query input for the Broccoli semantic search engine.

To demonstrate the quality of our approach, we show the quality of each phase separately. This allows for a more consistent evaluation and comparison with similar systems and also makes it simpler to identify issues that need further improvement.

During the first phase, we analyse the query, search the Web for relevant documents, extract entities, filter them and finally rank them. We dedicate an evaluation section to each quantifiable process step.

6.3.1 Entity Extraction & Filtering

To find the candidate entities, we extract all entities of the relevant documents using SENNA's named entity extraction feature.

A critical post-processing step after the extraction is the entity filtering. The goal of the filtering is to discard unreasonable entities without removing viable answer candidates. Let us analyse how effective the extraction works and whether the filtering applied improves the overall results. Table 6.3.1 shows the average results with different filtering combinations

Filter	*R*	*P*	*R@S*	*P@S*	*F@S*
–	0.62/0.88	0.03/0.04	0.17/0.28	0.16/0.30	0.14/0.24
ont	0.54/0.72	0.06/0.07	0.19/0.27	0.27/0.39	0.20/0.28
qsim	0.61/0.87	0.03/0.04	0.36/0.49	0.32/0.49	0.30/0.43
ctype	0.51/0.76	0.05/0.07	0.14/0.25	0.16/0.31	0.13/0.23
ont + qsim	0.53/0.70	0.06/0.07	0.32/0.41	0.43/0.57	0.32/0.41
ont + ctype	0.42/0.57	0.07/0.09	0.15/0.22	0.23/0.36	0.16/0.24
qsim + ctype	0.56/0.78	0.06/0.07	0.35/0.48	0.38/0.59	0.32/0.47
ont + qsim + ctype	0.48/0.63	0.08/0.11	0.30/0.39	0.44/0.61	0.31/0.41

Table 6.3.1: Average results with ontology filter (ont), query similarity filter (qsim) and coarse type filter (ctype)

over the queries. As expected, without filtering a high percentage of false positives remains in the result list and to some degree also in the selection.

The single best-performing filtering technique is to discarding entities, which resembled a too close similarity to some of the keywords in the query. This is a natural consequence of the fact that we extract the entities from documents which are relevant to the keywords on the lexical level. Combined with our selection method, this filter enables a relatively high precision among the top ranked entities, which results in greatly boosted F-scores, while leaving the rest of the stats untouched.

When filtered by the coarse type – as it is provided by the entity extraction tool – the overall precision increases, however, the selection suffers from lower recall at unchanged precision. The effect is caused by our dynamic selection cutoff method – in this case it automatically maintains the precision levels by shrinking the size of the selection. Despite this effect, coarse type filter provides the best overall results when combined with query similarity filtering.

One of our goals was to reduce the dependency on the knowledge base and the ontology filtering results support our motives. Removing entities, which are not known in the ontology constitutes a great hit on recall. It shows, that the ontology does not adequately support all the required entities for the queries.

6.3.2 Ranking Schemes

Ranking is a critical component of the search engine, so we utilized Pythia's customization options for parameter optimization – see table 6.3.2 for a brief description for the adjustable parameters, the detailed explanation is in section 4.6.

Parameter	Description
w_C	document entity frequency weight
w_H	snippet entity frequency weight
w_{CD}	document frequency weight
w_{HD}	snippet frequency weight
λ_C	document rank relevance
λ_H	snippet rank relevance

Table 6.3.2: Ranking parameter descriptions

Even though the selected ground truth does provide a good variety of query types, we try to avoid overfitting by keeping the weighting schemes reasonably simple and comprehensible. To allow for a strategic parameter optimization, we determine four ranking scheme classes with distinct characteristics, see table 6.3.3. Based on that, we create a set of plausible ranking schemes for each class – see table 6.3.4 – and start with their evaluation on our test set.

Scheme Class	Characteristics
rsc	document entity frequency only
rsh	snippet entity frequency only
rsd	document and snippet frequency only
rsx	cross-over scheme for optimization

Table 6.3.3: Ranking scheme classes

Scheme	w_C	λ_C	w_H	λ_H	w_{CD}	w_{HD}
rsc0	1	1	0	–	0	0
rsc1	1	2	0	–	0	0
rsc2	1	-1	0	–	0	0
rsh0	0	–	1	1	0	0
rsh1	0	–	1	2	0	0
rsh2	0	–	1	-1	0	0
rsd0	0	–	0	–	1	0
rsd1	0	–	0	–	0	1
rsd2	0	–	0	–	1	1
rsx0	0.30	2	0.35	2	1.00	1.0
rsx1	0.30	2	0.35	2	0.50	1.0
rsx2	0.30	2	0.35	2	0.25	1.0
rsx3	0.30	2	0.35	2	0.25	0.5
rsx4	0.50	2	0.35	2	0.25	0.5
rsx5	0.30	2	0.20	2	0.25	0.5
rsx6	0.20	2	0.20	2	0.25	0.5
rsx7	0.20	2	0.20	2	0.25	0.7
rsx8	0.25	2	0.30	2	0.25	0.6

Table 6.3.4: Ranking schemes

Table 6.3.5 shows the complete results, we note that additional cross-over ranking schemes *rsx* were added during the evaluation process to find the optimal configuration.

We retrieve over half of the answer entities if strictly matched, and close to 80% for approximate matchings. The overall precision is in the single digit percentage, due to the high quantity of candidate entities retrieved, which is usually in the range of 1000 to 10000 entities before frontend filtered is applied. The overall *R* and *P* ratios should not greatly variate between different rankings schemes, but with coarse type filtering enabled, it is possible for the ranking scheme to promote a different set of accepted SATs.

The *rsc* results form our baseline and is comparable to *TF-IDF* ranking for full-text search. For a basic scheme, it shows promising approximate matching results. The *rsh* scores, which are solely based on Google's result snippets, are competitive with the *rsc* results in the selection, which is a testament to Google's text summarization quality.

More surprisingly are the *rsd* results, which surpass both previous results based on entity frequencies. This could indicate that we are missing a lot of potential with the *rsc* and *rsh* schemes and encourages us to continue our research on finding more effective entity fre-

Ranking	R	P	P@R	R@S	P@S	F@S
rsc0	0.56/0.78	0.06/0.07	0.24/0.45	0.23/0.38	0.23/0.47	0.20/0.36
rsc1	0.55/0.77	0.05/0.06	0.22/0.44	0.22/0.37	0.22/0.48	0.19/0.34
rsc2	0.56/0.78	0.06/0.07	0.23/0.44	0.19/0.33	0.21/0.52	0.17/0.35
rsh0	0.53/0.77	0.06/0.08	0.28/0.43	0.26/0.35	0.27/0.45	0.24/0.35
rsh1	0.55/0.79	0.06/0.08	0.29/0.44	0.27/0.37	0.28/0.46	0.25/0.36
rsh2	0.52/0.76	0.05/0.07	0.23/0.39	0.25/0.38	0.29/0.49	0.23/0.35
rsd0	0.54/0.77	0.05/0.06	0.29/0.47	0.33/0.46	0.28/0.40	0.28/0.39
rsd1	0.54/0.77	0.06/0.08	0.33/0.49	0.23/0.30	0.41/0.58	0.25/0.34
rsd2	0.56/0.79	0.06/0.07	0.36/0.53	0.36/0.50	0.34/0.48	0.32/0.45
rsx0	0.56/0.79	0.06/0.07	0.37/0.54	0.36/0.49	0.35/0.51	0.32/0.45
rsx1	0.57/0.79	0.06/0.07	0.37/0.55	0.36/0.50	0.36/0.56	0.32/0.47
rsx2	0.56/0.78	0.06/0.07	0.38/0.56	0.32/0.45	0.38/0.60	0.31/0.46
rsx3	0.56/0.78	0.06/0.07	0.37/0.55	0.36/0.48	0.38/0.59	0.32/0.47
rsx4	0.55/0.77	0.06/0.07	0.36/0.54	0.34/0.47	0.37/0.59	0.31/0.46
rsx5	0.56/0.78	0.06/0.07	0.36/0.55	0.35/0.48	0.36/0.57	0.32/0.46
rsx6	0.56/0.79	0.06/0.07	0.37/0.54	0.36/0.49	0.36/0.56	0.32/0.46
rsx7	0.57/0.79	0.06/0.07	0.38/0.56	0.36/0.49	0.38/0.57	0.32/0.47
rsx8	0.56/0.78	0.06/0.07	0.37/0.55	0.36/0.48	0.39/0.60	0.33/0.47

Table 6.3.5: Ranking results with *qsim* + *ctype* filtering and $\gamma = 0.55$ selection

quency based techniques.

With the cross-over schemes we try to combine the effects of all classes. After a multitude of iterations – only eight settings are noted here – we found the results to peek at the *rsx8* scheme. We reach around 60% approximately and 40% strictly matched precision results in the selection at a similar recall, which yields up F-scores of 33%/47%.

6.4 Answer Entities Selection

As noted before, we can not completely isolate the evaluation of each component without affecting the overall results. For the same reason, we could not avoid showing the final selection results in the previous section. In this section, we discuss how we determined the optimal selection parameters and how much the results diverge from the theoretical optimal selection.

The *moving average pivot* selection described in 4.7.1 is a lightweight approach to answer entities selection. We recall that the selection E_S is based on a threshold score δ, which sets the minimum score for the answer entities:

$$E_S = \{e \in E_C \mid e_s \geq \delta\} \quad \text{with} \quad \delta = S_{avg} + (2\gamma - 1)(S_{max} - S_{avg})$$

Similar to the *R-precision*, we define the selection recall $R@S = R(E_S)$ and selection precision $P@S = P(E_S)$.

6.4.1 Selection Parameter Optimization

Before evaluating the results of the selection method, we need to find the optimal parameter setting for our scenario. The only value to tweak is γ, which controls the cutoff offset from the average base.

We test out the range $[0, 1]$ for γ and narrow down the optimal value to a reasonable degree. Table 6.4.1 shows the results of our experiments. As expected, lower cutoff offsets retain more of the relevant entities at the cost of lower precision, where higher offsets increase the precision of the selection. The optimal setting for the query selection is approximately at $\gamma = 0.55$, which is used throughout the rest of the evaluation, if not noted otherwise.

γ	R@S	P@S	F@S
0.00	0.56/0.78	0.06/0.10	0.10/0.16
0.25	0.56/0.78	0.06/0.10	0.10/0.16
0.35	0.55/0.77	0.08/0.13	0.12/0.19
0.45	0.50/0.70	0.17/0.26	0.23/0.34
0.50	0.46/0.63	0.26/0.38	0.30/0.43
0.52	0.45/0.62	0.29/0.42	0.32/0.45
0.55	0.43/0.59	0.34/0.49	0.34/0.48
0.57	0.41/0.56	0.33/0.51	0.33/0.47
0.60	0.38/0.52	0.35/0.54	0.33/0.47
0.65	0.35/0.47	0.39/0.58	0.32/0.46
0.75	0.21/0.31	0.39/0.62	0.23/0.36
1.00	0.13/0.18	0.38/0.60	0.17/0.24

Table 6.4.1: Selection optimization results with ranking scheme *rsx8*

6.4.2 Selection Optimality

The goal of the selection is to provide the correct answer entities based on the ranked candidates. Our approach to this is to find a score threshold to separate the candidate set into selected and non-selected entities.

Given a set of ranked candidate entities, we can manually determine an *optimal cutoff*, which guarantees the highest F-scores. In this section, we want to compare such an optimal section with our selection method. The optimal selection $E_{S_{opt}}$ is defined as follows:

$$E_{S_{opt}} = E_C|_k \quad \text{with} \quad k = \arg\max_k F(E_C|_k)$$

To assess the performance of the selection we define the *selection optimality* measure:

$$Q_S = \frac{F(E_S)}{F(E_{S_{opt}})}$$

Table 6.4.2 shows the average results for all queries, the complete results can be found in A.3. For the strict matching, the quality of the selection is at 78% of the optimal and it's 71% for the approximately matched results.

Query	$F@S_{opt}$	$R@S$	$P@S$	$F@S$	Q_S
1-47	0.45/0.65	0.43/0.59	0.34/0.49	0.34/0.48	0.78/0.71

Table 6.4.2: Moving average pivot selection results

The verdict on our selection method is overall positive. The results show that the MAP selection is suitable as a baseline technique for the answer entities selection with close to 80% optimality. However, we don't think that further refinements of the MAP selection could improve its quality without introducing bias, more promising research should concentrate on combining this technique with other approaches.

6.5 Deep Search

Based on the results of the first phase, we construct a query to be used on the Broccoli search engines. In this section, we evaluation the results of this phase, as they are returned by Broccoli.

We do not filter the results of this phase, but we apply the MAP selection method to increase the precision of the output. Just as in the selection optimization for the first phase, we evaluate a range of γ values to determine the optimal cutoff.

The scores of the deep search results follow a different distribution than the scores our system produces in the first phase, which makes it difficult to adjust the parameter to achieve the best results. Where in the first phase, we could boost the precision from 6%/7% to 38%/60% on average – a factor greater than 6 – the best improvement factor at this stage is around 2.5.

Query	Phase	R	P	P@R	R@S	P@S	F@S
1-47	1	0.56/0.78	0.06/0.07	0.38/0.56	0.33/0.47	0.38/0.60	0.31/0.46
1-47	2	0.44/0.54	0.09/0.12	0.24/0.31	0.20/0.27	0.22/0.31	0.19/0.25

Table 6.5.1: Average results for entity retrieval (1) and deep search (2) phases

Table 6.5.1 shows the comparison of the average results between both phases, you can find the complete results in A.2. The results indirectly mirror the quality of our semantic query construction, but they are also influenced by the result quality of Broccoli and the completeness of its knowledge base.

It's a poor sort of memory that only works backwards.

Lewis Carroll

7 Discussion

We have taken on the grand task of enabling keyword-based semantic search – but have we achieved all our goals? In this final chapter, we conclude our work on Pythia, discuss the presented results and give an outlook on future work that could improve the performance of the two-phase approach.

7.1 Conclusion

We introduced a multi-phased approach to enable keyword-based queries for semantic search and developed Pythia to evaluate its performance. Most existing work on providing convenient user interfaces for semantic search offer interactive query composition or direct semantic mapping of the keywords to a formal query. Such solutions do not provide the desired user experience or create artificial constraints on the accepted user queries. Pythia demonstrates that we can go beyond such constraints and offer a better user experience by providing a barrier-free keyword-based interface for semantic search.

The evaluation of the first phase gives a positive outlook on the capabilities of our approach with relatively high precision results in the answer selection. The evaluation shows that many results, when examined in detail, seem subjectively better than the numbers suggest, which is also supported by the approximate matching results. Some simple filtering methods have proved to be effective at reducing the number of candidate entities without high regression on recall. However, we could not effectively cope with the noise introduced by the low quality of the entity extraction. We attacked it with lexical entity clustering, but this introduced other issues, which diminished its positive effects.

The results of the deep search phase are considerably worse than the intermediate results from the first phase. The most diverging metric between the two phases is the precision of the selection, which is a natural consequence of our cutoff mechanics, which are better suited for highly quantitative results. We also do not apply any filtering at this stage, which proved to be effective for the results in the first phase. The semantic query construction failed to provide a correct mapping in many cases, that way we could not reproduce the result quality of manually composed Broccoli queries.

7.2 Future Work

The results of the entity retrieval phase are competitive with similar systems and we would like to extend some features to further increase the result quality at this stage. We consider the cause of many issues to be related to the inexact named entity extraction and the entity type classification. To counter this, further decreasing the dependence on NLP tools should be a desirable goal to pursue. Noun detection is more reliable and could replace the named entity extraction when combined with SAT classification based on Freebase[1]. Often, the query and LAT give good hints on the SAT – such as queries starting with *"who"* or *"where"* – adding pattern-based type matching could provide a considerable boost to SAT detection independent of the knowledge base.

With improved SAT detection, we would have a much better base for the construction of the semantic query. A deeper analysis of the relation type between keywords of the user query, could also increase the expressiveness of the semantic query and improve the results in the deep search phase.

Entity linking as a combination of lexical matching and semantic type detection is a another desirable feature, which could be used to refine the set of candidate entities. Complementary to that, an extended result set in the document retrieval phase should further improve the results by the increased frequencies of relevant entities.

[1]http://www.freebase.com

We think that some methods introduced in this thesis are well suited to be integrated into a fully-featured semantic search engine to harness its semantic database for improved type detection and that way enable keyword-based queries for semantic search.

A

Appendices

A.1 Yahoo Semantic Search Queries

Q1	apollo astronauts who walked on the moon
Q2	arab states of the persian gulf
Q3	astronauts who landed on the Moon
Q4	axis powers of world war II
Q5	boroughs of new york city
Q6	branches of the us military
Q7	continents in the world
Q8	nicole kidman's siblings
Q9	dioceses of the church of ireland
Q10	first targets of the atomic bomb
Q11	five great epics of tamil literature
Q12	gods who dwelt on mount olympus
Q13	henry ii's brothers and sisters
Q14	hijackers in the september 11 attacks

Q15	houses of the russian parliament
Q16	john lennon's parents
Q17	kenya's captain in cricket
Q18	kublai khan siblings
Q19	lilly allen parents
Q20	major leagues in the united states
Q21	manfred von richthofen parents
Q22	matt berry tv series
Q23	members of u2
Q24	movies starring erykah badu
Q25	movies starring joe frazier
Q26	movies starring rafael rosell
Q27	nations where Portuguese is an official language
Q28	orders or choirs of angels
Q29	permanent members of the un security council
Q30	presidents depicted on mount rushmore who died of shooting
Q31	provinces and territories of canada
Q32	ratt albums
Q33	republics of the former yugoslavia
Q34	revolutionaries of 1959 in cuba
Q35	standard axioms of set theory
Q36	states that border oklahoma
Q37	ten ancient greek city-kingdoms of cyprus
Q38	the first 13 american states
Q39	twelve tribes or sons of israel
Q40	what books did paul of tarsus write
Q41	what languages do they speak in afghanistan
Q42	what tv shows has thomas jane been in
Q43	where the british monarch is also head of state
Q44	who created stumbleupon
Q45	who has jackie weaver been married to
Q46	who invented the python programming language
Q47	wonders of the ancient world

A.2 Combined Search Results

Query	R	P	P@10	P@R	R@S	P@S	F@S
1-47.1	0.56/0.78	0.06/0.07	0.23/0.36	0.38/0.56	0.33/0.47	0.38/0.60	0.31/0.46
1-47.2	0.44/0.54	0.09/0.12	0.17/0.22	0.24/0.31	0.20/0.27	0.22/0.31	0.19/0.25
1.1	1.00/1.00	0.12/0.13	0.50/0.80	0.58/0.83	0.25/0.33	0.50/0.67	0.33/0.44
1.2	1.00/1.00	0.29/0.31	0.70/0.70	0.75/0.75	0.50/0.50	0.86/0.86	0.63/0.63
2.1	0.88/1.00	0.10/0.11	0.70/0.80	0.88/1.00	0.88/1.00	0.88/1.00	0.88/1.00
2.2	1.00/1.00	0.21/0.23	0.70/0.70	0.75/0.75	0.50/0.50	0.80/0.80	0.62/0.62
3.1	1.00/1.00	0.16/0.17	0.40/0.40	0.42/0.50	0.00/0.08	0.00/0.50	0.00/0.14
3.2	0.00/0.50	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
4.1	0.33/1.00	0.01/0.01	0.00/0.30	0.00/1.00	0.00/1.00	0.00/1.00	0.00/1.00
4.2	1.00/1.00	0.00/0.00	0.20/0.30	0.33/0.33	0.67/1.00	0.14/0.21	0.24/0.35
5.1	1.00/1.00	0.13/0.15	0.50/0.50	0.80/1.00	0.80/1.00	0.80/1.00	0.80/1.00
5.2	0.80/0.80	0.03/0.04	0.40/0.40	0.80/0.80	0.80/0.80	1.00/1.00	0.89/0.89
6.1	0.80/1.00	0.11/0.14	0.10/0.50	0.20/0.80	0.20/0.80	0.20/0.80	0.20/0.80
6.2	0.40/0.80	0.00/0.00	0.20/0.20	0.40/0.40	0.40/0.40	0.18/0.18	0.25/0.25
7.1	1.00/1.00	0.11/0.13	0.70/0.70	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00
7.2	0.86/0.86	0.03/0.04	0.10/0.20	0.14/0.29	0.29/0.43	0.12/0.18	0.17/0.25
8.1	1.00/1.00	0.01/0.02	0.10/0.10	1.00/1.00	1.00/1.00	0.33/0.33	0.50/0.50
8.2	1.00/1.00	0.00/0.01	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
9.1	0.00/0.50	0.00/0.01	0.00/0.40	0.00/0.47	0.00/0.03	0.00/1.00	0.00/0.05
9.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
10.1	1.00/1.00	0.03/0.05	0.20/0.20	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00
10.2	1.00/1.00	0.18/0.27	0.20/0.20	0.50/0.50	0.50/0.50	1.00/1.00	0.67/0.67
11.1	1.00/1.00	0.02/0.03	0.40/0.50	0.60/0.60	0.60/0.60	0.50/0.50	0.55/0.55
11.2	0.60/0.60	0.75/0.90	0.30/0.30	0.60/0.60	0.00/0.00	0.00/0.00	0.00/0.00
12.1	1.00/1.00	0.17/0.18	0.90/0.90	0.92/0.92	0.17/0.17	0.67/0.67	0.27/0.27
12.2	0.67/0.67	0.26/0.28	0.60/0.60	0.58/0.58	0.17/0.25	0.50/0.75	0.25/0.38
13.1	0.33/0.67	0.00/0.01	0.10/0.20	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
13.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
14.1	0.00/0.05	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
14.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
15.1	0.50/1.00	0.04/0.08	0.10/0.20	0.50/1.00	0.50/1.00	0.50/1.00	0.50/1.00

15.2	1.00/1.00	0.07/0.10	0.20/0.20	0.50/1.00	1.00/1.00	0.50/0.50	0.67/0.67
16.1	1.00/1.00	0.01/0.02	0.20/0.20	0.00/0.50	0.50/1.00	0.33/0.67	0.40/0.80
16.2	1.00/1.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
17.1	0.22/0.22	0.06/0.06	0.10/0.10	0.11/0.11	0.11/0.11	1.00/1.00	0.20/0.20
17.2	0.44/0.44	0.18/0.20	0.40/0.40	0.44/0.44	0.33/0.33	0.50/0.50	0.40/0.40
18.1	0.00/0.33	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
18.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
19.1	1.00/1.00	0.05/0.07	0.20/0.20	1.00/1.00	0.50/0.50	1.00/1.00	0.67/0.67
19.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
20.1	0.00/0.78	0.00/0.00	0.00/0.40	0.00/0.44	0.00/0.22	0.00/0.50	0.00/0.31
20.2	0.89/1.00	0.00/0.00	0.20/0.30	0.22/0.33	0.22/0.22	0.25/0.25	0.24/0.24
21.1	0.00/0.17	0.00/0.01	0.00/0.20	0.00/0.17	0.00/0.08	0.00/0.25	0.00/0.13
21.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
22.1	0.50/0.75	0.04/0.06	0.20/0.30	0.50/0.75	0.50/0.50	1.00/1.00	0.67/0.67
22.2	0.75/1.00	0.01/0.01	0.30/0.40	0.75/1.00	0.75/1.00	0.43/0.57	0.55/0.73
23.1	0.29/0.57	0.05/0.06	0.20/0.40	0.29/0.57	0.14/0.43	0.25/0.75	0.18/0.55
23.2	0.43/0.43	0.38/0.43	0.30/0.30	0.29/0.29	0.29/0.29	0.67/0.67	0.40/0.40
24.1	0.33/1.00	0.04/0.07	0.10/0.30	0.00/0.33	0.00/0.33	0.00/0.33	0.00/0.33
24.2	0.00/0.33	0.00/0.11	0.00/0.10	0.00/0.33	0.00/0.33	0.00/1.00	0.00/0.50
25.1	0.00/0.13	0.00/0.01	0.00/0.10	0.00/0.13	0.00/0.13	0.00/0.33	0.00/0.18
25.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
26.1	0.70/1.00	0.10/0.12	0.50/0.50	0.50/0.50	0.40/0.40	0.50/0.50	0.44/0.44
26.2	0.80/0.90	0.42/0.47	0.70/0.80	0.70/0.80	0.60/0.70	0.67/0.78	0.63/0.74
27.1	0.18/0.55	0.03/0.04	0.20/0.40	0.18/0.36	0.00/0.27	0.00/1.00	0.00/0.43
27.2	0.00/0.09	0.00/0.01	0.00/0.10	0.00/0.09	0.00/0.09	0.00/1.00	0.00/0.17
28.1	0.80/1.00	0.03/0.04	0.40/0.40	0.80/0.80	0.60/0.60	1.00/1.00	0.75/0.75
28.2	0.80/1.00	0.09/0.11	0.30/0.50	0.20/0.40	0.60/1.00	0.30/0.50	0.40/0.67
29.1	1.00/1.00	0.02/0.04	0.10/0.10	0.00/0.00	1.00/1.00	0.17/0.17	0.29/0.29
29.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
30.1	0.92/1.00	0.26/0.28	0.80/0.90	0.77/0.85	0.46/0.54	0.86/1.00	0.60/0.70
30.2	0.69/0.92	0.01/0.02	0.50/0.50	0.54/0.54	0.23/0.23	0.50/0.50	0.32/0.32
31.1	0.25/0.63	0.02/0.02	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
31.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
32.1	0.00/1.00	0.00/0.01	0.00/0.60	0.00/0.67	0.00/0.67	0.00/0.80	0.00/0.73
32.2	0.83/1.00	0.03/0.04	0.00/0.60	0.00/0.67	0.00/1.00	0.00/0.75	0.00/0.86
33.1	0.75/1.00	0.07/0.09	0.20/0.30	0.50/0.50	0.25/0.25	0.33/0.33	0.29/0.29

Query	R	P	P@10	P@R	R@S	P@S	F@S
33.2	0.00/0.25	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
34.1	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
34.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
35.1	1.00/1.00	0.14/0.16	0.60/0.60	1.00/1.00	0.83/0.83	1.00/1.00	0.91/0.91
35.2	1.00/1.00	0.23/0.27	0.50/0.50	0.67/0.67	0.17/0.17	0.50/0.50	0.25/0.25
36.1	0.46/0.54	0.13/0.14	0.20/0.30	0.23/0.31	0.15/0.15	0.29/0.29	0.20/0.20
36.2	0.62/0.85	0.57/0.63	0.60/0.80	0.62/0.85	0.00/0.00	0.00/0.00	0.00/0.00
37.1	1.00/1.00	0.18/0.19	0.70/0.70	0.77/0.77	0.08/0.08	1.00/1.00	0.14/0.14
37.2	0.92/1.00	0.03/0.03	0.10/0.10	0.08/0.08	0.00/0.00	0.00/0.00	0.00/0.00
38.1	1.00/1.00	0.05/0.07	0.20/0.20	0.25/0.25	0.25/0.25	0.20/0.20	0.22/0.22
38.2	1.00/1.00	0.02/0.03	0.40/0.40	0.25/0.25	0.25/0.25	0.25/0.25	0.25/0.25
39.1	0.00/1.00	0.00/0.02	0.00/0.80	0.00/0.79	0.00/0.43	0.00/0.86	0.00/0.57
39.2	0.00/0.29	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
40.1	0.29/0.93	0.03/0.04	0.10/0.30	0.07/0.29	0.07/0.29	0.08/0.33	0.08/0.31
40.2	0.00/0.21	0.00/0.01	0.00/0.20	0.00/0.14	0.00/0.00	0.00/0.00	0.00/0.00
41.1	0.04/0.43	0.02/0.03	0.10/0.40	0.04/0.35	0.04/0.13	0.14/0.43	0.07/0.20
41.2	0.00/0.17	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
42.1	0.08/0.25	0.02/0.03	0.10/0.20	0.08/0.17	0.00/0.00	0.00/0.00	0.00/0.00
42.2	0.00/0.67	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
43.1	0.31/0.50	0.04/0.04	0.30/0.30	0.19/0.19	0.19/0.19	0.60/0.60	0.29/0.29
43.2	0.19/0.19	0.10/0.11	0.20/0.20	0.19/0.19	0.06/0.13	0.20/0.40	0.10/0.19
44.1	1.00/1.00	0.14/0.29	0.10/0.10	1.00/1.00	1.00/1.00	0.50/0.50	0.67/0.67
44.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
45.1	1.00/1.00	0.06/0.08	0.40/0.40	0.50/0.50	1.00/1.00	0.57/0.57	0.73/0.73
45.2	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
46.1	1.00/1.00	0.04/0.07	0.10/0.10	1.00/1.00	1.00/1.00	0.50/0.50	0.67/0.67
46.2	1.00/1.00	0.50/1.00	0.10/0.10	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00
47.1	0.57/0.71	0.08/0.09	0.20/0.50	0.14/0.71	0.14/0.57	0.17/0.67	0.15/0.62
47.2	0.00/0.57	0.00/0.01	0.00/0.30	0.00/0.29	0.00/0.57	0.00/0.29	0.00/0.38
1-47.1	0.56/0.78	0.06/0.07	0.23/0.36	0.38/0.56	0.33/0.47	0.38/0.60	0.31/0.46
1-47.2	0.44/0.54	0.09/0.12	0.17/0.22	0.24/0.31	0.20/0.27	0.22/0.31	0.19/0.25

Table A.2.1: Results are referenced by query and phase numbers, e.g., 9.2 stands for query Q9, second phase (deep search)

A.3 Moving Average Pivot Selection Optimality

Query	$F@S_{opt}$	$R@S$	$P@S$	$F@S$	Q_S
1-47	0.45/0.65	0.43/0.59	0.34/0.49	0.34/0.48	0.78/0.71
1	0.67/1.04	0.75/0.92	0.56/0.69	0.64/0.79	0.96/0.76
2	0.93/1.00	0.88/1.00	0.88/1.00	0.88/1.00	0.94/1.00
3	0.58/0.79	0.25/0.33	0.50/0.67	0.33/0.44	0.58/0.56
4	0.02/1.00	0.00/1.00	0.00/0.38	0.00/0.55	0.00/0.55
5	0.80/1.00	0.80/1.00	0.80/1.00	0.80/1.00	1.00/1.00
6	0.33/0.89	0.20/0.80	0.20/0.80	0.20/0.80	0.60/0.90
7	1.00/1.00	1.00/1.00	0.64/0.64	0.78/0.78	0.78/0.78
8	1.00/1.00	1.00/1.00	0.17/0.17	0.29/0.29	0.29/0.29
9	0.00/0.48	0.00/0.03	0.00/1.00	0.00/0.05	1.00/0.11
10	1.00/1.00	1.00/1.00	0.50/0.50	0.67/0.67	0.67/0.67
11	0.67/0.89	0.80/1.00	0.36/0.45	0.50/0.62	0.75/0.70
12	0.74/0.74	0.33/0.33	1.00/1.00	0.50/0.50	0.68/0.68
13	0.15/0.31	0.33/0.67	0.09/0.18	0.14/0.29	0.93/0.93
14	0.00/0.02	0.00/0.00	0.00/0.00	0.00/0.00	1.00/0.00
15	0.50/1.00	0.50/1.00	0.33/0.67	0.40/0.80	0.80/0.80
16	0.67/1.00	1.00/1.00	0.29/0.29	0.44/0.44	0.67/0.44
17	0.20/0.20	0.11/0.11	0.33/0.33	0.17/0.17	0.83/0.83
18	0.00/0.02	0.00/0.00	0.00/0.00	0.00/0.00	1.00/0.00
19	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00
20	0.00/0.38	0.00/0.33	0.00/0.43	0.00/0.38	1.00/0.98
21	0.00/0.19	0.00/0.08	0.00/0.25	0.00/0.12	1.00/0.66
22	0.67/0.86	0.50/0.75	0.67/1.00	0.57/0.86	0.86/1.00
23	0.33/0.71	0.29/0.43	0.33/0.50	0.31/0.46	0.92/0.65
24	0.25/0.55	0.00/0.33	0.00/0.25	0.00/0.29	0.00/0.52
25	0.00/0.18	0.00/0.12	0.00/0.20	0.00/0.15	1.00/0.85
26	0.57/0.59	0.60/0.60	0.50/0.50	0.55/0.55	0.95/0.92
27	0.20/0.50	0.09/0.36	0.14/0.57	0.11/0.44	0.56/0.89
28	0.80/0.80	0.80/0.80	0.80/0.80	0.80/0.80	1.00/1.00
29	0.29/0.29	1.00/1.00	0.11/0.11	0.20/0.20	0.70/0.70
30	0.81/0.89	0.69/0.77	0.82/0.91	0.75/0.83	0.92/0.94

31	0.04/0.21	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
32	0.00/0.86	0.00/1.00	0.00/0.55	0.00/0.71	1.00/0.82
33	0.50/0.75	0.50/0.50	0.40/0.40	0.44/0.44	0.89/0.59
34	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00	1.00/1.00
35	1.00/1.00	1.00/1.00	0.75/0.75	0.86/0.86	0.86/0.86
36	0.26/0.34	0.15/0.15	0.25/0.25	0.19/0.19	0.74/0.56
37	0.90/0.90	0.54/0.54	0.70/0.70	0.61/0.61	0.68/0.68
38	0.31/0.33	0.50/0.50	0.22/0.22	0.31/0.31	1.00/0.92
39	0.00/0.84	0.00/0.57	0.00/0.73	0.00/0.64	1.00/0.76
40	0.20/0.47	0.21/0.50	0.16/0.37	0.18/0.42	0.91/0.91
41	0.08/0.38	0.04/0.17	0.10/0.40	0.06/0.24	0.73/0.63
42	0.14/0.24	0.08/0.08	0.33/0.33	0.13/0.13	0.93/0.57
43	0.29/0.29	0.19/0.19	0.43/0.43	0.26/0.26	0.91/0.91
44	1.00/1.00	1.00/1.00	0.50/0.50	0.67/0.67	0.67/0.67
45	0.73/0.73	1.00/1.00	0.44/0.44	0.62/0.62	0.85/0.85
46	1.00/1.00	1.00/1.00	0.33/0.33	0.50/0.50	0.50/0.50
47	0.32/0.74	0.14/0.71	0.11/0.56	0.12/0.63	0.40/0.85
1-47	0.45/0.65	0.43/0.59	0.34/0.49	0.34/0.48	0.78/0.71
Query	$F@S_{opt}$	$R@S$	$P@S$	$F@S$	Q_S

References

[1] Hannah Bast, Florian Bäurle, Björn Buchhold, and Elmar Haussmann, *Broccoli: Semantic full-text search at your fingertips*, CoRR **abs/1207.2615** (2012).

[2] Eric Brill, *A simple rule-based part of speech tagger*, 1992.

[3] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg, *A comparison of string distance metrics for name-matching tasks*, 2003, pp. 73–78.

[4] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, Pavel Kuksa, and Michael Collins, *Natural language processing (almost) from scratch. arxiv:1103.0398v1*, 2011.

[5] Yuangui Lei, Victoria Uren, and Enrico Motta, *Semsearch: A search engine for the semantic web*, Proc. 5th International Conference on Knowledge Engineering and Knowledge Management Managing Knowledge in a World of Networks, Lect. Notes in Comp. Sci., Springer, Podebrady, Czech Republic, Springer-Verlag, 2006, pp. 238–245.

[6] Vanessa Lopez and Enrico Motta, *Ontology-driven question answering in aqualog*, in proceedings of 9th international conference on applications of Natural Language to Information, 2004.

[7] Betsy Sparrow, Jenny Liu, and Daniel M. Wegner, *Google effects on memory: Cognitive consequences of having information at our fingertips*, Science **333** (2011), no. 6043, 776–778.

[8] Kristina Toutanova and Christopher D. Manning, *Enriching the knowledge sources used in a maximum entropy part-of-speech tagger*, In EMNLP/VLC 2000, 2000, pp. 63–70.

[9] Thanh Tran, Philipp Cimiano, Sebastian Rudolph, and Rudi Studer, *Ontology-based interpretation of keywords for semantic search*, The Semantic Web, Springer, 2007, pp. 523–536.

[10] Dong Wang, Qing Wu, Haiguang Chen, and Junyu Niu, *A multiple-stage framework for related entity finding*, Proceedings of the 19th Text REtrieval Conference (TREC), 2010.

[11] Qing Yang, Peng Jiang, Chunxia Zhang, and Zhendong Niu, *Reconstruct logical hierarchical sitemap for related entity finding,* Proceedings of the 19th Text REtrieval Conference (TREC), 2010.

Listing of figures

List of Tables

Declaration

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work. I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Offenburg, 26. February 2014

Place, Date

Eugen Sawin

Colophon

THIS THESIS WAS TYPESET using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 12 point Arno Pro, designed by Robert Slimbach in the style of book types from the Aldine Press in Venice, and issued by Adobe in 2007. The original template, which can be used to format a thesis with a similar look and feel, has been released under the permissive MIT (X11) license, and can be found online at github.com/suchow/ or from the author at suchow@post.harvard.edu.