Towards Precise and Convenient Semantic Search on Text and Knowledge Bases

Dissertation zur Erlangung des Doktorgrades
der Ingenieurwissenschaften (Dr.-Ing.)
der Technischen Fakultät
der Albert-Ludwigs-Universität Freiburg im Breisgau

vorgelegt von
Elmar Haußmann

Albert-Ludwigs-Universität Freiburg
Technische Fakultät
Institut für Informatik
2017

Abstract

In this dissertation, we consider the problem of making semantic search on text and knowledge bases more *precise* and *convenient*. In a nutshell, semantic search is *search with meaning*. To this respect, text and knowledge bases have different advantages and disadvantages. Large amounts of text are easily available on the web, and they contain a wealth of information in natural language. However, text represents information in an *unstructured* form. It follows no pre-defined schema, and without further processing, a machine can understand its meaning only on a superficial level. Knowledge bases, on the other hand, contain *structured* information in the form of *subject predicate object* triples. The meaning of triples is well defined, and triples can be retrieved precisely via a query language. However, formulating queries in this language is inconvenient and compared to text only a small fraction of information is currently available in knowledge bases.

In this document, we summarize our contributions on making semantic search on text and knowledge bases more precise and convenient. For knowledge bases, we introduce an approach to answer natural language questions. A user can pose questions conveniently in natural language and ask, for example, *who is the ceo of apple?*, instead of having to learn and use a specific query language. Our approach applies learning-to-rank strategies and improved the state of the art on two widely used benchmarks at the time of publication. For knowledge bases, we also describe a novel approach to compute relevance scores for triples from type-like relations like *profession* and *nationality*. For example, on a large knowledge base, a query for *american actors* can return a list of more than 60 thousand actors in no particular order. Relevance scores allow to sort this list so that, e.g., frequent lead actors appear before those who only had single cameo roles. In a benchmark that we generated via crowdsourcing, we show that our rankings are closer to human judgments than approaches from the literature. Finally, for text, we introduce a novel natural language processing technique that identifies which words in a sentence "semantically belong together". For example, in the sentence *Bill Gates, founder of Microsoft, and Jeff Bezos, founder of Amazon, are among the wealthiest persons in the world*, the words *Bill Gates*, *founder*, and *Amazon* do not belong together, but the words *Bill Gates*, *founder*, and *Microsoft* do. We show that when query keywords are required to belong together in order to match, search results become more precise.

Given the characteristics of text and knowledge bases outlined above, it is promising to consider a search that combines both. For example, for the query *CEOs of U.S. companies who advocate cryptocurrencies*, a list of CEOs of U.S. companies can be retrieved from a knowledge base. The information who is advocating cryptocurrencies is rather specific and changes frequently. It is, therefore, better found in full text. As part of this thesis, we describe how a combined search could be achieved and present and evaluate a fully functional prototype. All of our approaches are accompanied by an extensive evaluation which show their practicability and, where available, compare them to established approaches from the literature.

Kurzzusammenfassung

Diese Dissertation beschäftigt sich mit der Aufgabenstellung, semantische Suche in Text und Wissensdatenbanken präziser und komfortabler zu machen. Semantische Suche ist kurz gesagt eine „Suche mittels Bedeutung". In diesem Kontext haben Text und Wissensdatenbanken unterschiedliche Vor- und Nachteile. So ist Text in großer Menge im World Wide Web verfügbar und enthält eine Fülle an Informationen. Für Nutzer von Suchmaschinen ist die Suche nach Informationen über Schlagwörter einfach handzuhaben und hat sich für viele Anfragen als effektiv herausgestellt. Allerdings sind Informationen in Text *unstrukturiert*. Sie folgen keinem vorgegebenen Schema und ohne weitere Verarbeitung ist die Bedeutung von Text für eine Maschine nur oberflächlich erkennbar. In der Konsequenz ist es oft schwierig, präzise nach Informationen in Text zu suchen. Dagegen enthalten Wissensdatenbanken *strukturierte* Information in Form von Tripeln aus *Subjekt Prädikat Objekt*. Die Bedeutung von Tripeln ist für eine Maschine klar definiert und Tripel können präzise über eine spezielle Abfragesprache gefunden werden. Allerdings ist das Formulieren einer Abfrage in dieser Sprache umständlich. Zudem sind im Vergleich zu Text weniger Informationen in Wissensdatenbanken verfügbar.

Angesichts dieser Eigenschaften ist eine kombinierte Suche in Text und Wissensdatenbanken vielversprechend. Zum Beispiel kann für die Anfrage *CEOs von US-Firmen, die Kryptowährungen unterstützen* eine Liste mit CEOs von US-Firmen aus einer Wissendatenbank bezogen werden. Die Information, welche CEOs Kryptowährungen unterstützen, ist relativ spezifisch und kann sich häufig ändern. Derartige Information ist daher schwierig in einer Wissensdatenbank aktuell zu halten und lässt sich besser in Volltext, wie z.B. aktuellen Nachrichten, finden.

In dieser Arbeit beschreiben wir in Kapitel 3.1 zunächst die Idee der *Semantischen Volltextsuche*, eine kombinierte Suche in Text und Wissensdatenbanken. Wir präsentieren einen voll funktionsfähigen Prototypen, der alle wichtigen Probleme adressiert, um Suchen einfacher und Ergebnisse präziser zu machen. Dazu gehören eine natürliche Sprachverarbeitung, ein einfach zu verwendendes Benutzerinterface und eine einfach zu verstehende Wissensdatenbank. Anhand des Prototypen evaluieren wir das Potential der Semantischen Volltextsuche. Im Weiteren Verlauf der Arbeit präsentieren wir drei individuelle Problemstellungen, um semantische Suche auf Text oder Wissensdatenbanken präziser und komfortabler zu machen. Diese Problemstellungen sind nicht nur relevant für Semantische Volltextsuche, sondern darüber hinaus auch für andere Arten der semantischen Suche.

In Kapitel 3.2 stellen wir eine natürliche Sprachverarbeitungstechnik vor, um Ergebnisse von Schlagwortsuchen in Text präziser zu machen. Anstatt der bloßen Existenz von Schlagwörtern an beliebigen Stellen in einem Dokument müssen die Schlagwörter innerhalb eines Satzes in „inhaltlichem Zusammenhang" stehen, um als Treffer in Betracht zu kommen. In

dieser Arbeit zeigen wir anhand von Experimenten, dass Semantische Volltextsuche durch diese Technik präziser wird. Wir zeigen auch, dass die Technik verwendet werden kann, um Tripel - ähnlich derer in Wissensdatenbanken - aus Text zu extrahieren.

Des Weiteren stellen wir in Kapitel 3.3 eine Technik vor um die Relevanz von Tripeln aus Wissensdatenbanken für typähnliche Relationen wie *profession* oder *nationality* zu berechnen. Anhand der Relevanz von Tripeln lassen sich Ergebnisslisten sortieren. Eine Liste amerikanischer Schauspieler in der Wissensdatenbank Freebase enthält z.B. mehr als 60.000 Einträge in beliebiger Reihenfolge. In einer nützlichen Sortierung erscheinen Schauspieler mit häufigen Hauptrollen vor Schauspielern mit nur vereinzelten Nebenrollen. Unser Ansatz bestimmt die Relevanz von Tripeln anhand von Informationen aus Text. Da dies ein neuartiges Problem ist, erstellen wir mittels Crowdsourcing einen Datensatz zur Evaluation. Dieser Datensatz ermöglicht es, unseren Ansatz mit Ähnlichen aus der Literatur zu vergleichen. Er ist außerdem öffentlich für weitere Forschung verfügbar und findet bereits Verwendung (z.B. WSDM Cup 2017).

In Kapitel 3.4 stellen wir einen Ansatz vor, um Anfragen in natürlicher Sprache aus einer Wissensdatenbank zu beantworten. Ein Benutzer kann Fragen in natürlicher Sprache stellen, wie zum Beispiel *who is the ceo of apple?*, anstatt eine spezielle Abfragesprache erlernen zu müssen. Ein großes Problem hierbei ist die Entitätserkennung und -zuordnung: welche Wörter aus der Frage entsprechen welchen Entitäten aus der Wissensdatenbank. Im Beispiel bezieht sich *apple* auf die Entität *Apple Inc.*, die Firma, nicht auf *Apple*, die Frucht, und *ceo* auf die Entität mit der synonymen Bezeichnung *Managing Director*. Unser Ansatz löst gleichzeitig sowohl die Entitätserkennung und -zuordnung, als auch das Übersetzen in die Abfragesprache durch eine Abbildung auf ein Learning-to-Rank Problem. Wir evaluieren unseren Ansatz auf zwei, weit verbreiteten Benchmarks und übertreffen zum Zeitpunkt der Veröffentlichung den Stand der Technik.

Semantische Suche stellt kein einzelnes, wohldefiniertes Problem dar. Daher haben wir eine umfangreiche Zusammenfassung (158 Seiten) über das Gebiet der semantischen Suche auf Text und Wissensdatenbanken erstellt. Darin wird das Gebiet in neun Bereiche klassifiziert, basierend auf den zugrundeliegenden Daten (Text, Wissensdatenbanken, Kombinationen davon) und der Art der Suche (Stichwörter, strukturiert, natürliche Sprache). Diese Klassifizierung beschreiben wir im letzten Kapitel dieser Dissertation und vermitteln so einen kurzen Überblick über dieses weitreichende Gebiet.

Acknowledgments

First and foremost, I am grateful for my time at the University of Freiburg and the freedom I was provided to learn and explore during my studies at the Chair of Algorithms and Data Structures. Semantic search is a fascinating topic and the inspiring discussions, collaborations, and opportunities I encountered fostered my enthusiasm for it. I am especially thankful to my supervisor Hannah Bast who always provided guidance and support when I needed it, and who encouraged me to pursue a Ph.D. in the first place. My graduate studies were certainly one of the most challenging but also most rewarding times I have experienced so far.

I want to thank my colleagues Björn Buchhold, Claudius Korzen, Sabine Storandt, and Florian Bäurle. Our fruitful collaboration, lively discussions, as well as our travels to conferences around the world, were what made life at university exciting and diverse. Presenting months worth of research at conferences, receiving feedback from the audience, and discussing with peers is a reward on its own.

Last but not least, I want to thank my long-time partner and now wife, Simonette, who supported me throughout this dissertation and special period in my life. I cannot imagine having accomplished this without her.

Contents

1 Introduction

This thesis contributes to the field of semantic search on *text* and *knowledge bases*. Semantic search is not a single well-defined problem, but an umbrella term for all kinds of *search with meaning*. The kind of search and queries we consider in this work can be characterized by the kind of result. Queries always ask for one or more entities. An example query is *who is the ceo of apple?*[1], but we also consider more specific queries like *male friends of Ada Lovelace who are philosophers*. In all cases, we are concerned with making the search *convenient*. This means making it easy to use, for example, by allowing a user to pose a question in natural language instead of a complicated query language. We are also concerned with answering queries *precisely*. That is, for a given query we want to match and return the correct result or, in the case of a result list, rank it such that most relevant results come first. The two data sources we consider for answering queries are text and knowledge bases. With respect to convenience and preciseness, these are two complementary sources of information. Let us understand this by first looking at search on text followed by search on knowledge bases. Afterwards, we describe the individual problems we address in this context.

Given a query consisting of keywords, search on text returns a list of documents that match the keywords, or variations of them. For example, issuing the query *first computer programmer* to a web search engine returns a list of web pages which match these keywords in their text, title, or domain. As a top hit, we get a document with the title "Ada Lovelace: The First Computer Programmer". And we get the same match for variations of query keywords like synonyms (*computing*) or spelling mistakes (*computre*). To match this document, the search engine doesn't have to understand the query or the document's content. In fact, the keywords match prominently in the title as well as in the document's text: *Ada Lovelace has been called the world's first computer programmer*. Together, this indicates to the search engine that this is a good hit for the query.

This kind of search has turned out to be very successful, which is somewhat surprising since the underlying idea is rather simple. A major factor in the success is the massive amount of data that is now available on the World Wide Web. For example, the English Wikipedia alone contains about five million articles with detailed information on popular topics. For popular queries, a web page with the answer that also matches the keywords is likely to exist and formulating a keyword query is easy and intuitive for a user. Furthermore, the search engine requires no deep understanding of the text or query. Instead, these are treated as a mostly opaque and *unstructured* source of information.

[1] This example is from a benchmark [20] we use in one of our publications [17].

Of course, major search engines nowadays apply a myriad of techniques to improve this kind of search. For example, they analyze query logs and search sessions [77, 32], personalize results [59], and learn sophisticated ranking strategies [57]. However, the underlying search paradigm still reaches a limit for queries that require a deeper understanding or the combination of multiple sources of information.

Consider issuing the query *male friends of ada lovelace who are philosophers* on a web search engine. Certainly, the information is available somewhere on the Web, but likely not in a single document.[2] For example, the Wikipedia article on Ada Lovelace mentions that she was friends with Charles Baggage, but only his Wikipedia article mentions that he was a philosopher and nowhere is explicitly mentioned that Charles Baggage is male. However, the search still produces results, because keywords match "randomly" in various parts of documents, e.g., some keywords may match in the title, some in the abstract, and some in the main text. The user then needs to inspect each document, examine whether the content is helpful, and manually compile the list of persons she is looking for. What is needed is a deeper understanding of the query and documents.

Long queries with a narrow and specific intent as in this example are more common than one might expect. It has been observed that the length of queries on major search engines follows a power law distribution [46, 3]. Therefore, a significant proportion of queries consists of long queries. Bendersky and Gupta [46] cite that 17% of search engine queries contain five or more keywords. They note that long queries often have a narrow and specific intent and consist of composite queries with different subqueries - like the query above. They further argue that such queries may become even more frequent with the increasing availability of voice search, dialogue, and question answering systems.

While text contains a wealth of information in unstructured form, *knowledge bases* contain *structured* information. Here are some example statements about Ada Lovelace and Grace Hopper in the form of *subject predicate object* triples:

Ada Lovelace	is-a	Computer Scientist
Ada Lovelace	gender	Female
Ada Lovelace	place-of-birth	London
Grace Hopper	is-a	Computer Scientist
Grace Hopper	gender	Female
Grace Hopper	place-of-birth	New York City

Such statements, or *triples*, follow a precise and pre-defined schema. For example, there is only one way of expressing the birthplace of Ada Lovelace, whereas, in text, the same fact can be expressed with various sentences.

[2] We ignore the unlikely case where someone has manually compiled such a list.

Triples can be extracted automatically, e.g., from Wikipedia Infoboxes, or collected manually. The currently largest general purpose knowledge base is Freebase[3] [24], which contains a mix of automatic extractions and manual curations totaling about three billion statements on 40 million entities.

To retrieve statements from a knowledge base a *structured query language* called SPARQL[4] is used. The following shows an example query for *female computer scientists and their place of birth*:[5]

```
select ?entity ?place where {
  ?entity  is-a  Computer Scientist .
  ?entity  gender  Female .
  ?entity  place-of-birth  ?place .
}
```

The result is an unordered list of tuples of persons and their place of birth, containing, for example, *(Ada Lovelace, London)* and *(Grace Hopper, New York City)*. In contrast to search in text, the semantics of the query is precisely defined by the query language, as is the content of the knowledge base. If the answer is part of the knowledge base it can be retrieved via the correct query. However, formulating the query can be difficult for a user in the first place. It requires understanding the query language and the knowledge base schema, for example, what predicates and entities exist and what their identifiers are.

Knowledge bases allow precise and semantic queries, however, there will always be some information that is too specific to be included in a knowledge base, like who is friends with whom.[6] As a result, only a relatively small part of the world's knowledge is available in structured form. This is supplemented by a wealth of information available in full text. Combining search on knowledge bases and search on full text allows answering queries which cannot be easily answered by either one. Consider again the query for *male friends of ada lovelace who are philosophers*. A knowledge base query can be used to obtain a list of male philosophers. Finding out who was friends with Ada Lovelace can then be achieved via a full-text query. We consider the combined search on text and knowledge bases a promising search paradigm and present a viable approach as part of this work.

[3] Freebase was acquired by Google in 2010 and has been discontinued in 2015. All of the data is supposed to be migrated to Wikidata [82].

[4] `https://www.w3.org/TR/rdf-sparql-query/`

[5] We glance over a few syntax details of SPARQL for better readability, e.g., we omit namespaces.

[6] This has two reasons: First, adding new information in an accurate and complete way and keeping it up to date is expensive. Second, some circumstances are difficult to express in structured form, for example, sentiment or controversial or vague statements.

In this document, we summarize our work on problems that arise when answering semantic search queries using text, knowledge bases, or their combination. The queries above are good examples of the kind of queries we consider. They involve entities and their attributes, and they have one or more entities as a result. Such *entity queries* have been shown to make up a significant proportion of queries. In a query log analysis of a major web search engine, [70] found that almost 60% of queries had entities or their attributes as the desired result and more than 70% of all queries contained an entity or attribute. For academic search, [54] find that even 92% of all queries contain an entity.

We are concerned with making search *convenient*. This means making the search easy to use. For example, we present an approach that answers natural language questions, arguably the most convenient way of searching for a user. We are also concerned with answering queries *precisely*. That is, for a given query we want to match and return the correct result or, in the case of a result list, rank it such that most relevant results come first. In this context, we address and contribute to the following problems.

In Section 3.1, we introduce *semantic full-text search*, a combined search on text and a knowledge base. We present a fully functional prototype that includes our approaches on problems to make the search more precise and convenient. For example, we apply a novel natural language processing technique, *contextual sentence decomposition*, that returns more precise and semantic matches from the full text than the conventional matching strategy described above. We also provide an intuitive user interface with context-sensitive suggestions to construct queries, and we create a knowledge base that is easy to understand and use. In a quality evaluation, we show the potential of semantic full-text search.

Afterwards, we describe three individual problems that make search on text or knowledge bases more precise and convenient. These problems are not only relevant for semantic full-text search, but have applications beyond and on their own.

In Section 3.2, we describe how to perform contextual sentence decomposition. We already show in Section 3.1 that this technique can be used to get more precise results for semantic full-text search. In Section 3.2, we extend the technique to extract triples from text, similar to those contained in a knowledge base. In an evaluation, we compare our approach to existing approaches from the literature and show that our extracted triples are preferable for applications in semantic search.

In Section 3.3, we address how to return precise results for certain knowledge base queries. For example, on Freebase, a query for *american actors* returns a list of 64,757 actors in no particular order.[7] We present a technique to compute relevance scores in order to rank results such that, e.g., actors with frequent leading roles are ranked before those who only had few supporting roles. Crucially, we compute relevance scores from text, because

[7] This example is taken from our publication [15].

the required information is not reliably present in the knowledge base. Since this is a novel problem, we provide a benchmark created via crowdsourcing. We propose a variety of algorithms to compute ranking scores and evaluate them using our benchmark. The benchmark is also publicly available and already fosters further research on the problem (e.g., WSDM Cup 2017 [48]).

In Section 3.4, we show how to answer natural language questions from a knowledge base by automatically translating into a SPARQL query. A user can, for example, ask *who is the ceo of apple?* and is presented with the result from the knowledge base. This makes the search more convenient since the user doesn't have to formulate a structured query. A major problem in the translation is *entity recognition and disambiguation*: which words from the question correspond to which entities from the knowledge base. In the example, *apple* refers to the entity *Apple Inc.*, the company, not to *Apple*, the fruit, and *ceo* to the entity *Managing Director*, a synonym. Our approach solves both, entity recognition and disambiguation and query translation, by mapping to a single learning-to-rank problem. The approach focuses on giving precise results and improved the state of the art on two widely used benchmarks at the time of publication.

As we stated at the very beginning, semantic search is not a single well-defined problem. To give an overview of the vast field, we wrote an extensive survey on semantic search on text and knowledge bases [16] (156 pages). The survey classifies the field into nine groups based on the type of data that is used (text, knowledge bases, and their combination) and the kind of search that is performed (keyword, structured, and natural language). We describe the survey and our classification in Section 3.5 and thereby provide a short overview of the vast field.

This document summarizes our publications on the problems introduced above and is structured as follows. Chapter 2 lists all of our publications and attributes the work to individual authors. The complete publications are provided in the Appendix. In Chapter 3, we present the core of our work on these problems. Each section, 3.1 to 3.4, describes one problem and corresponds to one or more publications. In each section, we first give a concise description of the problem, related work, and our contributions. Then, we present the main ideas behind our approach, followed by a description of our experiments and main results. In all parts, we focus on the core of our approach. Detailed technical descriptions are available in the corresponding publications. In many places, we refer to our survey, which contains an extensive overview of a lot of recent related work. Finally, we conclude this dissertation with an outlook on future work in Chapter 4.

2 List of Publications

The following first lists our peer-reviewed publications followed by non peer-reviewed publications. For each publication, we give a short description and attribute the work to individual authors. The publications are grouped by topic which are ordered by how we describe them in the rest of this document. Note that authors for each publication are listed alphabetically by convention.

In the digital version of this dissertation, titles below (and in the references) are clickable and lead to an electronic copy. For readers of the hardcopy, we also provide direct links to the publications via a web page:
`http://ad-publications.cs.uni-freiburg.de/theses/Dissertation_Elmar_Haussmann.html`

2.1 Publications With Peer Review

A Case for Semantic Full-Text Search [9], Position Paper, SIGIR-JIWES 2012
Hannah Bast, Florian Bäurle, Björn Buchhold, and Elmar Haussmann

Position paper that motivates semantic full-text search and describes the requirements and resulting challenges. We address these as part of the papers below.

All authors wrote the paper.

Semantic Full-Text Search with Broccoli [12], SIGIR 2014
Hannah Bast, Florian Bäurle, Björn Buchhold, and Elmar Haussmann

Demo paper that presents our semantic full-text search prototype, its web application, and its public API. Covered in Section 3.1.

Research and implementation is based on the work listed below. All authors wrote the paper.

Easy Access to the Freebase Dataset [11], WWW 2014
Hannah Bast, Florian Bäurle, Björn Buchhold, and Elmar Haussmann

Demo paper that describes how to transform Freebase into a knowledge base that is easy to use and search. The paper also presents a web application that provides convenient access. Covered in Section 3.1.

HB, BB, and EH conducted the research. BB and EH implemented the ideas. FB adapted the semantic full-text search interface for the web application. HB, BB, and EH wrote the paper.

Open Information Extraction via Contextual Sentence Decomposition [19], ICSC 2013
Hannah Bast and Elmar Haussmann

Research paper that describes how to identify the parts of a sentence that semantically "belong together". The paper also describes how to extend the technique to extract triples from text (Open Information Extraction). Covered in Section 3.2.

Both authors conducted the research and designed the evaluation. EH implemented the ideas and performed the evaluation. Both authors wrote the paper.

More Informative Open Information Extraction via Simple Inference [18], ECIR 2014
Hannah Bast and Elmar Haussmann

Research paper that describes how to apply simple inference to increase informativeness of triples from Open Information Extraction. The technique can be incorporated into the triple extraction process from above [19]. Covered in Section 3.2.

Both authors conducted the research and designed the evaluation. EH implemented the ideas and performed the evaluation. EH wrote most of the paper with guidance and input by HB.

Relevance Scores for Triples from Type-Like Relations [15], SIGIR 2015
Hannah Bast and Björn Buchhold and Elmar Haussmann

Research paper that describes how to compute relevance scores for knowledge base triples. The scores can be used to properly rank results of entity queries on a knowledge base. Covered in Section 3.3.

All authors conducted the research and designed the crowdsourcing experiment and evaluation. BB and EH implemented the ideas and performed the evaluation. All authors wrote the paper.

WSDM Cup 2017: Vandalism Detection and Triple Scoring [48], WSDM 2017
Stefan Heindorf, Martin Potthast, Hannah Bast, Björn Buchhold, and Elmar Haussmann

Overview paper for the WSDM Cup 2017 and its two tasks, vandalism detection and triple scoring. The task is based on the work described in Section 3.3.

SH and MP organized the vandalism detection task. HB organized the Triple Scoring task. HB, BB and EH defined the triple scoring task, created a benchmark via crowd sourcing, and defined sensible evaluation metrics. SH, MP, and HB wrote the paper.

More Accurate Question Answering on Freebase [17], CIKM 2015
Hannah Bast and Elmar Haussmann

Research paper that describes how to translate natural language questions to SPARQL queries. This can be used to perform question answering on a knowledge base. Covered in Section 3.4.

Both authors conducted the research and designed the evaluation. EH implemented the ideas and performed the evaluation. Both authors wrote the paper.

Semantic Search on Text and Knowledge Bases [16], FNTiR 2016
Hannah Bast and Björn Buchhold and Elmar Haussmann

An extensive survey (156 pages) over the huge field of semantic search on text and knowledge bases. Covered in Section 3.5.

All authors contributed in deciding the overall structure and scope of the survey. All authors surveyed the literature and prepared summaries for systems to include or exclude. All authors wrote the survey.

2.2 Publications Without Peer Review

The following lists one publication at a non-peer reviewed venue and one publication that is still under submission at a peer-reviewed venue.

Broccoli: Semantic Full-Text Search at your Fingertips [10], CoRR 2012
Hannah Bast, Florian Bäurle, Björn Buchhold, and Elmar Haussmann

Research paper that describes a semantic full-text search prototype, Broccoli, including all important components: natural language pre-processing, search index, and user interface. Covered in Section 3.1.

All authors conducted the research on the general search paradigm, system design, and user interface. HB and EH conducted the research on contextual sentence decomposition. EH implemented it. HB and BB conducted the research on the search index and query processing and BB implemented it. BB, FB, and EH implemented the data pre-processing steps for the prototype (Wikipedia XML parsing, tokenizing, entity recognition and linking). FB implemented the user-interface. All authors performed the evaluation and wrote the paper.

A Quality Evaluation of KB+Text Search [14],
Invited Paper under Submission at KI-Journal 2017
Hannah Bast, Björn Buchhold, and Elmar Haussmann

Research paper that describes a detailed quality evaluation and error analysis of our KB+Text search paradigm. Covered in Section 3.1.

All authors designed the evaluation and analysed results. BB and EH performed most of the manual evaluation and error analysis. All authors wrote the paper.

3 Contributions

3.1 Semantic Full-Text Search

Semantic full-text search combines the capabilities of full-text search and search in knowledge bases. We have already motivated the idea in the introduction. In this section, we describe the idea in detail and present a fully functional prototype. The prototype addresses all important problems involved, including a natural language pre-processing technique to find precise matches in full text, an easy-to-use knowledge base, and an intuitive user interface to make the search convenient. We also describe an extensive quality evaluation that shows that our natural language pre-processing considerably improves results and that our prototype can answer a wide range of questions. A demo of our prototype, Broccoli, is available at `http://broccoli.cs.uni-freiburg.de`.

This section summarizes the work published in [10] at CoRR 2012, [9] as a position paper at SIGIR-JIWES 2012, [12] at SIGIR 2014, [11] at WWW 2014, and [14] at KI-Journal 2017 (under submission).

3.1.1 Problem, Related Work, and Contributions

Consider the example query for *plants with edible leaves and native to Europe*.[8] This query can be expressed in semantic full-text search in the following way:

```
select ?entity where {
  ?entity  is-a  Plant .
  ?entity  native-to Europe .
  ?entity  occurs-with  "edible leaves"
}
```

The query language is a subset of SPARQL extended with the special *occurs-with* relation. For the example, it combines search in full text and a knowledge base as follows. The first two triple patterns (*is-a Plant*, *native-to Europe*) match entities that are a plant and native to Europe in the knowledge base. One of the matches is, besides many other plants, broccoli. The special relation *occurs-with* requires that each plant, such as broccoli, is mentioned along the words *edible* and *leaves* somewhere in the full text. This requires that mentions of entities, e.g., broccoli, have been identified in the full text. For example, in the following sentence, matching words and entities are underlined:

The edible portions of broccoli are the stem tissue, the flower buds, as well as the leaves.

[8] This and the following example queries and sentences are taken from our publication [10].

In principle, this co-occurrence doesn't have to be restricted to a sentence but could also be within a larger unit, like a paragraph or document. The co-occurrence can also be in any text document, for example, in the Wikipedia article for broccoli or on a gardening website about leafy green vegetables. What is essential for a correct result, however, is that the co-occurrence provides evidence that broccoli indeed has edible leaves. We come back to this important problem below. Together, the results from the knowledge base and the full text can be used to infer a list of plants that match the query.

This kind of search combines the strengths of search on text and search on knowledge bases. Large knowledge bases, such as Freebase, focus on facts that are easy to define and extract automatically. A plant's origin is, for example, often part of Wikipedia's infobox. This makes it easy to find for an extraction system in order to populate the knowledge base. Once part of the knowledge base, it is straightforward to obtain a list of plants that are native to Europe, given the correct query.

Finding a full-text query to obtain this list from text is difficult. An answer requires factual knowledge in the first place, for example, that broccoli is a plant or that Italy (the originating country of broccoli) is a part of Europe. To determine whether broccoli has edible leaves it is the other way around. Corresponding facts are rather specific, more difficult to extract automatically, and, therefore, unlikely to be included in a knowledge base. However, the information is likely to be present in text, and it is easy to formulate a full-text query. The combination of both search paradigms allows answering queries that cannot be easily answered by either one.

Semantic full-text search uses *combined data*: a knowledge base as well as text in which entity mentions from the knowledge base are identified (like broccoli in the example sentence above). Queries are *semi-structured*: They are from a subset of SPARQL where the structured part matches facts in the knowledge base while the keywords given via the *occurs-with* relation match in the full text. There are two major approaches for searching in combined data that is followed in related work.

One prominent approach is that of creating *virtual documents*. All of the information pertaining to a specific entity is collected in a virtual document. For example, a document can be constructed for an entity by adding the subject, predict, and object of triples that have the entity as subject or object, together with text associated with that entity, like its Wikipedia page. Search can then be performed via classical keyword search. Since each document corresponds to an entity, results are lists of entities. This also allows applying traditional ranking functions from information retrieval, like BM25. Compared to semantic full-text search, keyword queries are more convenient to formulate for a user. However, our query language is far more powerful and allows more precise searches. The drawback is that a structured query needs to be constructed in the first place.

The virtual document approach has been popular, especially for searching the Semantic Web. The Semantic Web consists of triples similar to those in a knowledge base. Triples can be contributed by anybody, for example, via semantic markup in web pages. Because content can also be referenced and interlinked, the data from the Semantic Web is also referred to as *linked open data (LOD)*. No global schema of triples is enforced. Many of the triples contain literals so that a huge part of the data is actually text. Therefore, the two main challenges that approaches need to overcome are the sheer amount of data and the inconsistent schema.

Two established benchmarks to evaluate approaches for searching the Semantic Web are the TREC Entity Tracks (2010 and 2011) [4, 5] and the SemSearch challenges (2010 and 2011) [47, 81]. Two example queries from these benchmarks are *airlines that use boeing 747 airplanes* and *astronauts who walked on the moon*. The benchmarks come with a set of queries and desired results as well as a collection of triples extracted from the Semantic Web. Indeed, the best performing systems used keyword search on virtual documents with special attention to ranking. For example, [23] use a fielded inverted index with an adaptation of BM25 that boosts matches from certain relations. Since the kind of queries in these benchmarks are good examples for semantic full-text search, we used the two benchmarks in the evaluation of our prototype (see below).

The second prominent approach for searching in combined data is to perform keyword search on text, identify entities in the results, and rank them. Information from a knowledge base is usually applied in the ranking phase, but without formulating or translating to a precise SPARQL query that is used to restrict the result set. For example, the system described in [42] (the winning submission at the TREC Entity Track in 2009 [6]) issues queries to Google and identifies entities in the result documents using named entity recognition and disambiguation techniques. This establishes a link between the entity and its mention in text. Entities are then ranked using a combination of relevance scores based on how well the entity and the text surrounding the entity mention (document, sentence, or web table) match the query. In [76], the authors use the knowledge base to identify related similar entities (e.g., via the number of hops from one entity to another) that weren't matched prominently in text but that should be ranked higher. In contrast, semantic full-text search allows very precise and complex queries on the knowledge base that restrict the result set in the first place. Then, a simplistic ranking suffices to give good results as we show in our evaluation.

Besides the two approaches discussed above, there are various approaches for performing semantic search on text and knowledge bases alone. Details on related work in all of these areas is available in our survey on semantic search [16]. There are a few systems that perform combined search using (semi-) structured queries in a similar way to ours. We describe their difference to our semantic full-text search prototype below.

A good prototype implementation of semantic full-text search should give correct results and be easy to use. This requires the following, which we elaborate on next: finding correct matches in text with the *occurs-with* relation, an easy to use knowledge base, an intuitive user interface, and an efficient index and query processing. We have addressed these problems in our fully functional prototype, called Broccoli. Figure 1 shows a screenshot of our final prototype answering the query from above.

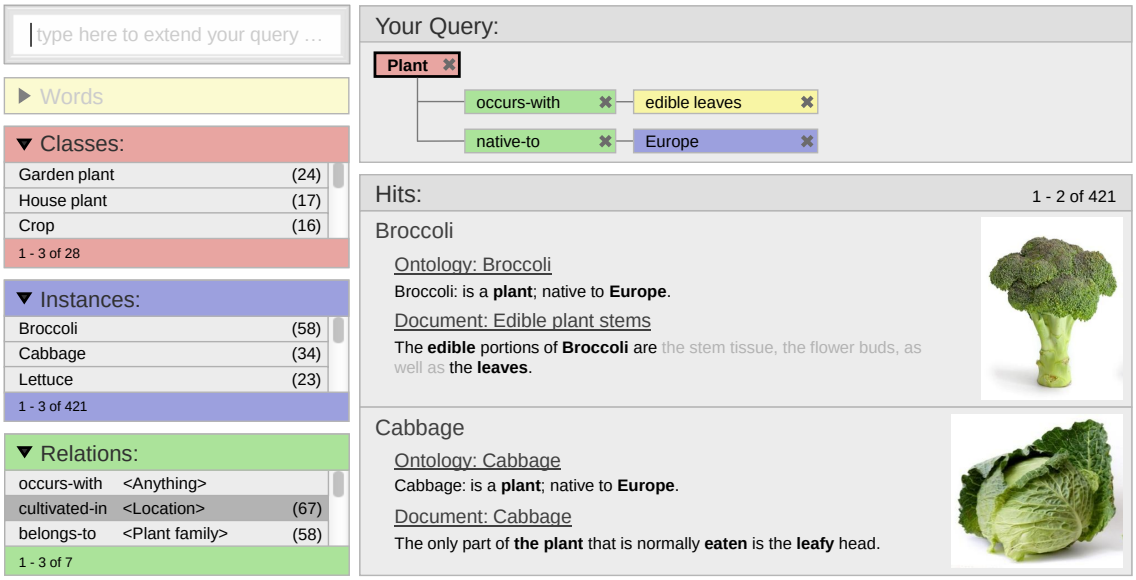

Figure 1. From our publication [10]: A screenshot for our example query. The structured query is visualized as a simplified tree on the top (the triple *?entity is-a Plant* is expressed by the root). Below, results are grouped by entity. Each result provides evidence from both the knowledge base and the full text. The search field on the top left can be used to extend the query. Below, suggestions for sub*classes* and *instances* of plants and additional *relations* are shown. These can be used to further refine the query. The suggestions are *context-sensitive*: They take the query so far into account and always lead to results. More screenshots can be found in [10].

As mentioned above, an important component of semantic full-text search is the *occurs-with* relation. It allows specifying co-occurrence of entities with words in text. Identifying that this co-occurrence is *semantic* is essential for the full-text part of queries. Consider the query *plants with edible leaves* and the following sentence with underlined matches:

The usable parts of rhubarb are the edible stalks, however its leaves are toxic.

All query elements (a plant, *rhubarb*, and words *edible leaves*) co-occur in the sentence, however, rhubarb is clearly a wrong answer (its leaves are toxic). Indeed, this problem is even more pronounced when restricting co-occurrence to the same paragraph or document, as we show in the experiments below. Ideally, the above match should be avoided, but it should still match a query for *plants with edible stalks*. We address this problem by

identifying words that "semantically belong together", which we call *contexts*. Intuitively the words *rhubarb*, *edible* and *leaves* do not belong together in the sentence above but the words *rhubarb*, *edible*, *stalks* do. Once identified, co-occurrence can be restricted to contexts and matches are more likely follow the intent of the query. The technique behind this idea is called *contextual sentence decomposition* and described in detail in Section 3.2. In this section, we introduce the basic idea and present an evaluation of the impact on semantic full-text search.

For finding results in text, formulating a keyword query via the *occurs-with* relation is easy, and the challenging part is semantically matching the keywords. This is the opposite for structured queries on a knowledge base. The result set is well defined and easy to retrieve because it is fully specified via the query language. However, the structured query can be difficult to construct in the first place. It uses unique entity as well as relation identifiers from the knowledge base. These are often opaque and difficult to determine. On top, the query must adhere to the (usually complex) schema of the knowledge base. For example, in Freebase, which we use for our prototype, the (seemingly) simple query for the *winners of the Palme d'Or* looks like this:[9]

```
select ?name where {
  ?x  ns:award.award_winner.awards_won  ?m  .
  ?m ns:award.award_honor.award  ?a  .
  ?a  ns:type.object.name  "Palme d'Or"@en  .
  ?x  ns:type.object.name  ?name  .
}
```

Who won a Palme d'Or is expressed via an intermediate *award nomination* object (*?m*) that connects the award (*?a*) and the person receiving the award (*?x*). The intermediate object is often referred to as *mediator* and allows to express n-ary relations, e.g., to link the person not only to the award but also to the date it was received and for which work it was awarded. Constructing the query above is not feasible without being acquainted with the schema of the knowledge base. The unwieldy relation and entity identifiers are especially problematic. This is apparent in Figure 1, which would look unpleasant without readable entity and relation identifiers.

We follow two approaches to tackle this problem. On the one hand, we construct a curated version of Freebase with simplified schema and readable identifiers. In this process, we resolve, for example, the complex award nomination relation from above to a binary *awards-won* relation. We also give entities their canonical and readable name as an identifier. Given the size of Freebase, this curation needs to be an automatic process.

[9] For readability, we omit namespace prefix definitions. The example query is from our publication [11].

On the other hand, we design a user interface that helps in incrementally constructing queries. A user can start to construct a query by selecting a class or entity (like *Palme d'Or*) and is presented with relations with meaningful names (e.g., *awards-won*) for this class or entity. At each step, she is guided by meaningful *context-sensitive* suggestions. Whenever the query is extended, new suggestions take the query so far into account so that all suggestions actually lead to results. This avoids constructing queries without results. Together, the curated knowledge base and user interface make it more convenient to explore the knowledge base and construct the correct query.

Once a query is constructed, the text and knowledge base must be searched to retrieve results. Besides reasonable query times ($<$ 100ms per query) several features are essential for our user interface, in particular, context-sensitive suggestions for words, classes, entities, and relations. Since no existing index structure provides this, we developed a special index that can answer our queries efficiently. The index and query engine is explained in detail in [13]. We don't consider it here any further since the corresponding work is not part of this thesis.

There are several related systems that perform *Semi-Structured Search on Combined Data* [16, Section 4.6.2]. ESTER [8] was one of the first systems. It uses a special-purpose index that also provides query suggestions from the text and knowledge base after each keystroke. Compared to ESTER our user interface is far more involved and instead of documents our system returns lists of entities. KIM [69] and its successor Mímir [80] allow semi-structured search via two different indices based on off-the-shelf software, one for text and one for the knowledge base. This causes efficiency issues when the structured part of the query matches many entities (e.g., a query that requires the list of all persons in Freebase). Both systems return lists of documents and provide no context-sensitive user interface. STICS [49] finds documents with mentions of keywords, entities, or entities of a certain category. A knowledge base provides the information which entity belongs to which category, but no further relations from the knowledge base can be used in queries. Suggestions for entities and categories are provided by a user interface and ranked by coherence. For example, as the authors note, the suggestions for the category *Ukranian politicians* rank the Klitschko brothers higher, once *Angela Merkel* is added to the query, since they are long-time German residents. All of the systems above lack our natural language processing technique to improve matches in the full text.

In the next subsection, we describe our approaches to the problems discussed above by means of our semantic full-text search prototype, Broccoli. Our main contributions are the following:

- The novel idea of semantic full-text search, which combines search in structured data with search in full text. Queries are a subset of SPARQL with a special *occurs-with* relation that can be used to specify co-occurrence of entities and words in text. In contrast to previous approaches, results are lists of entities (not documents).
- A novel kind of pre-processing that decomposes a sentence into contexts: parts that semantically "belong together". This is important for precise matches with the *occurs-with* relation. The technique behind this decomposition is called *contextual sentence decomposition* and described in detail in Section 3.2. In our evaluation in this section, we show that it makes the search more precise.
- A curated version of Freebase with unique readable entity and relation names and a simplified schema. We also improved the taxonomy, and the knowledge base comes with entity popularity scores for ranking. This makes it is easy-to-use, both for semantic full-text search and in other applications.
- An implementation of above components into a fully functional prototype system with an intuitive user interface. The user can incrementally construct her queries and is guided by meaningful suggestions on how to extend the query at each point.
- An extensive evaluation showing the potential of semantic full-text search in general and the improvements due to contextual sentence decomposition in particular.

3.1.2 Approach

On a high level, our semantic full-text search prototype, Broccoli, works as follows. First, a pre-processing pipeline takes as input a knowledge base and text. The pipeline links entity mentions in text to the knowledge base and decomposes sentences into contexts. The resulting contexts and knowledge base facts are stored in an index, which also provides a query engine with context-sensitive suggestions. The query engine is instrumented by a user interface which helps in constructing and executing queries and displays results. We start by explaining the knowledge base we use, followed by a description of the pre-processing pipeline and the user interface.

Easy-to-use knowledge base. As we illustrated above, the knowledge base should be convenient for an end user. In particular, it should have readable entity identifiers and an intuitive schema. This is not provided by the largest publicly available knowledge

base, Freebase. We transform Freebase into an easy-to-use version using a few simple yet effective techniques, which we describe in [11] and summarize below.

First, we obtain popularity scores for each entity by combining the number of mentions in a reference text corpus and the number of knowledge base triples where the entity occurs as subject or object. As a reference corpus, we use ClueWeb'12 [34] with entity mentions by [45]. We use these scores to assign the most popular entity the canonical name and resolve conflicts for other entities with the same name via heuristics. For example, for persons, we append the notable profession, as in *Michael Moore (Soccer Forward)*, and for locations the containing country, as in *Berlin (United States)*.

To simplify the schema, we resolve n-ary relations using a heuristic based on frequencies of involved facts. Intuitively, more facts connect award winners to their award than for what work or at what date it was won. Therefore, we create a new binary relation *Awards-Won* and ignore facts about the date or award-winnning work involved. The query for the winners of the Palme d'Or is now simple and intuitive:[10]

select ?x **where** { ?x Awards-Won "Palme d'Or" }

As part of the curation, we also merge duplicate types (like *Person* and *person*) by checking for overlap on their members. Finally, we compute the transitive closure for manually selected relations like *profession* and *specialization-of*. We use this to enhance the taxonomy, to ensure that, for example, a person who has the type *Physicist* (the specialization) also has the type *Scientist*. All of the above problems are addressed in an automatic way. The only required manual input is which relations to compute the transitive closures over. Details on the applied heuristics can be found in the corresponding publication [11].

Pre-processing pipeline. The pre-processing pipeline takes a knowledge base and text as input. Its main tasks are to link entity mentions in the text to the knowledge base, split the text into sentences, and decompose the sentences into contexts. We describe the pre-processing in more detail in [10].

Consider the following sentences, where mentions of the entity rhubarb are underlined:

> *The stalks of rhubarb are edible and its roots are medicinally used. However, the leaves of the plant are toxic.*

Identifying the first mention of rhubarb in the sentence is also known as *named entity recognition and disambiguation* or *entity linking*. This is an established problem, and a large body of approaches exists that solve this for any text input, e.g., [89, 72, 43]. The next two mentions, *its* and *the plant*, are references to the first mention of rhubarb. Identifying these is also known as *co-reference* or *anaphora resolution*.

[10] The example query is from our publication [11].

In our prototype, we use the following simple heuristics to identify all of these mentions. Since we use Wikipedia as text, we can obtain initial entity mentions from its markup: as an annotation rule (by Wikipedia) the first mention of an entity always references its Wikipedia page. By mapping Wikipedia pages to entities in the knowledge base[11] we obtain entity mentions of higher precision than would be possible with automatic approaches. We then resolve references as follows. Whenever a part or the full name of an entity is mentioned again, we recognize it as that entity (for example, *Ada* as *Ada Lovelace*). This is restricted to the same section of a document, which our pipeline is also able to identify. Additionally, we resolve pronouns to the last identified entity of matching gender and identify the pattern *the <class>* as the last entity of matching class. This identifies the references of *its* and *the plant* to rhubarb in the example above.

Given entity mentions in each sentence, the next step is to decompose sentences into parts that "belong together". This is important to achieve precise matches with the *occurs-with* relation. For example, consider the query *plants with edible leaves* and the following matching sentence with underlined matches:[12]

> *The usable parts of rhubarb, a plant from the Polygonaceae family, are the medicinally used roots and the edible stalks, however its leaves are toxic.*

We decompose this sentence into the following contexts:

(C1) *rhubarb, a plant from the Polygonaceae family*
(C2) *The usable parts of rhubarb are the medicinally used roots*
(C3) *The usable parts of rhubarb are the edible stalks*
(C4) *however rhubarb leaves are toxic*

After decomposition, the query for *plants with edible leaves* no longer matches any of these contexts, since *rhubarb* no longer co-occurs with the words *edible* and *leaves*. Note that entities and co-references (*its* refers to *rhubarb*) have been identified beforehand. Thus, no information is lost as part of this process. The technique for this is called *contextual sentence decomposition* and covered in detail in Section 3.2. It takes a deep grammatical parse tree of the sentence as input and applies tree transformations to retrieve contexts.

The pipeline performing entity linking and sentence decomposition as described above is implemented as a UIMA[13] chain. This allows us to easily exchange components, for example, to switch to a different entity linking strategy. The pipeline also contains components that parse the Wikipedia markup and split the text into (Wikipedia) sections, sentences, and words. UIMA also allows to scale out the pre-processing to a cluster of servers for computation intensive tasks like the required sentence parsing for our decomposition.

[11] Freebase provides the corresponding Wikipedia page for each entity, if it has one.
[12] The example is taken from our publication [10].
[13] http://uima.apache.org/

Index and user interface. Given the knowledge base and pre-processed text, we construct our search index. The index and query engine are explained in detail in [13], which is not part of the work of this thesis. The query engine reads results from the index and applies a simplistic but, as our evaluation shows, effective ranking. For a query that only uses the knowledge base, result entities are ranked by their popularity (see above). For queries that use the full text, results are ranked by their number of matches in text. The query engine also provides context-sensitive suggestions to the user interface.

We designed the user interface in an iterative process in which we developed ideas and features, tested, and then revised them. In the end we came up with the user interface already shown in Figure 1. It has the following main features as listed in our publication [10], where we also show additional screenshots:

- Search as you type: Each keystroke updates the results and suggestions. Suggestions are context-sensitive. They actually lead to hits and the higher-scored the hits, the higher scored their suggestion.
- Pre-selected suggestions: New users may be overwhelmed by the multitude of suggestions, therefore, the most likely suggestion is automatically highlighted.
- Visual query representation: The current query is always visualized as a tree. Suggestions are displayed for the element in focus, which can be changed using clicks.
- Transparency: Results are grouped by entity, and displayed together with context snippets that provide full evidence from the knowledge base and text.

The features above make it more convenient to construct queries. Furthermore, they make results transparent. While the first is important for new users the latter is especially important for researchers to identify mistakes. This helped in the analysis and evaluation of the system which we describe next.

3.1.3 Experiments and Results

Here, we report on our quality evaluation of our semantic full-text search prototype, Broccoli, which we describe in our publication [14]. We performed the evaluation with two goals in mind. First, we wanted to draw conclusions on the general potential of semantic full-text search. Second, we wanted to evaluate to what extent decomposing sentences into contexts makes results more precise.

The following was our experimental setup. As a text corpus, we used the English Wikipedia from January 2013. We performed entity recognition and contextual sentence decomposition as described above. This way we recognized a total of 285 million entity occurrences and decomposed 200 million sentences into 418 million contexts. To keep running times reasonable we scaled out this computation to a cluster of eight servers, each with an 8-core CPU and 16 GB of main memory. As a knowledge base, we used YAGO [79], which has about 26.6 million facts on 2.6 million entities. YAGO is smaller than Freebase but was the most promising knowledge base at that time. A larger knowledge base, like our curated version of Freebase from above, would improve all results presented here since more questions are answerable directly from the knowledge base. The conclusions we draw on the potential of semantic full-text search in general and the improvement due to context decomposition are valid nonetheless.

We evaluated search quality using three benchmarks. Each benchmark consists of a set of queries and a corresponding ground truth of relevant results, i.e., lists of entities. We used 15 queries from the TREC Entity Track in 2009 [6]. An example query is *Airlines that currently use Boeing 747 planes.* From the SemSearch Challenge 2011 [81], we used 46 queries, for example, *Apollo astronauts who walked on the moon.* We also derived ten queries from manually compiled Wikipedia lists. These can be thought of as a query with the ground truth provided by the corresponding list, for example, the *List of participating nations at the Winter Olympic Games.*

For each benchmark, we manually translated each query into our query language in a straightforward way (i.e., we didn't tune the queries to the results). All constructed queries make use of the text corpus via the *occurs-with* relation. The ground truth for each query is usually a small, well-defined result set. We believe that in this case, the quality of the result set as a whole is more important than the ranking within the result set, i.e., the result set should be as precise as possible. Therefore, we first focused on set related measures: precision, recall, and F1, the harmonic mean of precision and recall.

We distinguished between three variants of semantics of *occurs-with*: co-occurrences restricted to *sections*, *sentences*, and *contexts*. Table 1 shows our results. For all benchmarks, sections achieve the highest recall. However, a lot of returned results are due to "random" co-occurrences within a section. This causes a large amount of false-positives resulting in the smallest precision and F1 score. Compared to sections, sentences decrease the number of false-positives resulting in a higher precision but at a cost of recall. Our contexts further decrease the number of false-positives and also slightly increase the number of false-negatives[14]. This results in an increased precision with a slight decrease in recall. However, the increase in precision is far more pronounced, so that overall, contexts yield the best precision and F1 score on all three benchmarks.

		#FP	#FN	Prec.	Recall	F1
SemSearch	sections	44,117	92	6 %	78 %	9 %
	sentences	1361	119	29 %	75 %	35 %
	contexts	676	139	39 %	67 %	43 %†
Wikipedia lists	sections	28,812	354	13 %	84 %	21 %
	sentences	1758	266	49 %	79 %	58 %
	contexts	931	392	61 %	73 %	64 %∗
TREC	sections	6890	19	5 %	82 %	8 %
	sentences	392	38	39 %	65 %	37 %
	contexts	297	36	45 %	67 %	46 %∗

Table 1. From our publication [14]: sum of false-positives and false-negatives and averages for other measures over all SemSearch, Wikipedia list, and TREC queries. ∗ and † denote a p-value of < 0.02 and < 0.003, respectively, for the two-tailed t-test compared to the figures for sentences.

To evaluate the potential of semantic full-text search, we also computed and compared ranking related measures. In a ranked list of results, *Precision at K* (P@K) is the precision within the top K results. *R-precision* is the precision within the top R results, where R is the number of relevant results according to the ground truth. The best system [28] at the TREC Entity Track in 2009 achieved an average P@10 of 45% and R-precision of 55%. Our system achieved an average P@10 of 58% and R-precision of 62%, which is a considerable improvement given the small range of results for previous systems.

[14]For TREC, the number of false-negatives actually decreases. This is due to how our parser preprocesses Wikipedia lists. It appends each list item to the preceding sentence allowing contexts to cross sentence boundaries. See our publication [14].

However, these results are not directly comparable for several reasons. First, the TREC system only used a text corpus, ClueWeb'09 [34] Category B, which, however, is bigger (50 million web pages) than our corpus. Second, we constructed our queries manually (albeit not tuning them to the result). This was permitted for this benchmark, albeit submissions that used automatic approaches yielded better results. Still, we believe this makes the problem easier for our system. This also motivates automatically translating into structured queries, which we address for knowledge base search in Section 3.4. Last, the ground truth was approximated via pooling results from the participants. This may put systems that are evaluated later on the same ground truth at a disadvantage [74].

A detailed error analysis on the TREC Entity Track questions revealed that most errors are caused by an incomplete ground truth (55% of false-positives) or errors in third party components (33% of false-positives and 63% of false-negatives): the knowledge base, the constituent parser used for context decomposition, or our entity recognition. If we assume all of these errors can be corrected, our system achieves an average F1 score of 86%, P@10 of 94%, and R-precision of 92%. This motivates work on third party components for correcting these errors. If we only assume a fixed ground truth (by adding missing entities), our system achieves an average F1 score of 65%, P@10 of 79%, and R-precision of 77%. Together, these results show the high potential of semantic full-text search.

3.2 Contextual Sentence Decomposition

Contextual sentence decomposition is the task of decomposing sentences into parts that semantically "belong together". We have already presented the basic idea in the previous section, where we also showed how it makes semantic full-text search more precise. As will become clear, a closely related problem is to extract *(subject) (predicate) (object)* triples from a sentence. Extracting these triples is an established task known as *Open Information Extraction* (OpenIE) [7].

In this section, we describe an approach for contextual sentence decomposition and how to extend it to extract OpenIE triples. Our goal is to extract as many correct triples as possible but also to keep them *minimal* and *informative*. These properties are neglected in previous work, but important in applications like semantic full-text search, where precise facts are essential. In our evaluation, we show that our approach matches the state of the art with respect to correctness but improves upon minimality and informativeness.

This section summarizes the work published in [19] at ICSC 2013 and [18] at ECIR 2014.

3.2.1 Problem, Related Work, and Contributions

The goal of contextual sentence decomposition is to compute, for a given sentence, all sub-sequences of words in that sentence that semantically "belong together". The sub-sequences are called the *contexts* of the sentence. Consider the following sentence:[15]

(S1) *Ruth Gabriel, daughter of the actress and writer Ana Maria Bueno, was born in San Fernando.*

A correct decomposition yields the following contexts:

(C1) *Ruth Gabriel was born in San Fernando*
(C2) *Ruth Gabriel, daughter of Ana Maria Bueno*
(C3) *actress Ana Maria Bueno*
(C4) *writer Ana Maria Bueno*

If we restrict co-occurrence of entities and words to these contexts, as we did with semantic full-text search (see Section 3.1), matches become more precise. For example, if we search for co-occurrences of a person with the word *writer* (with the intent of finding writers) these only match in C4 for Ana Maria Bueno. Her daughter Ruth Gabriel no longer co-occurs with the word *writer*.

[15] The example is taken from our publication [19].

A highly related problem is that of extracting triples from a sentence. For example, from the sentence above the following triples can be derived:

(T1) *(Ruth Gabriel) (was born in) (San Fernando)*
(T2) *(Ruth Gabriel) (is) (daughter of Ana Maria Bueno)*
(T3) *(Ana Maria Bueno) (is) (actress)*
(T4) *(Ana Maria Bueno) (is) (writer)*

Extracting such triples from a sentence is known as Open Information Extraction (OpenIE) [7]. As can be seen, the triples T1-T4 are already close to the contexts C1-C4. What is missing is an assignment of each word to subject, predicate, or object, and inserting the implicit relation *is* in T3 and T4.

In this section, we first describe how to perform contextual sentence decomposition and then how to extend it to an OpenIE system. We have already shown in the previous section that our approach for decomposition can considerably improve results for semantic full-text search. By extending our approach to an OpenIE system, we can directly compare it to other approaches from the literature. This allows an intrinsic evaluation of the technique based on extracted triples in addition to the extrinsic evaluation in the previous section.

Note that mapping subject, predicate, and object of an OpenIE triple to their corresponding elements in a knowledge base is not considered part of the problem. For example, in T1, the subject *Ruth Gabriel*, the object *San Fernando*, and the predicate *was born in* could be mapped to their identifiers in Freebase: *m.02z4mfm*, *m.0r0wy*, and *people.person.place_of_birth*, respectively. This distinguishes OpenIE from *relation extraction*, where only triples of a given knowledge base relation, like *place of birth*, should be extracted, but with many possible variations of mentioning it. An overview of relation extraction and OpenIE can be found in our survey [16, Section 3.6]. Both problems are examples of *information extraction* [75].

OpenIE triples can be used in, for example, knowledge base construction (after linking to canonical entities and relations) or simple semantic querying, for example, to find all subjects matching *(?x) (was born in) (San Fernando)*.[16]

Early OpenIE systems, e.g., TextRunner [87], ReVerb [41], and KnowItAll [40], use shallow natural language processing techniques, for example, part-of-speech tagging or chunking. Using this shallow syntactic information they either apply hand-crafted or automatically learned rules to identify triples. More recent systems, like OLLIE [58], ClausIE [37], and our system, use a deeper linguistic analysis in the form of a syntactic parse tree of the sentence. The parse tree expresses the grammatical structure of the sentence, which clearly identifies nested phrases and clauses. This allows more powerful rules that also work

[16] A demo of such a system is available at: `http://openie.allenai.org/`.

in complicated sentences, e.g., with nested relative clauses. OLLIE uses automatically learned rules on this parse tree. ClausIE and our system use manually defined rules. In contrast to ClausIE, our rules were developed with the additional goal of minimality (see below) and a direct application for semantic full-text search. Our publications [19, 18] and semantic search survey give more details on related approaches as well as related natural language processing techniques like pos-tagging, chunking, and parsing.

OpenIE systems are usually compared based on accuracy and number of extracted triples. Accuracy refers to the percentage of triples that are correct, i.e., that express a meaningful fact which is also expressed in the original sentence. In addition to this, we also consider *minimality* to be essential. Consider the following triple:

(Ruth Gabriel) (is) (daughter of the actress and writer Ana Maria Bueno)

This triple is correct, but it also contains two other facts, namely, that Ana Maria Bueno is an actress and a writer. Hence the triple is not minimal. This is problematic for applications like semantic full-text search, which assume that co-occurrences are "semantic". In our evaluation, we consider minimality in addition to the standard measures of accuracy and the number of extractions. This has not been addressed or evaluated in previous work.

Besides correctness and minimality, another important aspect of extracted triples is that of *informativeness*. Consider the sentence[17]

(S2) *The ICRW is a non-profit organization headquartered in Washington.*

and the extracted triples:

(U1) *(The ICRW) (is) (a non-profit organization)*
(U2) *(a non-profit organization) (is headquartered in) (Washington)*

Both triples are correct and minimal, but triple U2 is, by itself, not informative. The information that it is the ICRW that is headquartered in Washington is not explicit and cannot be found with a search in these triples (without the information that the object of U1 and subject of U2 refer to the same organization). Therefore, we propose to integrate a set of simple inference rules into the extraction process to increase informativeness. An informative triple that should be extracted instead of U2 is U3: *(The ICRW) (is headquartered in) (Washington).*

Informativeness has previously been addressed in [41] but only as part of correctness (uninformative triples were labeled as incorrect) and with a different definition, focusing on the predicate part of triples (is the predicate by itself informative). We explicitly consider informativeness of whole triples and in relation to the originating sentence.

[17] The example is taken from our publication [18].

To summarize, our main contributions are the following:

- An approach for contextual sentence decomposition that is based on a grammatical parse of a sentence. Compared to similar approaches, our output is tailored to an application in semantic search.
- An extension of contextual sentence decomposition to extract OpenIE triples. In an evaluation, we show that extracted triples match the state of the art with respect to correctness, but are better than existing systems with respect to minimality.
- A way to increase the informativeness of extracted triples by using inference rules during the extraction process. Our evaluation shows that a few simple inference rules can mitigate many uninformative triples.

3.2.2 Approach

We first describe how to perform contextual sentence decomposition followed by how to derive triples. This is described in more detail in [19]. We then outline how to improve informativeness of triples, with details available in [18].

Contextual sentence decomposition is performed in two steps. In the *sentence constituent identification* phase (SCI), we identify the basic "building blocks" of contexts and arrange them in a tree. *Sentence constituent recombination* (SCR) combines the identified constituents to form contexts.

Figure 2 shows an SCI tree for our example sentence:

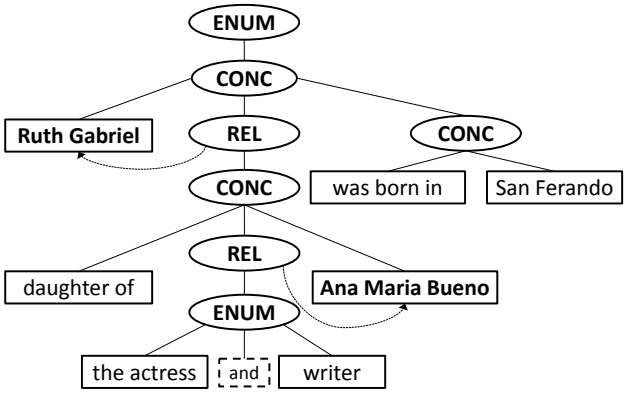

Figure 2. From our publication [19]: The SCI tree for our example sentence. The head of each relative clause is printed in bold, filler words in striped rectangles.

(S1) *Ruth Gabriel, daughter of the actress and writer Ana Maria Bueno, was born in San Fernando.*

The goal of SCI is to compute such a tree for a given sentence. We distinguish between three different types of internal nodes. ENUM nodes identify enumerations, where child nodes belong to different contexts. In our example, *the actress* and *writer* are two separate facts that describe Ana Maria Bueno, but have nothing to do with the rest of the sentence. REL nodes mark relative clauses, which form separate contexts with their optional head[18]. In our example, the nominal modifier (which we consider a type of relative clause) starting with *daughter of...* is connected to its head, *Ruth Gabriel*, whom it describes. Finally, CONC nodes group child nodes that belong to the same context. As is illustrated in Figure 2, words of a sentence (terminals) are only contained in leaf nodes. Nodes can be nested recursively and arbitrarily deep. In practice, deep nestings are rare, since the sentence also becomes hard to read for humans. Note how the SCI tree already expresses a lot of the semantic structure of the sentence.

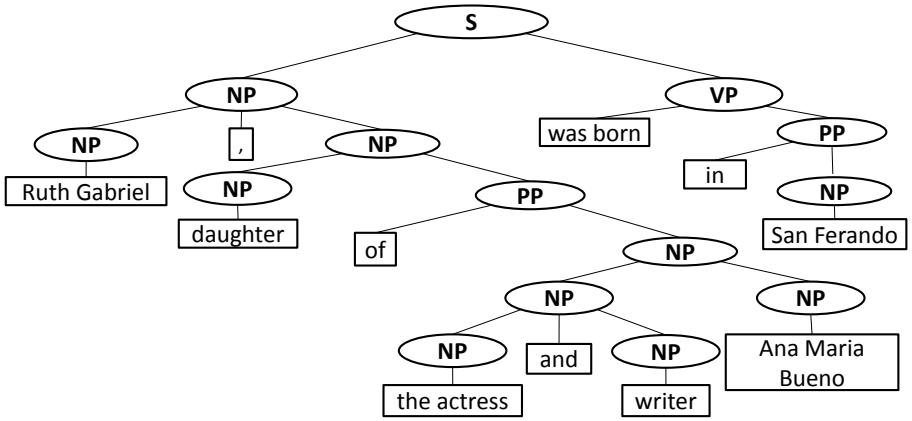

Figure 3. From our publication [19]: The constituent parse tree for our example sentence. It arranges noun phrases (NP), verb phrases (VP) and prepositional phrases (PP), according to the syntax of the sentence. For the sake of readability, the parse tree has been simplified.

A good starting point to derive the SCI tree is the constituent parse tree of the sentence. Figure 3 shows the parse tree for our example sentence. It can be seen that the tree hierarchically groups important constituents for the SCI tree. Identifying these constituents based on a more shallow analysis, like part-of-speech tags, is difficult.

We carefully designed a small set of 14 (prioritized) rules to transform a parse tree into an SCI tree. An applied rule in our example is: "in a sequence consisting of two NPs, split by a comma, mark the second NP as REL and the first NP as its head" [19]. This identifies the relative clause *daughter of the actress and writer Ana Maria Bueno* and its head, *Ruth Gabriel*. Another applied rule is: "mark a node as ENUM for which children all have the same type (e.g., NP)" [19]. This identifies the enumeration in *the actress and*

[18]In our publication, we also describe SUB nodes, which are REL nodes without the optional head.

writer. The complete list of rules can be found in our publication [19]. After applying all of the rules, we get the tree shown in Figure 2.

Given the SCI tree, computing contexts is straightforward. We first take out subtrees labeled REL, and change the root of this new tree to CONC. If the relative clause had a head, we attach it as first left child to the new tree. Then we recursively compute contexts for each tree, which now only contain leafs, ENUM, or CONC nodes. The context of a leaf consists exactly of the words contained in it. The contexts of an ENUM node is computed as the union of the sets of all child node contexts. The contexts of a CONC node is computed as the cross-product of the sets of its child node contexts. This gives us the desired contexts C1 to C4 shown in the beginning.

To transform contexts into triples, we apply a set of (relatively) simple heuristics. In each context, we identify the first explicit verb phrase and surrounding adverbs or prepositions to be the predicate. The words before the predicate belong to the subject and the words after it to the object. For example, in the context *Ruth Gabriel was born in San Fernando* we identify *was born in* as predicate and can derive the subject and object accordingly. Our heuristics also insert implicit verbs, for example, we use the verb *is* between the head and its REL attachment (if it doesn't begin with a verb). This allows deriving the triple *(Ana Maria Bueno) (is) (writer)* from the context *writer Ana Maria Bueno*.

In a final step, we improve informativeness of extracted triples using inference rules. For each triple, we first classify the predicate into one of five semantic relation classes.

$OTHER(A', B)$	$\leftarrow$	$OTHER(A, B) \wedge SYN(A, A')$
$OTHER(A', B)$	$\leftarrow$	$OTHER(A, B) \wedge SYN(A', A)$
$OTHER(A, B')$	$\leftarrow$	$OTHER(A, B) \wedge SYN(B, B')$
$OTHER(A, B')$	$\leftarrow$	$OTHER(A, B) \wedge SYN(B', B)$
$IN(A, C)$	$\leftarrow$	$IN(A, B) \wedge PART\text{-}OF(B, C)$
$IN(A, C)$	$\leftarrow$	$IN(A, B) \wedge IS\text{-}A(B, C)$
$OTHER(A, C)$	$\leftarrow$	$IS\text{-}A(A, B) \wedge OTHER(B, C)$

Table 2. From our publication [18]: Inference rules for new triples.

For example, IS-A expresses hyponymy, the relation between a specific instance and its more generic term, in triple U1: *(The ICRW) (is a) (non-profit organization)*. The predicate *is headquartered in* expresses a location placement, IN, in triple U2: *(a non-profit organization) (is headquartered in) (Washington)*. Other relation classes are SYN (synonymy), PART-OF (meronomy, for "part-whole" relationships), and OTHER for all remaining relations. Based on the semantic relations, we apply a set of seven inference rules shown in Table 2 to derive new triples.

In the example, the rule "if A IS-A B and B IN C → A IN C"[19] matches. We, therefore, conclude *(The ICRW) (is headquartered in) (Washington)*. Crucially, this inference step can only be performed during triple extraction for a given sentence, when involved subjects and objects refer to the same words in a sentence. In the example, it must be clear that *a non-profit organization* in triples U1 and U2 actually refer to the same real-world entity.

We also remove existing triples that we consider uninformative depending on how they were used to derive new triples. For example, we remove triple U2 in favor of the inferred triple above. While our rules are manually selected and simplistic, they improve informativeness of extracted triples as our evaluation shows.

3.2.3 Experiments and Results

We have already described an extrinsic evaluation of contextual sentence decomposition in Section 3.1, which showed that contexts make semantic full-text search more precise. Here, we describe an evaluation based on extracted OpenIE triples from [19]. Our system is called CSD-IE in the following.

For evaluation, we used two datasets from [37]: 200 random sentences from the English Wikipedia and 200 random sentences from the New York Times. We compared our system against three state-of-the-art OpenIE systems: ReVerb [41], using learned rules on part-of-speech tags, OLLIE [58], using learned rules on a parse tree, and ClausIE [37], using manually defined rules on a parse tree. In the first experiment, which we describe in [19], we didn't apply our inference technique to improve informativeness. We evaluated this separately in a second experiment.

For the first experiment, we extracted triples with each of the four systems and manually assigned two labels for each triple: one for correctness (yes or no) and one for minimality (yes or no). From these labels, we computed the following accumulated measures for each system as defined in [19]:

precision wrt accuracy (prec-a):	the percentage of triples labeled as correct
precision wrt minimality (prec-m):	the percentage of correct triples labeled as minimal
coverage:	the percentage of words of the sentence that occur in at least one extracted triple for that system
average triple length in words:	average length of extracted triples for that system in words (ignoring special characters)

[19] More formally: $\mathit{IS\text{-}A}(A, B) \wedge \mathit{IN}(B, C) \rightarrow \mathit{IN}(A, C)$

	ReVerb	OLLIE	ClausIE	CSD-IE
#facts	249	408	610	677
#facts correct	188	230	421	474
prec-a	75.5 %	56.4 %	69.0 %	70.0 %
prec-m	87.2 %	80.4 %	57.0 %	76.8 %
coverage	47.2 %	62.7 %	95.4 %	97.5 %
triple length	7.3	9.7	11.0	8.4

Table 3. From our publication [19]: Results of our quality evaluation on the Wikipedia dataset.

Table 3 shows the results for the Wikipedia dataset. The results on the New York Times dataset are very similar and gave no additional insights. For both datasets, the results for previous systems closely agree with those reported in [37], confirming our labeling.

In comparison to the other systems, CSD-IE extracts the largest number of correct facts (*#facts correct*). It also provides the highest coverage (*coverage*) and largest number of extracted facts overall (*#facts*). ReVerb produces a higher percentage of minimal and correct triples, however, for a considerably lower number of extracted facts. In an application for semantic search this can be detrimental to search quality. In particular, this is likely to cause missing extractions, which in turn lead to missing results (lower recall).

Compared to ClausIE, the most similar system, CSD-IE achieves similar precision with respect to accuracy (*prec-a*) and coverage. The triples extracted by CSD-IE are shorter on average (*triple length*). Furthermore, the precision wrt minimality (*prec-m*) is 20% higher. This is a considerable improvement that can be attributed to the fact that our rules were explicitly tailored towards minimality.

An investigation of errors for CSD-IE on both datasets reveals that most of the inaccurate extractions are caused by mistakes in the parse trees. This is also what we observed in the analysis of our results in Section 3.1. Small mistakes, like attaching a subtree to the wrong parent, cause wrong extractions for all contexts and triples involving that parent-child relationship. Current state-of-the-art parsers still make such mistakes for about 8% of parent-child relationships.[20] More details on parsing and an overview on the current state of the art are available in our survey [16, Section 3.3].

We believe there are two worthwhile directions to handle wrong extractions caused by the parser. First, we could compute confidence scores for triples. Some parsers can provide an estimate of their confidence for each subtree being correct. Wrong subtrees should get assigned a low confidence score and triples from such subtrees could be assigned low

[20]Results in [52, Table 1] show that the best performing parser attaches 92% of words correctly. This is for a dependency tree, which is similar to a constituent tree with respect to parent-child relationships.

scores. Second, it would be interesting to extract triples without the help of a parse tree. This could be achieved via learning a task-specific extractor from scratch, i.e., without a pipeline of cascading tools that propagate errors. This has been shown to work well for other natural language processing tasks, especially sentence parsing [35].

In a second experiment, which we describe in [18], we investigated how our inference rules improve the informativeness of extracted triples. We manually assigned two labels for triples extracted from the 200 Wikipedia sentences from above: one for correctness (yes or no) and one for informativeness (yes or no). A correct triple was considered informative if there is no extraction that is more precise, according to the sentence it was extracted from. We gave an example for this above.

	#facts	#facts corr	#facts corr-inf	prec corr	prec corr-inf
No Inference	649	429	385	66%	90%
Inference	762	484	444	64%	92%

Table 4. From our publication [18]: Results for our quality evaluation with inference (top row) and without inference (bottom row) over the labels correct (corr) and informative (inf). *prec corr* refers to the percentage of all triples labeled correct, *prec corr-inf* to the percentage of correct triples labeled informative. The experiments used a faster parser, hence the results differ slightly from those in Table 3.

The results in Table 4 show that about 90% of all correctly extracted triples are informative. After applying our inference rules this increases to 92%. In addition recall increases. The number of correctly extracted triples increases by 13% (#facts corr) and the number of triples both correct and informative by 15% (#facts corr-inf). Since a few incorrect triples are inferred the overall percentage of correct triples (prec corr) drops by 2%, but at a higher recall.

An error evaluation shows that many incorrectly inferred triples are caused by wrong parses or a wrong mapping of predicates to their semantic class. Eliminating these should obviate the small negative effect on precision. A more sophisticated mapping to semantic classes could be achieved by utilizing "relational patterns" as, e.g., described in [65]. To further increase informativeness, it may also be interesting to automatically learn inference rules, as, e.g., described in [56, 84, 63]

Overall, the results show that CSD-IE achieves good precision and high recall, while providing very good coverage and minimality. We attribute this to the fact that our original motivation for this problem is an application in a semantic search engine. Hence our rules were developed and tailored with minimality in mind. The results also show that already a few simple inference rules can improve informativeness of extracted triples. Especially minimality and informativeness are important in applications like semantic search where a precise representation of facts is beneficial.

3.3 Relevance Scores for Triples

In the previous section, we have described a technique that makes semantic search in full text more precise. In this section, we show how to improve the results of structured queries on a knowledge base. In particular, we address how to rank results of queries that return a list of entities, such as *american actors*. For the queries we consider, the information required for a good ranking is not contained in the knowledge base. Therefore, we describe a variety of algorithms to compute relevance scores from a text corpus. Since this is a novel problem, we also describe a benchmark that we designed. On this benchmark, our methods achieve an agreement of about 80% with the ground truth and outperform existing methods from the literature. The benchmark and our code are publicly available via `http://ad.informatik.uni-freiburg.de/publikationen`.

This section summarizes the work published in [15] at SIGIR 2015.

3.3.1 Problem, Related Work, and Contributions

Knowledge bases are queried using a structured query language called SPARQL. A structured query has precisely defined semantics, and the corresponding result set is well-defined. For example, here is a query for *american actors* in SPARQL:[21]

```
select ?entity where {
  ?entity  has-nationality  American .
  ?entity  has-profession Actor .
}
```

This returns 64,757 matches on Freebase in no particular order.[22] Clearly, when presenting this huge result set to a user, a ranking is desirable. A straightforward ranking would be by some form of popularity. This can be measured, e.g., by the number of occurrences in a reference text corpus, which leads to the following top-5 results:

George Bush, *Hillary Clinton*, *Tim Burton*, *Lady Gaga*, *Johnny Depp*

All persons in this list are actors in the sense that they had a role in some movie. For example, George Bush appeared in the documentary Capitalism: A Love Story and Tim Burton had various cameo roles. However, in this list, only Johnny Depp is best known for being an actor. Hillary Clinton and George Bush are better known for being politicians, Lady Gaga as a musician, and Tim Burton as a film director. Consequently, one would expect Johnny Depp to be ranked before all others.

[21] This example, including results and relevance scores, is taken from our publication [15].

[22] An order can be specified explicitly, e.g., with an ORDER BY clause, but this doesn't solve the problem we consider here.

The set of *american actors* is huge, but an ordering also makes sense for smaller result sets. Consider the query for professions of Ronald Reagan in Freebase:

Actor, *Lifeguard*, *Politician*, *Radio Personality*, *Soldier*, *Spokesperson*,

All of them are correct (indeed, Reagan worked as a lifeguard in his youth), but his main professions are certainly *Politician* and *Actor*. In this case, ranking by the popularity of a profession makes no sense at all.

Queries as above are also typical queries for semantic full-text search (see Section 3.1). However, such queries are likely to appear for any kind of semantic search that utilizes a knowledge base. Hence, we consider this an interesting and relevant problem on its own that warrants a separate study.

In this section, we address the problem behind the examples above, which we defined in [15] as follows:

> **Definition:** Given a type-like relation from a knowledge base, for each triple from that relation compute a score from $[0, 1]$ that measures the degree to which the subject belongs to the respective type (expressed by the predicate and object). In the remainder, we often refer to these scores simply as *triple scores*.

Another way to describe the problem for the score of the profession *Actor* of Johnny Depp is: "how surprised would we be to see Actor in a list of professions of Johnny Depp" [15].

The desired scores for some of the entities in the queries above might look as follows:

Tim Burton	*has-profession*	*Actor*	0.3
Tim Burton	*has-profession*	*Director*	1.0
Johnny Depp	*has-profession*	*Actor*	1.0
Ronald Reagan	*has-profession*	*Actor*	0.6
Ronald Reagan	*has-profession*	*Lifeguard*	0.1

The actual score of a triple is ill-defined in our definition above. This is similar to the notion of relevance in information retrieval. There, a document is assigned a relevance score (in the case of graded relevance) given a query. It is common practice to determine document relevance via judgments from different people, simply because the actual relevance score of a document can be subjective. In our experiments (see below), we also collect feedback from multiple judges and show that there is a strong consensus on our relevance scores.

Once relevance scores are computed, it becomes straightforward to use these scores to rank the results of our example queries. In the first case (*american actors*), a ranking can

be derived by combining the scores with a popularity for each entity. In the second case ("professions of a single person"), the scores directly infer a ranking.

In this work, we consider scoring triples from *type-like relations*, such as *profession* or *nationality* (we use both in our evaluation). These present the biggest challenge in terms of ranking. For functional relations like *date of birth* or *height*, no ranking is needed or trivially achieved. For example, for *height*, simply ranking by this value will often be sufficient. Triples from non-functional relations between two concrete entities like *invested in* (between two organizations) or *featured in song* (between a musician and a song she featured in) are often better ranked by a single scalar from the knowledge base, like the value of the investment or the length of the song feature. Finding such a single scalar is difficult for type-like relations. For example, for the *profession* relation, the (computed) scalar depends on the actual profession: For a musician, it may be the number of records sold, while for an actor the appearances in high-grossing movies are more relevant.

Our motivation is ranking results of entity queries like *american actors*. Here, we focus on computing scores for individual triples in the first place. This has several reasons. First, scores as above are often all that is needed to obtain a good ranking, for example, for the query for professions of a single person. Second, our scores can serve as a crucial ingredient for approaches that rank results of entities queries. We discuss some of these approaches below. Finally, as our evaluation shows, computing good relevance scores is difficult and requires tailored approaches. These warrant a separate study.

To the best of our knowledge this is the first work addressing how to compute this kind of relevance scores for triples. There is some work on estimating scores for the correctness or accuracy (is the numeric value, e.g., a population count, off by some margin) of triples. Correctness scores can come from the system that extracted the triple, e.g., a probability provided by a machine learning classifier [58, 62]. They can also be inferred, for example, via a witness count as indicated by the number of times a given triple was extracted or found. Correctness scores are important, e.g., in knowledge fusion [38] or knowledge base construction [68, 63], where the task is to construct a consistent knowledge base from extractions from different sources. In this case, correctness scores can help resolving conflicting statements, e.g., if extracted triples disagree on the place of birth of a person. Correctness and accuracy scores are different to our relevance scores, where we assume non-conflicting facts and estimate a "degree of belonging". A beneficial side effect of our appoches is that incorrect triples are likely to get a low score.

A related line of work is that of detecting fine-grained types of entities. There, the motivation is, given a sentence and the identified mention of an entity, infer its types from a given type system. For example, from the input *Ada wrote the first computer program* and the identified named entity (underlined), it could be inferred that Ada belongs to the

person and *programmer* types of Wordnet [61]. The given input entity is not linked to a knowledge base (or does have to exist in one). Recent approaches focus on providing very fine-grained types. In [64], the authors match learned text patterns that are associated with a type and devise a probabilistic model and integer linear program to avoid type inconsistencies (e.g., a *person* cannot also have type *organization*). In [36], the authors design a huge set of patterns and rules based on, e.g., the mention's text (the mention *University of Freiburg* contains the type *university*) or verb phrases. The system is evaluated with 16k different types. In contrast to triple scores, approaches for type detection assign types in a binary fashion. Assigning Ronald Reagan the types *Lifeguard* and *Politician* to the same degree is both, correct and desired. Furthermore, approaches are designed to identify types from a taxonomy. While the *profession* relation describes part of a taxonomy, this is not the case for other type-like relations, e.g., *nationality*, which we also evaluate on. Our approaches are designed to work with all type-like relations.

For ranking results of queries on a knowledge base, a typical approach is to adapt and apply techniques from ranking in information retrieval. The adapted techniques usually rank entities based on existing scores that are assumed to be given or from ranking signals (machine learning features) that are computed from the knowledge base and query. For example, in [39], the authors investigate how to define language models for a structured SPARQL query and result graphs. The language models incorporate witness counts to estimate the confidence and importance of a triple. These are assumed to be given. In [31], the authors compare learning to rank methods for keyword queries on a knowledge base. They devise a set of 26 features that incorporate, for example, the type of an entity and the similarity between an entity and the query. Both of these example approaches could benefit from using our triples scores in addition or instead of the scores or type information they already use. More approaches for ranking results of entity queries can be found in the ranking section of our survey [16, Section 5.1].

As part of our work on this problem we have created and published a benchmark which has lead to a public challenge on triple scoring at WSDM 2017 [48] with good participation (21 teams). Most of the approaches (including the winning approach) relied on finding witnesses for each type from text and applying supervised learning methods. This is similar to one of the approaches we suggest below.

There are several challenges we face for computing triple scores. First, we cannot compute scores from the knowledge base alone. We found that the required information, especially for less popular entities, is not available reliably, even in a large knowledge base like Freebase. Therefore, we focus on computing relevance scores from text. Second, we cannot rely on purely supervised learning. There are more than 3,552 different professions in Freebase and relevance is expressed differently for each. Hence, a different model and therefore labeled examples are required for each profession. This rules out manually la-

beling examples, even with crowdsourcing. Finally, since this is a novel problem, we must also establish a ground truth that allows a realistic and reliable comparison of approaches.

In summary, our main contributions are:

- The novel and interesting research problem of triples scores for type-like relations. The scores are an essential component for properly ranking many popular kinds of entity queries. Such queries occur, for example, as part of semantic full-text search.
- A benchmark to evaluate triple scores for the *profession* and *nationality* relations. The benchmark consists of more than 14 thousand relevance judgments that were obtained via crowdsourcing. The judgments confirm that there is a broad general consensus on the problem definition of triple scores.
- A variety of approaches to computing triple scores from a text corpus. On our benchmark, our best methods perform significantly better than non-trivial baselines and achieve an agreement of about 80% with the ground truth.

3.3.2 Approach

We describe our approach that computes triple scores from a text corpus. Initially, we also experimented with using a knowledge base, either as a supplement to a text corpus or by itself. The scores we are trying to compute are certainly not explicitly expressed as facts but may be implicit in other facts. For example, if a person has acted in a lot of high-grossing movies, she is likely to be well-known as an actress, and that triple should receive a high score. However, experiments showed that this knowledge is consistently less available, especially for less popular entities.[23] Therefore, we compute triple scores from a large text corpus.

In the text corpus, we try to find "witnesses" for each triple. Optionally, each witness has an associated significance, which we also compute. The higher the count and significance of witnesses for a triple, the larger the score. For example, consider the following triples for two of *Johnny Depp*'s professions:

Johnny Depp	*has-profession*	*Actor*
Johnny Depp	*has-profession*	*Musician*

We assume that, because he is more of an *Actor* than a *Musician*, he is more likely to be mentioned along words like: *actor*, *film*, *cast* than *album*, *band*, or *singer*. The word *performed* can be associated with *Actor* as well as *Musician* and has less significance for

[23] For example, in Freebase, out of 612 thousand persons with more than one profession, 400 thousand have less than ten and 243 thousand have less than six facts (besides type and profession information).

determining these professions. By counting the number of mentions of these words (and their significance) along *Johnny Depp* in the text corpus, we can compute a score for each profession triple. Ideally, this gives a much higher score for *Actor* than for *Musician*.

This approach involves three steps. First, we need to learn *indicator words* (like *actor*, *film*, *cast* above) for each profession. Each word can be associated with a weight, depending on how significant it is for the profession. Next, for a given entity, we need to find co-occurrences with these words from which we derive an *intermediate score*. The scores should be comparable across entities. Therefore, in a final step, we *normalize* the intermediate scores to a desired output range, for example, the interval $[0, 1]$.

We designed multiple variants of how to learn indicator words, how to use them to derive intermediate scores, and how to normalize to the final output range. We shortly summarize these in the following. Note that we use *profession* as an example relation, but our approaches work for any type-like relation.

Our first step to learning indicator words is to derive positive and negative training examples for each profession. This is done in a weakly supervised fashion using the following criteria: For a given profession, the positive training examples are persons who *only* have that profession (according to the knowledge base). The intuition is that these triples naturally have the highest score assigned. Correspondingly, negative training examples are persons who don't have the given profession at all. Clearly, these should have the lowest score. For example, Marlon Brando is a positive example for the profession *Actor*, because this is his only profession according to Freebase. Alan Turing would be a negative example, because he is a *Computer Scientist*, *Mathematician*, and more, but not an *Actor*.

Next, we represent each person with a *virtual document*. This document consists of all the words the person co-occurs with. To identify semantically co-occurring words we use contextual sentence decomposition as described in Section 3.2. The virtual document of a person now consists of all (of the words) of the contexts she appears in. In our experiment, this gave better results than using whole sentences.

We now have a set of positive and negative examples (virtual documents of persons) for each profession. From here, we consider multiple variants of learning indicator words and deriving intermediate scores. We shortly mention the basic ideas, in our experiments we also considered many slight variations:

- *Binary classifier*: Learn a classifier (logistic regression) that makes a binary decision for each profession. Each person is represented by the bag-of-words of her virtual document with words weighted by tf-idf. This way, the classifier will learn a weight for each word. The intermediate score for a profession is the binary decision of the classifier whether the profession is primary or not.

- *Count profession words*: First, identify indicator words. A simple variant is to use manually chosen prefixes of the profession, such as *act* for *Actor*. This as a baseline in our experiments. A more complex variant is to compute and sum tf-idf scores for words of the positive examples, sort by that score, and assign a weight of $1/r$ for word at rank r. This gives weighted words for each profession. To get intermediate scores, for each profession, count the prefix matches or sum up the word weights in the virtual document of a person.
- *Generative model*: Identify weighted indicator words for each profession as with the counting-based approach. Then apply a model, which, intuitively, generates the text in the virtual document of a person. Maximizing the likelihood of the person's text gives a probability distribution over her professions as intermediate scores.

The approaches output different types of intermediate scores for each triple: word counts, weighted sums, binary judgments (logistic regression), or probabilities. In the final step, we normalize these for each person by mapping to the desired output range. In our experiments, we map to the integers $[0, 7]$ in order to facilitate comparison with our crowdsourcing results. We either map to these scores on a linear scale or on a logarithmic scale, where the next highest score corresponds to twice the intermediate score (sum or probability).

3.3.3 Experiments and Results

Since this is a novel problem, we started by creating a new benchmark. To obtain a large number of relevance judgments we used crowdsourcing. Each human judge is given a description of the task. In general, the judges aren't familiar with the problem and a major challenge was describing the task in way that results in consistent and reliable relevance scores. This gave us valuable insight into our problem and its definition, also allowing us to verify whether there is a general consensus on our scores. We first describe our crowdsourcing task, followed by results on the new benchmark.

Crowdsourcing Benchmark

We experimented with several different crowdsourcing tasks, refining the task definition and description along the way. Figure 4 shows the final task that worked well.

Figure 4. Adapted from our publication [15]: Excerpt of the crowdsourcing task for Ludwig van Beethoven. His professions must be classified into *primary* or *secondary* by dragging each profession into the respective box. The Wikipedia link to his page can be used to find additional information (important for lesser known persons). In addition, example classifications and further instructions were provided (not shown here).

Judges must classify all of the professions of a single person into either *primary* or *secondary*. This worked better than asking for the label of a single profession of a person (without showing the other professions). Judges then often simply labeled the first mentioned profession of a person in the Wikipedia article as primary. We used this strategy as one of the (simple) baselines in our experiments.

Each person is judged seven times by different judges. The number of primary judgments for each profession of a person is aggregated to give a score from 0 to 7. For the profession relation, we ran the above task for a random selection of 298 persons (1225 triples) which gave us 8575 judgments. The random selection takes into account the popularity (i.e. the number of mentions in our text corpus) of persons, providing a fair selection across

all popularity levels. We also evaluated the nationality relation, for which we selected 77 different people (162 triples) resulting in 1134 judgments.

To verify inter-annotator agreement we performed a *control* run of the above experiment (with different judges) on one-third of the persons selected for the profession relation. We also created *random* judgments choosing primary or secondary with equal probability.

Figure 5 shows a histogram of the results. As can be seen, judgments are far from random and the control run provided very similar results.. This, together with a Fleiss' Kappa [44] of 0.47, shows that there is broad general consensus on the problem.

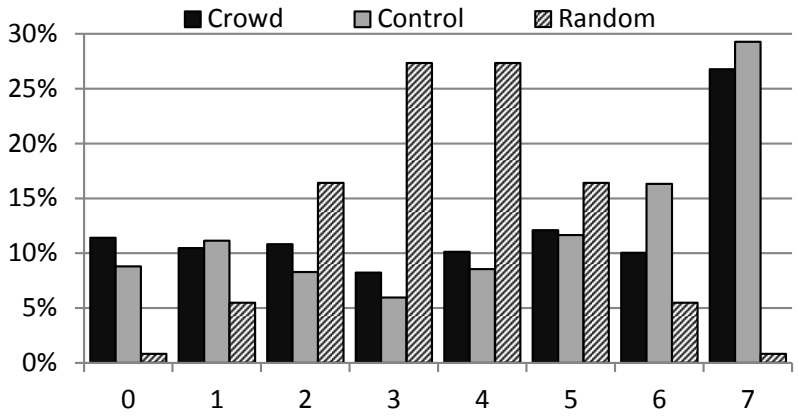

Figure 5. From our publication [15]: Histogram of score distribution of our crowdsourcing task, the control run, and expected results for randomly (with $p = 0.5$) guessing judges (rounded).

Main Results

We evaluated our approaches using the above benchmark. As a text corpus, we used the English Wikipedia where we recognized entity mentions as described in Section 3.1. As a knowledge base, we used Freebase.

We performed experiments for the profession and the nationality relation. Experiments on the nationality relation gave no major additional insights, hence we only describe the main results for the profession relation here.

We compared our variants against several baselines. The *first* baseline selects the first literal occurrence of a profession in the person's textual description of Freebase to be primary. The *prefixes* baseline applies the counting approach from above using prefixes that were manually derived from the profession names (e.g., *act* for *Actor*). The *Labeled Latent Dirichlet Allocation (LLDA)* [71] baseline is a topic model that can be learned in a supervised fashion using our weakly-supervised training data. Given the text associated with a person, it estimates a probability distribution over topics (e.g., professions).

All scores were mapped to the range 0 to 7, which is the range we get from our crowdsourcing experiments. To map our intermediate scores to this range, we used the mapping that gave best results: linear mapping for count-based approaches and logarithmic mapping for approaches that output probabilities. As main evaluation measures, we computed accuracy-δ, which measures the percentage of triples that deviate by at most δ from the ground truth score, and average score deviation, the average over all deviations. We also computed rank based measures: Kendalls' Tau, footrule distance, and nDCG. These correlated well with accuracy-δ and we omit them for brevity.

Method	Accuracy (Acc)			Average Score Diff
	$\delta = 1$	$\delta = 2$	$\delta = 4$	
First	41 %	**53 %**	71 %	2.71
Prefixes	50 %	**64 %**	83 %	2.07
LLDA	50 %	**68 %**	89 %	1.86
Binary Classifier	47 %	**61 %**	78 %	2.09
Weighted Indicators	57 %	**75 %**	94 %	1.61
Generative Model	57 %	**77 %**	95 %	1.61
Combined (GM + Classifier)	63 %	**80 %**	96 %	1.57
Control Judges	76 %	**94 %**	99 %	0.92

Table 5. Adapted from our publication [15]: Accuracies and average score deviation for the *profession* relation. The top part shows baselines, the middle part our approaches, and the bottom part our control run.

Table 5 summarizes the results for the profession relation. We consider accuracy-2 the most intuitive measure (percentage of triples with difference to ground truth score at most two), hence it is emphasized. Note that accuracy-2 can be optimized by truncating results to range between 2 and 5, which is never worse than predicting more extreme values. However, this goes at the cost of other measures like average score difference [48]. We performed no such truncation and our scores correlate well with average score difference.

The baselines perform well, but our more sophisticated approaches clearly outperform them. Our best approach consists of a combination of the generative model and the binary classifier (use the average of the two predicted scores if the binary classifier predicts score 7). It achieves performance not far from human judges with a gap of 14%. For all approaches we also evaluated many variants, including: using word pairs instead of single words, using stemming, utilizing the knowledge base type hierarchy, other non-linear score mappings, using tf instead of tf-idf etc. None of these improved results.

An evaluation of errors showed two main sources. The first source of errors is word co-occurrences that refer to the wrong entity. For example, Michael Jackson gets a high score as a *Film Director* because he is often mentioned along *directed* or *director*, which however, does not refer to him, but to a person directing one of his shows.[24] Using co-occurrence based on our context decomposition instead of full sentences helps in some, but not all of these cases. The second source of errors are words that occur with a different meaning. For example, John F. Kennedy gets a high score as a *Military Officer*, simply because he had many political actions during his presidency with respect to the military. In some of these cases, a deeper linguistic processing of text as, e.g., in fine-grained entity typing [64, 36] may help. In general, it appears that a deep natural language understanding will be necessary to close the gap to human judges.

[24] This and the following error example are taken from our publication [15].

3.4 Question Answering on Knowledge Bases

The previous section was concerned with making knowledge base search more precise. In this section, we address how to make the search more convenient. We present an approach that automatically translates a natural language question into a structured query on a knowledge base. This makes searching the knowledge base easy by allowing the user to freely formulate her questions in natural language, without knowing about a query language or the structure of the underlying data. In addition to providing precise translations, we also consider efficiency. We make sure that questions can be answered in an interactive manner within at most one second. At the time of publication, our approach improved the state of the art in terms of quality and efficiency on two large established benchmarks.

This section summarizes the work published in [17] at CIKM 2015. In the following, the examples are taken from this paper.

3.4.1 Problem, Related Work, and Contributions

Mapping a search desire to the corresponding structured query on a knowledge base can be difficult, even for an expert. For example, consider answering the (seemingly) simple question *who is the ceo of apple?* on the knowledge base Freebase.[25] The SPARQL query to retrieve the answer is:

```
select ?name where {
  ?0 leadership.role Managing_Director .
  ?0 leadership.company Apple_Inc .
  ?0 leadership.person ?name
}
```

This matches the following triples in the knowledge base:[26]

m.0k8z	leadership.role	Managing_Director
m.0k8z	leadership.company	Apple_Inc
m.0k8z	leadership.person	Tim_Cook

The triples contain an abstract "leadership" entity (*m.0k8z*) with three relations: *role*, *company*, and *person*. The relations and the leadership entity connect the entities *Managing_Director* and *Apple_Inc* to the answer *Tim_Cook* we are looking for.

[25] This query is from the WebQuestions benchmark [20] which we use in our evaluation. For illustration purposes, we omit namespace prefixes and use readable identifiers in the corresponding SPARQL query.

[26] We have updated the triples with the current CEO. The original Freebase data still mentions Steve Jobs. Freebase is being migrated to Wikidata [82], which is constantly updated with new information.

To construct the query above, a user would need to find the identifiers of the correct entities and relations and connect them in the correct way. This can be facilitated by an interactive construction guided by suggestions like our semantic full-text search prototype provides (see Section 3.1). However, this still passes some of the complexity to the user. It would be far more convenient to ask the question in natural language and automatically get the structured query that produces the answer. This is the problem we address here.

Answering natural language questions is a difficult task. Here, we focus on answering questions from a knowledge base, albeit a very large one. As we show in our evaluation in this section, this is hard enough to warrant a separate study. Our survey [16, Section 4.6, 4.7, 4.8] provides an overview of the state of the art on question answering on text, knowledge bases, and the few works that attempt a combination.

The challenge in answering natural language questions stems from the high variation and ambiguity inherent in natural language. This becomes apparent when looking at two important subproblems of the translation process: identifying knowledge base entities and identifying knowledge base relations that are mentioned in the question. If these subproblems were solved perfectly, the correct query would be trivial to infer in most cases. However, matching entities is difficult because there are many ways of mentioning a specific entity (synonymy) but a single mention might also refer to many different entities (polysemy). For example, the mention of *apple* in the query above refers to 218 entities in Freebase but only one entity, *Apple_Inc*, is actually correct. The same problem holds for matching relations but is even harder. It can involve n-ary relations and some of the relations might only be mentioned implicitly. For example, the question above contains no word or synonym of the query's relation names *person*, *company*, or *role*. This problem becomes even harder with the size of the knowledge base. Allowing weak and fuzzy matches drastically increases the search space. On the other hand, only allowing strict and lexical matches misses many correct matches.

Our goal is to answer questions from a large knowledge base like Freebase, which contains about 2.9 billion facts on 44 million entities. To keep the problem manageable, we focus on "structurally simple" queries. These involve two or three entities and either a single binary or n-ary ($n > 2$) relation. In the example above, the result entity *Tim_Cook* is connected to the entities *Apple_Inc* and *Managing_Director* via an n-ary leadership relation. In SPARQL, this n-ary relation is represented using several binary relations, and the special leadership entity (*m.0k8z*) also referred to as *mediator* in Freebase. This is a typical way of representing n-ary relations for $n > 2$ in triple stores.[27] While this may sound restrictive, it still allows answering a wide variety of questions, as can be seen from our evaluation.

[27] More generally, one can represent a k-ary relation by a special entity (one for each k-tuple in the relation) and k binary relations between the special entity and the k entities of the tuple.

Given that the involved problems are mainly concerned with natural language, it is unsurprising that a lot of recent work on question answering on knowledge bases has come from the natural language community. There, it is addressed as part of *semantic parsing*: translating the meaning of a sentence (or in this case, question) to a formal representation of its meaning (in this case, the SPARQL query)

Recent work can be roughly categorized into two groups. The first group only considers a fixed set of query structures as we do, e.g., [86, 85, 25, 22, 83, 88]. This puts the focus on the hard problems of matching relations and entities in the question. The second group of approaches allows a translation to (in theory) arbitrary queries, e.g., [30, 20, 53, 21, 1]. These systems either rely on a pre-trained syntactic parser or, essentially, are parsers by themselves. This increases complexity and introduces new problems. In particular, the huge space of possible patterns must be searched efficiently. Note, however, that the two approaches are complementary. Any system will have to deal with the problem of matching relations and entities. In contrast to our approach, previous approaches lack a proper addressing of the entity matching problem and either assume matching entities are given or only perform very simplistic matching. A very recent system that is close to ours is [1]. It follows our approach in applying a learning-to-rank strategy and focuses on answering more complex queries. More details on related approaches and systems can be found in our publication [17] and survey on semantic search [16, Section 4.7].

Besides quality, we also consider efficiency aspects, which is neglected in previous work. In particular, we took care that questions can be answered in an interactive manner, that is, within a second. Furthermore, we made sure that our approach works well on multiple datasets (with different kind of questions). A lot of previous approaches were evaluated only on one dataset.

Our main contributions are the following:

- A new end-to-end system that translates a natural language question into a SPARQL query on a knowledge base. In contrast to previous work, the system treats entity identification in the question as an integral part of the problem and also considers efficiency aspects.
- A joint disambiguation of entities and relations mentioned in the question. This is achieved by applying learning-to-rank techniques to learn a pairwise comparator of query candidates. Previous approaches framed this ranking as a classification or parsing problem.

- An improvement over the state of the art (at the time of publication) on two established benchmarks in terms of quality and efficiency. The two benchmarks exhibit quite different challenges. Many previous systems were evaluated and work well on only one of the benchmarks.

3.4.2 Approach

In the following, we describe how to answer natural language questions from a knowledge base. We use Freebase for our examples and in our evaluation. However, our approach is not specific to Freebase, but works for any knowledge base with entities and possibly n-ary relations between them.

On a high level, the approach can be split into four steps, which we explain in the following. As a running example, assume we are trying to answer the following question (from the WebQuestions benchmark):

what character does ellen play in finding nemo?

Entity identification. We begin by identifying entities from the knowledge base that are mentioned in the question. In our example, *ellen* refers to the tv host *Ellen DeGeneres* and *finding nemo* refers to the movie *Finding Nemo*. However, these words are ambiguous, for example, *ellen* could also refer to the actor *Ellen Page* and *finding nemo* to a video game with the same name.

Often, entities are not mentioned with their full knowledge base name, but with a synonym or alias. To find these, we utilize CrossWikis [78], which was built by mining the anchor text of links to Wikipedia entities (articles) from various large web crawls. CrossWikis covers around 4 million Wikipedia entities and provides prior probability distributions $p(e|s)$, of entity e being mentioned by text span s. We extend these distributions with information about Freebase entities (details are in our publication [17]). This way, we are able to recognize around 44 million entities with about 60 million aliases. In order to keep the number of matched entities as small as possible without affecting recall we apply part-of-speech (POS) tagging to the question and restrict matches to reasonable POS sequences (mainly nouns).

Note that the disambiguation above can be helped by facts from the knowledge base. In our example, the fact that *Ellen DeGeneres* actually performed in the movie *Finding Nemo* makes the joint mentioning of both entities more likely. Therefore, instead of fixing the mentioned entities at this point, we delay the decision and jointly disambiguate entities and relations in a later step. The result of this step is a set of entity mentions $p(e|s)$ with attached probabilities. Subsequent steps also make use of a popularity score of each entity.

Template matching. Next, we match a set of query templates to the question using the previously identified entities. Figure 6 shows our templates. Each template consists of entity and relation placeholders. A matched template has these placeholders filled and corresponds to a *query candidate*, which can be executed against the knowledge base to obtain an answer.

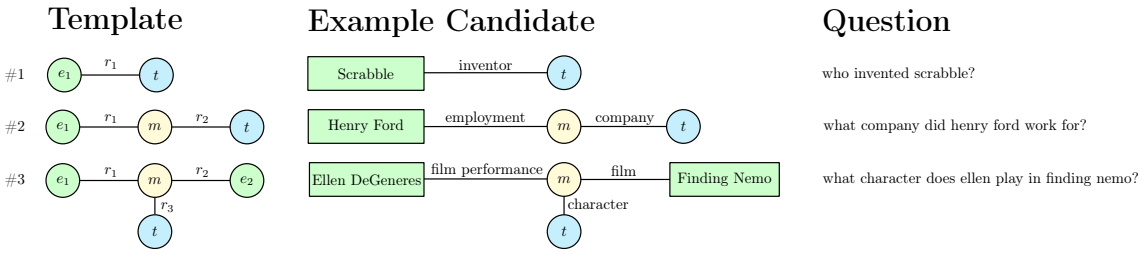

Figure 6. From our publication [17]: Query templates and example candidates with corresponding questions. A query template contains entity placeholders e_i, relation placeholders r_i, a mediator m and an answer node t.

Conceptually, we match the templates as follows. Let E be the set of all entities matched to a subsequence of the question in the previous step. Consequently, a word in the question can be part of several entity matches. For each template, fill the entity placeholder(s) with entities $e_i \in E$ for which the words in the question don't overlap. Then, look up relations R_i for each entity e_i in the knowledge base and create a query candidate for each relation. If multiple entities are part of the template, the relations must connect them in the knowledge base. This avoids generation of query candidates that have no answer.

For our example, we match template #3, besides others, as follows. We map e_1 and e_2 to *Ellen DeGeneres* and *Finding Nemo*, respectively. Then we look up relations r_1 as *film performance* and r_2 as *film*. These connect the two entities in the knowledge base because Ellen DeGeneres played in Finding Nemo. One of the possible relations for r_3 is *character*, which generates the correct candidate shown in Figure 6 (center of bottom row).

Technically, this matching process is more involved to perform efficiently. In particular, for matching template #3, we need to find r_1 and r_2 in connected triples (e_1, r_1, m), (m, r_2, e_2) given e_1 and e_2. Such queries are slow in current triple stores.[28] We, therefore, constructed a special inverted index for faster lookup. Details can be found in our publication [17].

In this step, we favor recall over precision and generate a lot of query candidates, most of them wrong. Wrong candidates can be identified in a later step, but a missing correct candidate will lead to a wrong final answer. For our example question, we generate 356 candidates, only one of which of is correct. The final result of this step is the set of all generated query candidates.

[28] In our experiments we use Virtuoso: `http://virtuoso.openlinksw.com`.

Relation matching. The query candidates still miss the fundamental information about which relations were actually mentioned in the question. We distinguish four ways of matching relations of a query candidate to the question text:

- *Relation name*: We match the name of the relation to words in the question. We distinguish between literal matches (e.g., the relation named *character* matches the word *character*), synonyms based on word embeddings and cosine distance [60] (e.g., *started* matches *founded*), and word derivations extracted from Wordnet [61] (e.g., *high* matches *elevation*).
- *Distant supervision* [62]: We learn indicator words for each relation using text from Wikipedia where entity mentions were identified. This allows deriving noisy training examples: A sentence expresses relation r if it contains two co-occurring entities that are in relation r according to a knowledge base. For each relation, we rank the words by their tf-idf to learn, for example, that *born* is a good indicator for the relation *place of birth*.
- *Supervised learning*: Our evaluation benchmarks come with a training set of questions and corresponding answer entities. We can use this to derive positive and negative training examples: Generate all query candidates for a question; if a query candidate answers the question it is a positive, otherwise, a negative example. As indicator features, we concatenate the n-grams of the question with the relation name. This way we can learn a logistic regression classifier.
- *Deep (supervised) learning*: Using the same supervised data, we can train a deep neural network instead of a logistic regression classifier. This is an extension of our work and not described in [17]. We provide technical details below (Section 3.4.4). In our experiments, we explicitly state when we use the neural network.

Each of the techniques provides a confidence score that we use for ranking candidates.

In our example, a word learned using distant supervision for the relations *film performance* and *film* (between an actor and the film she acted in), is *play*. Furthermore, the word *character* matches the relation with the same name.

Ranking. We now have a set of query candidates with information about which entities and relations match which parts of the question how well. In a final step, we rank these to find the best matching candidate. Note that performing ranking at this final step has the strong benefit of jointly disambiguating entities and relations. A candidate can have a weak match for an entity, but a strong match for a relation, and vice versa. We can identify these combinations as correct, even when one of the matches seems unlikely when considered separately.

To rank the candidates, we apply learning to rank [57]. The training data from the benchmarks allows sorting the generated candidates for each question by how well they answer it. Using the ranked list of candidates, we learn a comparator (a random forest [26]) that, given two candidates, decides which should be ranked first. We engineer a total of 23 features for this. These indicate how well entities and relations match but also include more general features like coverage (ratio of question words matched by an entity or relation) or result size. The exact features are listed in our publication [17].

For our example above, the candidate covering most words of the question is best. Matching *ellen* to *Ellen Page* no longer allows matching *Finding Nemo* because these aren't related in the knowledge base. On the other hand, the *character* relation matches the word *character* in the question, which is not the case for alternative relations like *performance type*. This leaves us with the correct interpretation of asking for *Ellen DeGeneres*' character in *Finding Nemo*.

3.4.3 Experiments and Results

To evaluate our system, we used two established benchmarks: *Free917* [30] and *WebQuestions* [20]. Each benchmark consists of a set of questions and their answers (one or more entities) from Freebase. As our knowledge base, we used the original Freebase data and not the curated version we described in Section 3.1. This has two reasons. First, we want to use all available facts in our answering process, and our curation simplifies n-ary relations (like the film performance or leadership relations above). Second, and more importantly, this allows a fair comparison with other approaches.

Each benchmark comes with a pre-defined set of training and test questions. The two benchmarks differ substantially in the types of questions and their complexity.

Free917 contains 917 manually generated natural language questions. The questions cover a wide range of domains. Two examples are:

who won the 1964 united states presidential election?	answer: *Lyndon B. Johnson*
how many languages has jrr tolkein created?[29]	answer: *10*

Questions are mostly grammatical and tend to be tailored to Freebase. The benchmark also provides an *entity lexicon*: a manually constructed mapping from text (the surface form) to the mentioned entity. This was used for identifying entities by all systems reporting results on the dataset so far. We only make use of this lexicon where explicitly stated. The established evaluation measure on this benchmark is *accuracy*, the fraction of queries answered with the exact gold answer.

[29] The typo in *tolkien* is indeed part of the dataset.

WebQuestions consists of 5,810 questions that were selected by crawling the Google suggest API. Contrary to Free917, questions are not necessarily grammatical and are more colloquial. For example:

who brad pitt has dated? answer: *Angelina Jolie, Jennifer Anniston, ...*
who plays dwight in the office? answer: *Rainn Wilson*

Due to the selection process, questions are biased towards topics that are frequently asked from Google. Furthermore, the structure of questions tends to be simpler. Answers to the questions were obtained by using crowdsourcing. This introduces additional noise. In particular, for some questions, only a subset of the correct answer is provided as gold answer. Therefore, what is usually reported is *average F1*: the F1 measure for each question (obtained by comparing the gold answer set and the system's answer set of entities) averaged over all questions.

Table 6 compares the quality of our system, Aqqu, to recent systems:

		Free917		WebQuestions
Method	Venue	Accuracy+	Accuracy	Average F1
Cai et al. [30]	ACL'13	59 %	–	–
Jacana [86]	ACL'14	–	–	35.4 %
Sempre [20]	EMNLP'13	62 %	52 %	35.7 %
Kwiat. et al. [53]	EMNLP'13	68 %	–	–
Bordes et al. [25]	EMNLP'14	–	–	39.2 %
ParaSempre [22]	ACL'14	68.5 %	46 %	39.9 %
Aqqu [17]	CIKM'15	**76.4 %**	**65.9 %**	**49.4 %**
STAGG [88]	ACL'15	–	–	52.5 %
Berant et al. [21]	TACL'15	–	–	49.7 %
Xu et al. [83]	CoRR'16	–	–	53.3 %
Reddy et al. [73]	TACL'16	78.0 %	–	50.3 %
QUINT [1]	WWW'17	72.8 %	–	51.0 %
Aqqu + NN	–	**78.7 %**	**70.2 %**	**51.8 %**

Table 6. Results on the Free917 (267 questions) and WebQuestions (2032 questions) test set. Results of our system, Aqqu, in bold. The bottom part of the table corresponds to work published after Aqqu [17]. For the results in the third column (Accuracy+), a manually crafted entity lexicon was used. Note that better performing systems fundamentally relied on external data (see text).

Most of the systems in Table 6 have only been evaluated on one of the two benchmarks. Our system, *Aqqu*, uses a single approach that considerably improved the state of the art on both benchmarks at the time of publication. The extension with a deep neural network for relation matching, *Aqqu + NN* (see Section 3.4.4), is also competitive for the current state of the art, in particular, since other approaches make considerably more use of external data than our rather shallow learning of indicator words via distant supervision (see Section 3.4.2). Xu et al. [83] issue queries against a full-text search engine on Wikipedia during the answering process. STAGG [88] uses large amounts of additional data (derived via distant supervision) to train their neural network. Without this external data, [83] report a drop of 6.2% and [88] a drop of 0.9% in average F1. Both techniques are likely to benefit our approach as well.

To inspect how many questions can be answered by our query templates we analysed oracle results: the score achievable when assuming perfect ranking. On Free917 *Aqqu* achieves an oracle accuracy of 85% (without manual entity identification) and on WebQuestions an oracle average F1 of 68%. Note that, as explained above, the WebQuestions dataset consists of frequent but very noisy questions that often have an incomplete ground truth. In a manual inspection we estimate that the best achievable score is around 80%. This shows that our patterns allow answering the majority of questions with high quality.

Since we focused on rather short and frequent questions our templates will not work well for arbitrarily complex questions. The work in [1] shows that templates can also be learned automatically in order to answer compositional questions such as *Which were the alma maters of the PR managers of Hillary Clinton?*. The described approach is orthogonal to ours and could be integrated on top of our current system. We consider this direction worthwhile future work.

On both benchmarks we also look at the top-k results of *Aqqu*. The best candidate is within the top two in 74% of questions for Free917 and 67% for WebQuestions. This demonstrates that our learned ranking is very strong. Indeed, in many cases, the top two candidates are hard to distinguish and often match the question very well. For example, *where is chris paul from?* can be answered with his place of birth or his nationality, but only one interpretation is considered correct in the ground truth.

Besides quality, we also evaluated efficiency of our system. On average, *Aqqu* answers a question within 217ms and 143ms on the test sets of Free917 and WebQuestions, respectively.[30] *Aqqu + NN* requires 193ms and 169ms, respectively. Berant et al. [21] report 291ms per question on average for WebQuestions. For other systems that provide code and for which we reproduced results, run times are (at least) several seconds per query.

In an error analysis, we found that there is no single large source of errors worth pointing out. Instead, each of the components (entity recognition, pattern and relation matching, ranking) fails about equally often for various reasons. The accompanying materials of our publication [17] provide a list of errors. Our publication also includes a more detailed evaluation and analyses of, for example, feature and component importances.

3.4.4 Matching Relations in Questions Using Deep Learning

We shortly describe the neural network used in *Aqqu + NN* above. This is an extension that is not part of our original publication [17]. The neural network computes a score that indicates how well a candidate query and its relations matches a question. Essentially, it is an additional, more sophisticated approach of the supervised relation matching classifier. The neural network score is used in ranking query candidates. Besides this additional ranking feature, all other components of the system are identical.

The architecture of the neural network is similar to that of [88]: a Siamese neural network [27] that learns a real-valued vector representation (embedding) of the question and knowledge base query. The cosine between both vector representations is used as a score on how well the question matches (translates to) the query. Ideally, the representation of the correct query is close to the representation of the question and the resulting cosine similarity is large.

To compute an embedding of the question, we first transform it into a sequence of 128-dimensional word vectors. Word vectors are learned using word2vec [60] on a corpus of 50 million sentences extracted from ClueWeb [34]. The vectors are kept fixed during training of the neural network, i.e., we perform no fine-tuning. On the sequence of vectors we apply a 1-d convolution followed by max-pooling as described in [50]. The convolution uses filters of size one, two, and three with 300 filters of each size. The output of the convolution layer is followed by a fully connected feed-forward network with 200 nodes.

[30] These numbers differ from our original publication [17] because we have improved the implementation in obvious ways, e.g., by avoiding unnecessary re-computations of features. Originally we reported 644ms and 900ms, respectively. New experiments were performed on a system with Intel i7-6700 CPUs, 60GB of RAM, and a Titan X GPU.

To compute an embedding of a knowledge base query, we split each involved relation into its three inherent parts separated by a dot. Each part is then represented as the average of its word vectors (some parts have several words like *place_of_birth* in *people.person.place_of_birth*). We use the same word vectors as above that are also fixed during training. Because a query can have up to three relations (see the templates in Section 3.4.2) this results in nine 128-dimensional vectors. Their concatenation is passed through two fully connected feed-forward layers with 200 nodes each.

All nodes of the network use exponential linear units [33] as activation function. The 200-dimensional outputs for the question and query representation are then compared using their cosine, which gives the final output of the network.

The training data is the same as for the supervised relation matching classifier, i.e., it is constructed from the training questions of the benchmark. We train the network using Adam [51] for a fixed duration of 30 epochs, determined after observing performance on a development set. As loss function, we use mean squared error.

We also experimented with many variations and extensions: using dropout for regularization, different activation functions, different loss functions, and optimizers. In our experiments, none gave performance improvements on the development set over the network described above.

3.5 Semantic Search Survey

In this section, we describe our extensive survey (156 pages) on semantic search on text and knowledge bases [16]. We start by illustrating our motivation for the survey followed by giving a short outline.

As we stated in the introduction, semantic search is not a single well-defined problem. Rather, it is understood by many different communities in different ways. As a result, researchers are often not aware of related work in other communities, though the addressed problems are similar. This is the main motivation behind the survey. It should give an extensive overview of addressed problems, their approaches, and state of the art in the different communities. Crucially, no such overview was available in the literature. Our survey categorizes the vast research field. It also explains basic techniques which underpin many of the described approaches and provides details on advanced techniques. This should make it useful to newcomers as well as seasoned researchers.

To categorize the research, we devised a classification scheme along two dimensions: the *type of data* that is searched on and the *search paradigm*. We consider this classification scheme a major contribution. It can serve as a guideline to navigate the field, categorize new approaches and techniques, and find related work in that area. Figure 7 shows the categories according to our classification scheme. We shortly describe each of the dimensions.

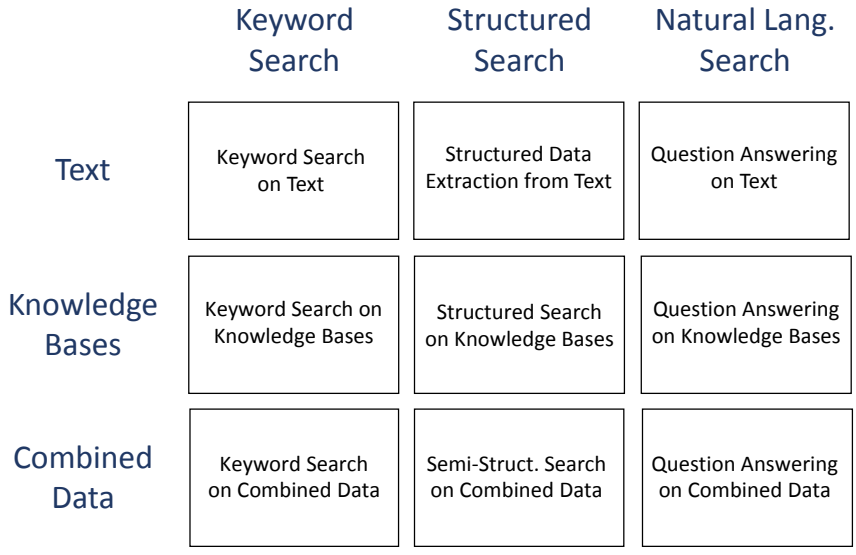

Figure 7. Adapted from our survey [16]: Our basic classification of research on semantic search into categories by underlying data (rows) and search paradigm (columns). Each category corresponds to a subsection in the survey.

Data in the form of *text* usually consists of a collection of documents that contain natural language. This is the most abundant kind of information available, but completely unstructured. A typical example is the Web, where each web page corresponds to a document. *Knowledge bases*, on the other hand, contain structured statements, often in the form of *subject predicate object* triples. An important trait is that identifiers of entities and relations are used consistently, that is, the same entity or relation should have the same identifier in all triples. A typical example of a knowledge base is Freebase. *Combined data* refers to a combination of the two previous types. Entities from a knowledge base can be linked to their mentions in text. This is the kind of data semantic full-text search (see Section 3.1) uses. Combined data also refers the case where several knowledge bases (with different naming schemes) are combined into one huge interlinked knowledge base. In this case, the same entity or relation can exist multiple times with different identifiers. A typical example is *linked open data*, the data behind the Semantic Web.

The classification into our three search paradigms is as follows. *Keyword search* accepts a list of (typically few) keywords. It places no restriction on the structure of the query. This is still the best known and most ubiquitous search paradigm. *Structured search* uses a query language, like SQL or SPARQL. Queries must adhere to the syntax defined by the language. This is the obvious choice when the data to be searched is structured. The language can also be extended to incorporate search in unstructured data, for example with keyword search in text, as in semantic full-text search (Section 3.1). *Natural language search* answers complete questions as a human would pose them. We presented an approach for question answering on a knowledge base in this document in Section 3.4.

Our classification leads to a total of nine categories. Each category corresponds to a subsection that contains: a profile of the corresponding line of research (including strengths and limitations) and a description of the basic techniques, important systems, benchmarks, and the state of the art. These nine categories form the core section of the survey.

In addition, our survey provides a lot of supplementary information, for example, we list and reference important datasets for each category. We also provide introductory and background information, for example, we dedicate a whole section to fundamental natural language processing techniques. In a final section on advanced techniques, we address ranking, indexing, ontology matching and merging, as well as inference.

Altogether, we believe the survey is a valuable contribution to the semantic search community that was missing and long overdue, especially since the field is so vast, hard to delineate, and difficult to get an overview of.

4 Future Work

The ultimate semantic search engine is one that has human-like understanding of the world. It is able to perfectly determine the intent of each query, whether given as keywords or complete question. And it can reason about the answer using whatever data it has at its disposal, much like a human. Such a level of understanding and where it might come from is still elusive. Nonetheless, we consider the following three directions of future research as promising steps (albeit comparably small) towards this ultimate goal.

First, extending the question answering approach presented in Section 3.4.2 to use full text in addition to a knowledge base will allow answering a wider range of questions. The obvious way to achieve this is to make use of the *occurs-with* relation from semantic full-text search (see Section 3.1). The main challenge will be to determine when to make use of the text in addition to the knowledge base. For example, for questions like *who is the ceo of apple?* results could always be kept up-to-date with a combined search in full text and a knowledge base. However, searching full text is not necessary and possibly detrimental in other cases, for example, for *who founded apple?*, which is answered perfectly from the knowledge base alone.

Second, using semantic completions may direct a user to formulate questions that are less ambiguous and better understood. For example, after typing "*who played rick deckard in*" the user can be presented suggestions for the different Blade Runner movies. If she selects one of the suggestions, no ambiguity arises compared to typing *blade runner*, which can mean the movie from 1982 or 2017. This requires that the question typed so far is sufficiently understood, for example, that the movie character Rick Deckard has been identified and that the list of suggestions contains movies in which he appears. Suggestions could be provided for all words of the question, not only for entities. In particular, words that describe relations between entities, such as *played* above, are a major source of ambiguity (see Section 3.4) and could be suggested as well. There is a large body of work on auto-completing keyword queries summarized in [29]. However, only a few works make use of entities or knowledge bases, and no work seems to address the problem for question answering. Preliminary experiments indicate that a class-based language model can learn meaningful suggestions. There is also recent work showing how knowledge base facts can be incorporated in a neural language model [2].

Finally, it will be interesting to apply recent advances in learning neural networks in an end-to-end fashion to answer questions [55, 67, 66]. This has the potential to enable search on multiple data sources by learning which data source is reliable for which kind of information. For example, it may be possible to learn when to query text in addition to a knowledge base as in the example above. However, to achieve good results, large amounts of training data may be required, and to answer a wide range of questions, it will

be necessary to transfer the learnings from one domain or dataset to others - which is still an open research problem.

The problems we have addressed in this thesis have a wide range of applications in semantic search. In particular, they can serve as building blocks in the suggestions outlined above. Together, this represents a promising direction of future work which may bring us a step closer to the ultimate semantic search engine.

References

The following lists the references cited in this summary. Note that the list of references accumulated in all publications given in Section 2, especially the survey (see Section 3.5), is much larger. In the digital version of this dissertation, titles below are clickable and link to an electronic copy.

[1] A. Abujabal, M. Yahya, M. Riedewald, and G. Weikum. "Automated Template Generation for Question Answering over Knowledge Graphs". In: *WWW*, 2017, pp. 1191–1200.

[2] S. Ahn, H. Choi, T. Pärnamaa, and Y. Bengio. "A Neural Knowledge Language Model". In: *CoRR*, 2016.

[3] P. Bailey, R. W. White, H. Liu, and G. Kumaran. "Mining Historic Query Trails to Label Long and Rare Search Engine Queries". In: *TWEB*, 2010, 15:1–15:27.

[4] K. Balog, P. Serdyukov, and A. P. de Vries. "Overview of the TREC 2010 Entity Track". In: *TREC*, 2010.

[5] K. Balog, P. Serdyukov, and A. P. de Vries. "Overview of the TREC 2011 Entity Track". In: *TREC*, 2011.

[6] K. Balog, A. P. de Vries, P. Serdyukov, P. Thomas, and T. Westerveld. "Overview of the TREC 2009 Entity Track". In: *TREC*, 2009.

[7] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. "Open Information Extraction from the Web". In: *IJCAI*, 2007, pp. 2670–2676.

[8] H. Bast, A. Chitea, F. M. Suchanek, and I. Weber. "ESTER: Efficient Search on Text, Entities, and Relations". In: *SIGIR*, 2007, pp. 671–678.

[9] H. Bast, F. Bäurle, B. Buchhold, and E. Haußmann. "A Case for Semantic Full-Text Search". In: *SIGIR-JIWES*, 2012, 4:1–4:3.

[10] H. Bast, F. Bäurle, B. Buchhold, and E. Haußmann. "Broccoli: Semantic Full-Text Search at your Fingertips". In: *CoRR*, 2012.

[11] H. Bast, F. Bäurle, B. Buchhold, and E. Haußmann. "Easy Access to the Freebase Dataset". In: *WWW*, 2014, pp. 95–98.

[12] H. Bast, F. Bäurle, B. Buchhold, and E. Haußmann. "Semantic Full-Text Search with Broccoli". In: *SIGIR*, 2014, pp. 1265–1266.

[13] H. Bast and B. Buchhold. "An Index for Efficient Semantic Full-Text Search". In: *CIKM*, 2013, pp. 369–378.

[14] H. Bast, B. Buchhold, and E. Haußmann. "A Quality Evaluation of KB+Text Search". In: *KI (under submission)*, 2017.

[15] H. Bast, B. Buchhold, and E. Haußmann. “Relevance Scores for Triples from Type-Like Relations”. In: *SIGIR*, 2015, pp. 243–252.

[16] H. Bast, B. Buchhold, and E. Haußmann. “Semantic Search on Text and Knowledge Bases”. In: *Foundations and Trends in Information Retrieval*, 2016, pp. 119–271.

[17] H. Bast and E. Haußmann. “More Accurate Question Answering on Freebase”. In: *CIKM*, 2015, pp. 1431–1440.

[18] H. Bast and E. Haußmann. “More Informative Open Information Extraction via Simple Inference”. In: *ECIR*, 2014, pp. 585–590.

[19] H. Bast and E. Haußmann. “Open Information Extraction via Contextual Sentence Decomposition”. In: *ICSC*, 2013, pp. 154–159.

[20] J. Berant, A. Chou, R. Frostig, and P. Liang. “Semantic Parsing on Freebase from Question-Answer Pairs”. In: *ACL*, 2013, pp. 1533–1544.

[21] J. Berant and P. Liang. “Imitation Learning of Agenda-based Semantic Parsers”. In: *TACL*, 2015, pp. 545–558.

[22] J. Berant and P. Liang. “Semantic Parsing via Paraphrasing”. In: *ACL*, 2014, pp. 1415–1425.

[23] R. Blanco, P. Mika, and S. Vigna. “Effective and Efficient Entity Search in RDF Data”. In: *ISWC*, 2011, pp. 83–97.

[24] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. “Freebase: a Collaboratively Created Graph Database for Structuring Human Knowledge”. In: *SIGMOD*, 2008, pp. 1247–1250.

[25] A. Bordes, S. Chopra, and J. Weston. “Question Answering with Subgraph Embeddings”. In: *EMNLP*, 2014, pp. 615–620.

[26] L. Breiman. “Random Forests”. In: *Machine Learning*, 2001, pp. 5–32.

[27] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. “Signature Verification Using a Siamese Time Delay Neural Network”. In: *NIPS*, 1993, pp. 737–744.

[28] M. Bron, K. Balog, and M. de Rijke. “Ranking Related Entities: Components and Analyses”. In: *CIKM*, 2010, pp. 1079–1088.

[29] F. Cai and M. de Rijke. “A Survey of Query Auto Completion in Information Retrieval”. In: *Foundations and Trends in Information Retrieval*, 2016, pp. 273–363.

[30] Q. Cai and A. Yates. “Large-Scale Semantic Parsing via Schema Matching and Lexicon Extension”. In: *ACL*, 2013, pp. 423–433.

[31] J. Chen, C. Xiong, and J. Callan. “An Empirical Study of Learning to Rank for Entity Search”. In: *SIGIR*, 2016, pp. 737–740.

[32] A. Chuklin, I. Markov, and M. de Rijke. “Click Models for Web Search”. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2015.

[33] D. Clevert, T. Unterthiner, and S. Hochreiter. “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. In: *CoRR*, 2015.

[34] ClueWeb. The Lemur Project, http://lemurproject.org/clueweb09. 2009.

[35] R. Collobert et al. “Natural Language Processing (Almost) from Scratch”. In: *Journal of Machine Learning Research*, 2011, pp. 2493–2537.

[36] L. D. Corro, A. Abujabal, R. Gemulla, and G. Weikum. “FINET: Context-Aware Fine-Grained Named Entity Typing”. In: *EMNLP*, 2015, pp. 868–878.

[37] L. D. Corro and R. Gemulla. “ClausIE: Clause-Based Open Information Extraction”. In: *WWW*, 2013, pp. 355–366.

[38] X. L. Dong et al. “From Data Fusion to Knowledge Fusion”. In: *PVLDB*, 2014, pp. 881–892.

[39] S. Elbassuoni, M. Ramanath, R. Schenkel, M. Sydow, and G. Weikum. “Language-Model-Based Ranking for Queries on RDF-graphs”. In: *CIKM*, 2009, pp. 977–986.

[40] O. Etzioni et al. “Web-scale Information Extraction in KnowItAll: (Preliminary Results)”. In: *WWW*, 2004, pp. 100–110.

[41] A. Fader, S. Soderland, and O. Etzioni. “Identifying Relations for Open Information Extraction”. In: *ACL*, 2011, pp. 1535–1545.

[42] Y. Fang, L. Si, Z. Yu, Y. Xian, and Y. Xu. “Entity Retrieval with Hierarchical Relevance Model, Exploiting the Structure of Tables and Learning Homepage Classifiers”. In: *TREC*, 2009.

[43] P. Ferragina and U. Scaiella. “TAGME: On-the-Fly Annotation of Short Text Fragments (by Wikipedia Entities)”. In: *CIKM*, 2010, pp. 1625–1628.

[44] J. L. Fleiss. “Measuring Nominal Scale Agreement Among Many Raters”. In: *Psychological bulletin*, 1971, p. 378.

[45] E. Gabrilovich, M. Ringgaard, and A. Subramanya. “FACC1: Freebase Annotation of ClueWeb Corpora, Version 1”. (Release Date 2013-06-26, Format Version 1, Correction Level 0).

[46] M. Gupta and M. Bendersky. “Information Retrieval with Verbose Queries”. In: *Foundations and Trends in Information Retrieval*, 2015, pp. 91–208.

[47] H. Halpin et al. “Evaluating ad-hoc object retrieval”. In: *IWEST*, 2010.

[48] S. Heindorf, M. Potthast, H. Bast, B. Buchhold, and E. Haußmann. “WSDM Cup 2017: Vandalism Detection and Triple Scoring”. In: *WSDM*, 2017, pp. 827–828.

[49] J. Hoffart, D. Milchevski, and G. Weikum. “STICS: Searching with Strings, Things, and Cats”. In: *SIGIR*, 2014, pp. 1247–1248.

[50] Y. Kim. “Convolutional Neural Networks for Sentence Classification”. In: *ACL*, 2014, pp. 1746–1751.

[51] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR*, 2014.

[52] E. Kiperwasser and Y. Goldberg. “Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations”. In: *TACL*, 2016, pp. 313–327.

[53] T. Kwiatkowski, E. Choi, Y. Artzi, and L. S. Zettlemoyer. “Scaling Semantic Parsers with On-the-Fly Ontology Matching”. In: *EMNLP*, 2013, pp. 1545–1556.

[54] X. Li, B. J. A. Schijvenaars, and M. de Rijke. “Investigating Queries and Search Failures in Academic Search”. In: *Information Processing and Management*, 2017, pp. 666–683.

[55] C. Liang, J. Berant, Q. Le, K. D. Forbus, and N. Lao. “Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision”. In: *CoRR*, 2016.

[56] D. Lin and P. Pantel. “DIRT @SBT@Discovery of Inference Rules from Text”. In: *KDD*, 2001, pp. 323–328.

[57] T. Liu. “Learning to Rank for Information Retrieval”. In: *Foundations and Trends in Information Retrieval*, 2009, pp. 225–331.

[58] Mausam, M. Schmitz, S. Soderland, R. Bart, and O. Etzioni. “Open Language Learning for Information Extraction”. In: *EMNLP*, 2012, pp. 523–534.

[59] M. Melucci. “Contextual Search: A Computational Framework”. In: *Foundations and Trends in Information Retrieval*, 2012, pp. 257–405.

[60] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. “Distributed Representations of Words and Phrases and their Compositionality”. In: *NIPS*, 2013, pp. 3111–3119.

[61] G. A. Miller. “WordNet: A Lexical Database for English”. In: *Commun. ACM*, 1995, pp. 39–41.

[62] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. “Distant Supervision for Relation Extraction Without Labeled Data”. In: *ACL*, 2009, pp. 1003–1011.

[63] T. M. Mitchell et al. “Never-Ending Learning”. In: *AAAI*, 2015, pp. 2302–2310.

[64] N. Nakashole, T. Tylenda, and G. Weikum. “Fine-grained Semantic Typing of Emerging Entities”. In: *ACL*, 2013, pp. 1488–1497.

[65] N. Nakashole, G. Weikum, and F. M. Suchanek. “PATTY: A Taxonomy of Relational Patterns with Semantic Types”. In: *EMNLP-CoNLL 2012, July 12-14, 2012, Jeju Island, Korea*, 2012, pp. 1135–1145.

[66] K. Narasimhan, A. Yala, and R. Barzilay. “Improving Information Extraction by Acquiring External Evidence with Reinforcement Learning”. In: *EMNLP*, 2016, pp. 2355–2365.

[67] A. Neelakantan, Q. V. Le, M. Abadi, A. McCallum, and D. Amodei. “Learning a Natural Language Interface with Neural Programmer”. In: *CoRR*, 2016.

[68] H. Paulheim and C. Bizer. “Improving the Quality of Linked Data Using Statistical Distributions”. In: *Int. J. Semantic Web Inf. Syst.* 2014, pp. 63–86.

[69] B. Popov et al. “KIM - Semantic Annotation Platform”. In: *ISWC*, 2003, pp. 834–849.

[70] J. Pound, P. Mika, and H. Zaragoza. “Ad-hoc Object Retrieval in the Web of Data”. In: *WWW*, 2010, pp. 771–780.

[71] D. Ramage, D. L. W. Hall, R. Nallapati, and C. D. Manning. “Labeled LDA: A Supervised Topic Model for Credit Attribution in Multi-Labeled Corpora”. In: *ACL*, 2009, pp. 248–256.

[72] L. Ratinov, D. Roth, D. Downey, and M. Anderson. “Local and Global Algorithms for Disambiguation to Wikipedia”. In: *ACL*, 2011, pp. 1375–1384.

[73] S. Reddy et al. “Transforming Dependency Structures to Logical Forms for Semantic Parsing”. In: *TACL*, 2016, pp. 127–140.

[74] M. Sanderson. “Test Collection Based Evaluation of Information Retrieval Systems”. In: *Foundations and Trends in Information Retrieval*, 2010, pp. 247–375.

[75] S. Sarawagi. “Information Extraction”. In: *Foundations and Trends in Databases*, 2008, pp. 261–377.

[76] M. Schuhmacher, L. Dietz, and S. P. Ponzetto. “Ranking Entities for Web Queries Through Text and Knowledge”. In: *CIKM*, 2015, pp. 1461–1470.

[77] F. Silvestri. “Mining Query Logs: Turning Search Usage Data into Knowledge”. In: *Foundations and Trends in Information Retrieval*, 2010, pp. 1–174.

[78] V. I. Spitkovsky and A. X. Chang. “A Cross-Lingual Dictionary for English Wikipedia Concepts”. In: *LREC*, 2012, pp. 3168–3175.

[79] F. M. Suchanek, G. Kasneci, and G. Weikum. “YAGO: a Core of Semantic Knowledge”. In: *WWW*, 2007, pp. 697–706.

[80] V. Tablan, K. Bontcheva, I. Roberts, and H. Cunningham. “Mímir: An Open-Source Semantic Search Framework for Interactive Information Seeking and Discovery”. In: *J. Web Sem.* 2015, pp. 52–68.

[81] T. Tran, P. Mika, H. Wang, and M. Grobelnik. “SemSearch’11: the 4th Semantic Search Workshop”. In: *WWW*, 2011, pp. 315–316.

[82] D. Vrandecic and M. Krötzsch. “Wikidata: a Free Collaborative Knowledgebase”. In: *CACM*, 2014, pp. 78–85.

[83] K. Xu, Y. Feng, S. Reddy, S. Huang, and D. Zhao. “Enhancing Freebase Question Answering Using Textual Evidence”. In: *CoRR*, 2016.

[84] B. Yang, W. Yih, X. He, J. Gao, and L. Deng. “Embedding Entities and Relations for Learning and Inference in Knowledge Bases”. In: *CoRR*, 2014.

[85] X. Yao, J. Berant, and B. V. Durme. “Freebase QA: Information Extraction or Semantic Parsing?” In: *ACL, Workshop on Semantic Parsing*, 2014.

[86] X. Yao and B. V. Durme. “Information Extraction over Structured Data: Question Answering with Freebase”. In: *ACL*, 2014, pp. 956–966.

[87] A. Yates et al. “TextRunner: Open Information Extraction on the Web”. In: *NAACL*, 2007, pp. 25–26.

[88] W. Yih, M. Chang, X. He, and J. Gao. “Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base”. In: *ACL*, 2015, pp. 1321–1331.

[89] M. A. Yosef, J. Hoffart, I. Bordino, M. Spaniol, and G. Weikum. “AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables”. In: *PVLDB*, 2011, pp. 1450–1453.

Appendix: Publications

The following contains all of our publications in the order in which we listed them in Section 2. The publications are in original form including their page number in the corresponding proceedings or journal. Electronic copies of all publications are available via a web page: `http://ad-publications.cs.uni-freiburg.de/theses/Dissertation_Elmar_Haussmann.html`

A Case for Semantic Full-Text Search

(position paper)

Hannah Bast, Florian Bäurle, Björn Buchhold, Elmar Haussmann
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast,baeurlef,buchholb,haussmann}@informatik.uni-freiburg.de

ABSTRACT

We discuss the advantages and shortcomings of full-text search on the one hand and search in ontologies / triple stores on the other hand. We argue that both techniques have an important quality missing from the other. We advocate a deep integration of the two, and describe the associated requirements and challenges.

1. FULL-TEXT SEARCH

The basic principle of full-text search is that the user enters a (typically small) set of keywords, and the search engine returns a list of documents, in which some or all of these keywords (or variations of them like spelling variants or synonyms) occur. The results are ranked by how *prominent* these occurrences are (term frequency, occurrence in title, relative proximity, absolute importance of the document, etc.)

1.1 Document-oriented queries

This works well as long as (i) the given keywords or variants of them occur in enough of the relevant documents, and (ii) the mentioned prominence of these occurrences is highest for the most relevant documents. For example, a Google query for *broccoli* will return the Wikipedia page as the first hit, because it's a popular page containing the query word in the URL. A query for *broccoli gardening* will also work, because relevant documents will contain both of the words, most likely in a title / heading and in close proximity.

For large document collections (as in web search), the number of matching documents is usually beyond what a human can read. Then *precision* is of primary concern for such queries, not recall. The informational Wikipedia page (or a similar page) should come first, not second or fourth. And it is not important that we find *all* broccoli gardening tips on the internet.

Bottom line: *Full-text queries work well when relevant documents contain the keywords or simple variations of them in a prominent way. The primary concern is precision, not recall.*

JIWES '12 August 12 2012, Portland, OR, USA

1.2 Entity-oriented queries

Consider the query *plants with edible leaves*. As we explain now, this kind of query is inherently problematic for full-text search engines.[1]

The first problem is as follows. Relevant documents are likely to contain the words *edible leaves* or variations of them (see above). But there is no reason why they should contain the word *plants*, or variations of it like *plant* or *botany*. Rather, they will contain the name of a particular plant, for example, *broccoli*. This is exactly the kind of knowledge contained in *ontologies*, discussed in Section 2.

The second problem is that the sought-for results are not documents and also not passages in documents, but rather a list of entities, plants with a certain property. This is not only an issue of convenience for the user, but also one of *result diversity*. Even if the search engine would manage to match instances of plants (like *broccoli*) to the keyword *plant*, the problem remains that the result list contains many hits referring to one and the same (well-known) plant, while many (lesser known) plants will be missing.

Worse than that, the information for a single hit could be spread over several documents. For example, for the query *plants with edible leaves and native to Europe*, the information that a particular plant has edible leaves may be contained in one document, while the information that it is native to Europe may be contained in another document. This is beyond the capabilities of full-text search engines.

Unlike for the queries from the previous subsection, *recall* is much more important now. Precision must not be ignored, but becomes a seconday concern. For example, consider the query *apollo astronauts who walked on the moon* from the 2011 Yahoo Semsearch challenge[2]. A user would certainly like to find all 12 entities matching this query. Regarding precision, it would be acceptable if a number of irrelevant results of the *same* order of magnitude were interspersed.

Bottom line: *Entity-oriented queries typically require ontological knowledge. High recall is of primary concern. Precision must not be ignored, but becomes secondary.*

2. ONTOLOGIES

For the purpose of this short paper, we view ontologies as collections of subject-predicate-object triples (often called *facts*), where each element of each triple has an identifier that is consistent among triples. For example, *Broccoli is-*

[1]Let us ignore the untypical case that a precompiled document containing those words and the result list exist.
[2]`semsearch.yahoo.com`

a Vegetable or *Vegetable is-subclass-of Plant* or *Broccoli is-native-to Europe*.

Given an ontology with sufficient information, it is easy to ask even complex queries, which require the connection of many facts, with a precisely defined semantics. For example, the query for all *plants native to europe* would require the connection of all three facts from the previous paragraph.

2.1 Obtaining the facts

The first obvious problem of ontologies is how to obtain the facts. This is easy, if the facts are already organized in a database. Then all that is needed is a conversion to the proper format. A large part of *linked open data (LOD)* [5] and hence of the BTC datasets [7] is of this kind.

Another source of ontology data are users creating machine-readable fact triples explicitly. Only a relatively small part of the BTC datasets is of this kind.

However, much (if not most) information is stored in the form of natural language text, without semantic markup. For the obvious reason that this is the primary form of communication between human beings. For example, there are thousands of documents, including several Wikipedia articles, on the web stating somewhere in a sentence that the leaves of broccoli are edible. But this information is neither contained in DBpedia, nor in current LOD, nor in BTC.

Bottom line: *Much information is available only in the form of natural language text. This is unlikely to change also in the long run. In particular, for recent and specific information.*

2.2 Information extraction

Extracting facts of the form above from natural language text is a hard problem due to the diversity and ill-definedness of natural language. This task, known as *information extraction*, is an offline process. Whatever information was failed to be extracted will not be contained in the ontology, although it should be. Whatever wrong information was extracted will be in the ontology, although it should not be.

It is exactly one of the secrets of success of full-text search that it avoids this problem in the first place. Full-text search engines simply index (almost) every word in all documents. When the document contains your keywords, you have a chance to find it. Also note how, for the sake of precision, a search engine company like Google has introduced new features (like error-tolerance or synonyms or returning) only at a point where they were virtually error-free.

In contrast, state of the art information extraction from natural language text is far from being error-free. For example, the best system on the ACE 2004 dataset for the extraction of 7 predefined relations reported a precision of 83% and a recall of 72% [11], [1]. In [2], a state of the art system that automatically identifies and extracts arbitrary relationships in a text gives an average precision of 88% and a recall of only 45%. Both systems only extract binary relationships and the extraction of multiway relationships is a significantly more complicated task [13].

Bottom line: *Fact extraction from natural language text is an offline problem with a high error rate. The typical recall is 70% or less even for popular relations.*

2.3 Consistent names

The other big problem with ontology data is consistent naming of entities and relations. LOD solves this by unifying different names meaning the same thing via user-created links (*owl:sameAs*). This works well for popular relations and entities, but for more specific and less popular relations, such user-created links are less likely to exist.

Bottom line: *Unified names for entities and relations are feasible for a core of popular facts, but unreasonable to expect for other facts.*

3. INTEGRATION OF FULL TEXT AND ONTOLOGY DATA

In the previous two sections we have argued how a large part of the world's information is (and will be for a long time) available only as full text, while for a certain core of popular knowledge an ontology is the storage medium of choice. We therefore advocate an integration of the two types of search, which we will refer to as *semantic full-text search*.

We see *four* major research challenges associated with such a semantic full-text search. We will describe each of them in one of the following four subsections. We will also comment how we addressed them in our own prototype for an integrated such search, called *Broccoli* [3]. We encourage the reader to try our online demo available under `broccoli.informatik.uni-freiburg.de`

We remark that we are not claiming that our own prototype is the only way to address these research challenges.

3.1 Entity recognition in the full text

An essential ingredient of a system for semantic full-text search is the recognition of references (including anaphora) to entities from the given ontology[3] in the given full text. For example, consider the following sentence: *The stalks of rhubarb are edible, but its leaves are toxic.* Both of the underlined words should map these words to the corresponding entity or entities from the given ontology, for example, `dbpedia.org/resource/Rhubarb`.

For reasonable query times, this kind of entity recognition has to be done *offline*. However, unlike the fact extraction described in Section 2.2, state-of-the-art methods for entity recognition achieve relatively high values for both precision and recall of around 90% [12].

Bottom line: *Offline entity recognition is an essential ingredient of semantic full-text search. The task is much simpler than full information extraction, with precision and recall values of around 90%.*

3.2 Combined Index

Typical entity-oriented queries like our *plants with edible leaves native to Europe* require three things: (1) finding entities matching the ontology part of the query (*plants native to Europe*), (2) finding text passages matching the full-text part of the query (*edible leaves*), and (3) finding occurrences of the entities from (1) that co-occur with the matches from (2).

For both (1) and (2), efficient index structures with fast query times exist. To solve (3), the solutions from (1) and (2) could be combined at query time. However, this is a major obstacle for fast query times, for two reasons. First, the entity recognition problem described in the previous section would have to be solved at query time. Second, even if the

[3]In particular, this could be from LOD or BTC.

final result is small, the separate result sets for (1) and (2) will often be huge, and fully materializing them is expensive.

In our own prototype *Broccoli*, we therefore propose a joint index with hybrid inverted lists that refer to both word and entity occurrences; for details, see [3].

Bottom line: *Semantic full-text search with fast query times seems to require a joint index over both the word and the entity occurrences.*

3.3 Semantic Context

For queries with a large number of hits, prominence of keyword occurrence has turned out to be a very reliable indicator of relevance. However, entity-oriented queries tend to have a long tail of hits with relatively little evidence in the document collection. Then natural language processing becomes indispensable [8].

For example, consider again the query *plants with edible leaves* and again the sentence *The stalks of rhubarb are edible, but its leaves are toxic.* This sentence is one of only few in the whole Wikipedia matching that query. But it should not count as a hit, since in it *edible* refers only to the *stalks* and not to the *leaves.* For such queries, we need an instrument for determining which words semantically "belong together".

In our own prototype Broccoli, we solve this problem by splitting sentences into subsentences of words that belong together in this way. For the sentence above, after anaphora resolution this would be *The stalks of rhubarb are edible* and *Rhubarb leaves are toxic.* Again, see [3] for details.

Bottom line: *Entity-oriented queries often have hits with little evidence in the document collection. To identify those, a natural language processing is required that tells which words semantically "belong together".*

3.4 User interface

We discuss two challenges which are particularly hard and important for semantic full-text search, especially in combination: *ease of use* and *transparency.*

A standard query language for ontology search is SPARQL [10]. The big advantage is its precise query semantics. The big disadvantage is that most users are either not willing or not able (or both) to learn / use such a complex query language. Languages like SPARQL are useful for the work behind the scenes, but not for the front-end.

On the other extreme of the spectrum is keyword search, the simplicity of which is one the secrets of success of full-text search. For full-text search, the query semantics of keyword search is reasonably transparent: the user gets documents which contain some or all of the keywords. For semantic full-text search this is no longer the case. Which part of the query was considered as an entity, which as a word, and which as a class of entities? How were these parts put in relation to each other?

The other major ingredient of transparency, besides a precise query semantics, are *results snippets.* Result snippets serve two main purposes. First, clarifying why the respective hit was returned. Second, allowing the user a quick check whether the hit is relevant.

Systems for what has become known as *ad-hoc object retrieval* [9] try to infer the query semantics from a simple keyword query. Result snippets are treated as a separate problem [6]. In existing semantic search engines on the web, they are often of low quality and little use, e.g. Falcons or SWSE [4].

In our own prototype Broccoli, we use a hybrid approach. Like in keyword search, there is only a single search field. However, using it, the user can build a query, where part of the semantic structure is made explicit. This process is guided by extensive search-as-you-type query suggestions. Due to lack of space here, we refer the reader to our online demo under `broccoli.informatik.uni-freiburg.de`.

Bottom line: *Particular challenges for a user interface for semantic full-text search are ease of use and transparency. Of the currently existing semantic search engines, most neglect one or even both.*

4. REFERENCES

[1] C. C. Aggarwal and C. Zhai, editors. *Mining Text Data*. Springer, 2012.

[2] M. Banko and O. Etzioni. The tradeoffs between open and traditional relation extraction. In *ACL*, pages 28–36, 2008.

[3] H. Bast, F. Bäurle, B. Buchhold, and E. Haussmann. Broccoli: Semantic full-text search at your fingertips. *CoRR*, ad.informatik.uni-freiburg.de/papers, 2012.

[4] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[5] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked data on the web. In *WWW*, pages 1265–1266, 2008.

[6] R. Blanco and H. Zaragoza. Finding support sentences for entities. In *SIGIR*, pages 339–346, 2010.

[7] Billion triple challenge dataset 2012. `http://km.aifb.kit.edu/projects/btc-2012/`.

[8] S. T. Dumais, M. Banko, E. Brill, J. J. Lin, and A. Y. Ng. Web question answering: is more always better? In *SIGIR*, pages 291–298, 2002.

[9] J. Pound, P. Mika, and H. Zaragoza. Ad-hoc object retrieval in the web of data. In *WWW*, pages 771–780, 2010.

[10] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C recommendation, W3C, Jan. 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[11] L. Qian, G. Zhou, F. Kong, Q. Zhu, and P. Qian. Exploiting constituent dependencies for tree kernel-based semantic relation extraction. In *COLING*, pages 697–704, 2008.

[12] E. F. T. K. Sang and F. D. Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. *CoRR*, cs.CL/0306050, 2003.

[13] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

Semantic Full-text Search with Broccoli

Hannah Bast, Florian Bäurle, Björn Buchhold, Elmar Haußmann
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast, baeurlef, buchhold, haussmann}@informatik.uni-freiburg.de

ABSTRACT

We combine search in triple stores with full-text search into what we call *semantic full-text search*. We provide a fully functional web application that allows the incremental construction of complex queries on the English Wikipedia combined with the facts from Freebase. The user is guided by context-sensitive suggestions of matching words, instances, classes, and relations after each keystroke. We also provide a powerful API, which may be used for research tasks or as a back end, e.g., for a question answering system. Our web application and public API are available under `http://broccoli.cs.uni-freiburg.de`.

1. INTRODUCTION

Knowledge is available in electronic form in two main representations: as natural language text (e.g., Wikipedia), and in structured form (e.g., Freebase). The central motivation behind our system is that both representations have their advantages and should be combined for high-quality semantic search.[1]

For example, consider the query *Plants with edible leaves and rich in vitamin C*. Information about which plant contains how much vitamin C is naturally represented as fact triples. Indeed, this information is found in a knowledge base like Freebase. Information about which plants have edible leaves is more likely to be mentioned in natural language text. It is mentioned many times in Wikipedia, but we don't find it in Freebase (or any other knowledge base that we know of). In principle, the information could be added, but there will always be specific or recent information only described as text.

In the following, we describe how we combine these two information sources in a deep way. Figure 1 shows a screenshot of our demo system in action for our example query.

[1]As a matter of fact, Wikipedia also contains some structured data, and Freebase also contains natural language text.

SIGIR'14, July 6–11, 2014, Gold Coast, Queensland, Australia.
ACM 978-1-4503-2257-7/14/07.
http://dx.doi.org/10.1145/2600428.2611186.

2. SYSTEM OVERVIEW

Preprocessing In principle, our system works for any given text corpus and ontology. For our demo we use the English Wikipedia (text) + Freebase (ontology). We preprocess this data in three phases. First, we link entities from the ontology to mentions in the full text, utilizing Wikipedia links and a set of heuristics as described in [1]. This provides the basis for our *occurs-with* operator explained below. Second, the full text is split into *contexts* that "semantically belong together" as described in [3]. This is key for results of high quality. Third, the special-purpose index described in [2] is built. This is key for providing results and suggestions in real time.

Queries The user interface allows to incrementally construct basic tree-like SPARQL queries, extended by an additional relation *occurs-with*. This relation allows to specify the co-occurrence of entities from the ontology with words from the text. For our example query, the back end computes all occurrences of plants that occur in the same context (see above) as the words *edible* and *leaves*. We also provide the special relation *has-occurrence-of*, to search for documents in which words and entities co-occur. Regular full-text search is thus included as a special case.

Query Suggestions Based on the input of a user, our system gives context-sensitive suggestions for words, classes, instances, and relations. The displayed suggestions always lead to hits, and the more / higher-scored hits they lead to, the higher they are ranked. This is an elementary feature for any system that utilizes a very large ontology. Without good suggestions it is very hard to guess how entities and relations are named, even for expert users.

Excerpts For each result hit (an entity or a document), matching evidence for each part of the query is provided. This is invaluable for the user to judge whether a hit indeed matches the query intent. The UI also provides (on click) detailed information about the NLP preprocessing (see above).

3. TARGET USERS

We see two uses of our system, and hence also two target groups of users.

Our first target group is expert searchers. Our search interface is more complex than ordinary keyword search or asking questions in natural language. The benefit is a powerful query language with precise result semantics. Under these constraints, we have made the query process as easy-to-use as possible. For example, there are tens of thousands of Wikipedia List pages like "Plants with edible leaves".

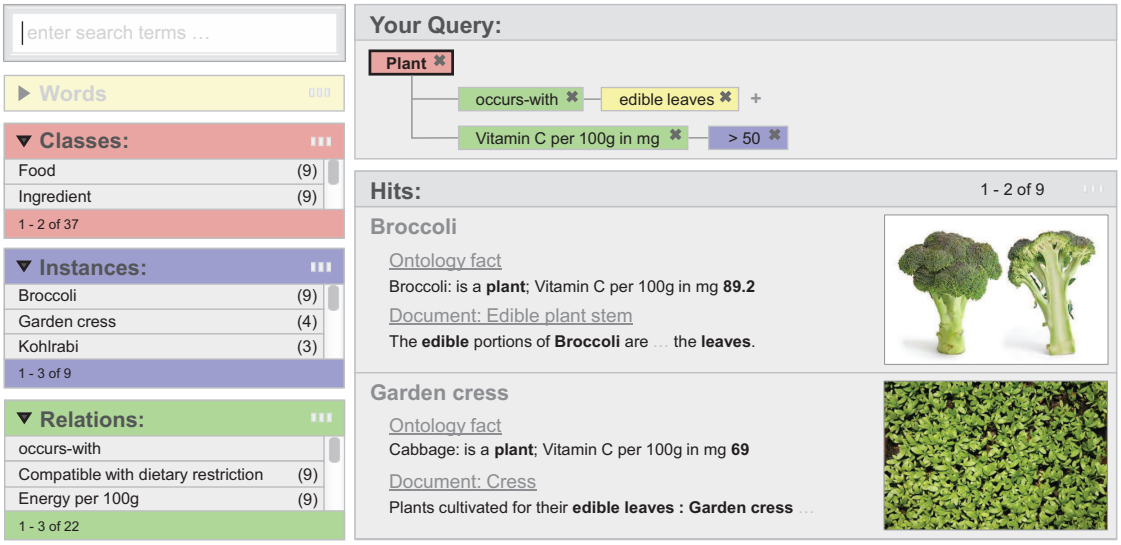

Figure 1: A screenshot of our demo system. The current query is visualized on the top right as a tree. Below, the result hits are shown, grouped by instance (entity) and ranked by relevance, together with evidence from the ontology and the full text. The query can be extended further with the search field on the top left. The four boxes below provide context-sensitive suggestions that depend on the current focus in the query, here: matching sub and super classes, instances, and relations for plants matching the current query.

Many of these are actually non-trivial semantic queries, which are hard to answer with traditional tools, like Google web search. We expect our tool to be a great asset for contributors to such List pages. We expect a similar benefit for expert searches in other areas, e.g., news (presidential campaign backers) or medicine (symptoms of a disease).

Our second target group is researchers in semantic search or engineers of such systems. They may want to use our system to explore the given data and thus gain insight into which facts are expressed in which ways. Engineers may also use our API as a back end for a more simplistic front end, suited for non-expert users. As a first step towards such a front end, we have integrated the following feature in our demo: when typing a query with three or more words without following any of the suggestions, the system tries to convert these keywords into a matching structured query. For example, try *mafia films directed by francis coppola*.

4. RELATED WORK

We see three lines of research closely related to our system.

First, we already mentioned systems for semantic search with more elaborate front ends. In particular, such allowing natural language queries like IBM's well-known Watson [4], or standard keyword queries like in ad-hoc entity search [5]. When they work, such more intuitive front ends are clearly to be preferred. However, semantic search is complex and hard, and queries often fail. Then simple front ends lack the feedback needed to understand what went wrong and what can be done to ask a better query.

Second, there are various extensions of ontology search by a free-text component. A good example is the MQL language (similar to the more standard SPARQL) provided by Freebase (`http://www.freebase.com/query`). In MQL, objects of triples can also be string literals and these can be matched against regular expressions and keyword queries. For example, find all songs containing the words *love* and *you* in their title. In principle, this could be used to simulate our *occurs-with* operator, but only very inefficiently; see [2, Section 4 and Table 1].

Third, information extraction (IE) aims at extracting factual knowledge from text. If this succeeded perfectly, ontology search would be all we need. There are two caveats, however. Whatever information was not extracted properly is lost. In our system, all the original information is kept and is, in principle, accessible by an appropriate query. Also, IE triples often have string literals as objects. Dealing efficiently with these requires a special index data structure, like the one behind our search.

5. REFERENCES

[1] H. Bast, F. Bäurle, B. Buchhold, and E. Haussmann. Broccoli: Semantic full-text search at your fingertips. *CoRR*, abs/1207.2615, 2012.

[2] H. Bast and B. Buchhold. An index for efficient semantic full-text search. In *CIKM*, pages 369–378, 2013.

[3] H. Bast and E. Haussmann. Open information extraction via contextual sentence decomposition. In *ICSC*, pages 154–159, 2013.

[4] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty. Building watson: An overview of the DeepQA project. *AI Magazine*, 31(3):59–79, 2010.

[5] J. Pound, P. Mika, and H. Zaragoza. Ad-hoc object retrieval in the web of data. In *WWW*, pages 771–780, 2010.

Easy Access to the Freebase Dataset

Hannah Bast, Florian Bäurle, Björn Buchhold, Elmar Haußmann
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast, baeurlef, buchhold, haussmann}@informatik.uni-freiburg.de

ABSTRACT

We demonstrate a system for fast and intuitive exploration of the Freebase dataset. This required solving several non-trivial problems, including: entity scores for proper ranking and name disambiguation, a unique meaningful name for every entity and every type, extraction of canonical binary relations from multi-way relations (which in Freebase are modeled via so-called mediator objects), computing the transitive hull of selected relations, and identifying and merging duplicates. Our contribution is two-fold. First, we provide for download an up-to-date version of the Freebase data, enriched and simplified as just sketched. Second, we offer a user interface for exploring and searching this data set. The data set, the user interface and a demo video are available from `http://freebase-easy.cs.uni-freiburg.de`.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General

Keywords

Freebase; Knowledge Base; Ontology

1. INTRODUCTION

Freebase [2] is designed as an open, community-curated knowledge base. With more than 40 million topics and over 2 billion facts, it is today by far the most comprehensive publicly available source of general-knowledge facts.

The complete Freebase data is available for free use, sharing, and adaption (even commercially) under a creative commons license. The data format is N-Triples RDF, which is standard for triple data. In principle, the data can therefore be loaded into any state-of-the-art triple store and queried via standard semantic query languages such as SPARQL. Freebase also provides an own API. The query language used there is MQL.

However, when working with this raw data via SPARQL or with the Freebase API via MQL, several major usability issues arise, also for expert users. Consider the query for winners of the Palme d'Or[1], shown in Figure 1. This appears to be a simple query, which requires only a single relation. In SPARQL one would like to write something like this:

```
select ?x where { ?x Awards-Won "Palme d'Or" }
```

But the required SPARQL query on the provided RDF data dump looks like this:

```
select ?name where {
  ?x  ns:award/award_winner/awards_won  ?m  .
  ?m  ns:award/award_honor/award  ?a  .
  ?a  ns:type/object/name  "Palme d'Or"@en  .
  ?x  ns:type/object/name  ?name  .
}
```

Already this simple example hints at a number of usability issues. How to guess the right relation names? How to guess the right schema (the object of the *awards_won* relation is a so-called mediator object, which is linked, via another relation, to the actual award entity)? How to guess the right entity names (Palme d'Or in this case)? The results are opaque, too. Here is the link to the result for the equivalent MQL query (the complexity of which is similar to that of the SPARQL query above): `http://tinyurl.com/l2pdms5`. In particular, the ranking is merely lexicographic and there are ambiguous names like *Michael Moore*. For more complex queries, e.g. `http://tinyurl.com/qzrc77j`, these problems intensify.

In contrast, the query in Figure 1 is as one would expect. As we will see later, the user interface helps in finding the proper relation names. The results are properly ranked, with the most prominent hits (directors in this case) at the top. The names of the directors are as expected, and accompanied by pictures. What is not shown is that there are 16 persons in Freebase with the name *Michael Moore*. In our version of the Freebase data set, only the (in)famous director gets exactly that name. The others are disambiguated by meaningful suffixes, e.g. *Michael Moore (Soccer Forward)*. Finally, the user interface offers suggestions for sensible ways to augment the query, e.g. by the relation *Country of nationality*.

1.1 Our contribution

We address all of the problems from the example query above, as well as several other problems that occur with typical queries and impact usability.

Entity Scores. As in standard text search, long result lists demand for a proper ranking. For example, for our example query above, we would like to have the most prominent

[1]This is the highest prize at the annual Cannes Film Festival.

WWW'14, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2744-2/14/04.
http://dx.doi.org/10.1145/2567948.2577016.

Figure 1: A screenshot of our demo system showing results for a query for winners of the Palme d'Or. For explanations of the various components and features, see the paragraph before Section 1.1 and Section 3.

people at the top. We provide a prominence score for each entity in Freebase; see Section 2.2.

Entity Names. In Freebase, each entity has a unique alpha-numerical so-called machine id or mid, e.g. */m/0jw67*. In most applications, it is desirable to also have unique names that are meaningful for humans. This is also the approach Wikipedia takes. There, entities are distinguished with suffixes. For example, *Europe* denotes the continent as expected, while the Swedish rock band with the same name is called *Europe (band)*. For the sake of consistency, these suffixes follow several rules, but ultimately they are chosen by humans. We automatically compute such names for each entity in Freebase. This is described in Section 2.3.

Mediators. In our introductory example, we have encountered the complex *awards_won* relation. It involves a mediator object that itself is related to several entities, including not only the person who won the award and the award won, but also supplementary information like the date of the award and the winning work. Still, for many queries the "main" binary relation (between the person and the award in this case) is all that is needed, and would be much easier to use. We automatically extract this binary relation from each mediator; see in Section 2.4.

Transitivity. Many relations are practically unusable when they are not closed under transitivity. The relation *Contained by* between locations is a prominent example. We compute the transitive hull for several large (manually selected) relations from Freebase; see Section 2.5.

Duplicates. Duplicate entities or types with the same or a similar name are frequent in Freebase. For example, there are four classes called *Person* or *person*. Usually, additional types with the same name have few instances and are added as a user's mistake. The problem is aggravated by our own addition of types to the taxonomy; see the next item. We identify duplicates, merge them and give them a proper canonical name; see Section 2.6.

Taxonomy. Freebase by itself has a comparably shallow taxonomy (3,557 different types at the time of this writing) expressed via its *type/object/type* relation. However, many intuitive semantic classes like *plant* or *politician* are not types. Instead this information is available only via relations, e.g. *Profession*. We apply a set of configurable relations with objects that are to be included in the taxonomy. Our resulting taxonomy has a total of 21,042 different types. See `http://freebase-easy.cs.uni-freiburg.de` for more details.

User Interface. We provide a fully-functional user interface that allows for an interactive exploration and search using all of the features above. See Figure 1 and our description in Section 3. The demo is available under the link above; we encourage the reader to try it out.

Download. Along with the demo, we also provide our version of the Freebase data set, with all of the mentioned features, for download. A zip file (2.4 GB at the time of this writing) is available under the link above. Our data curation pipeline (see Section 2) is fully automized. This allows us to easily update the data set on a regular basis, and thus keep pace with the continuously growing Freebase data.

We remark that some of the items above represent major research challenges. For this demo paper, we apply comparably simple and straightforward solutions. However, as can be seen from the demo, these already go a long way towards an easier access to and better usability of the Freebase data.

1.2 Related Work

There is an abundance of work on providing more convenient front ends for semantic search. See [4] for a small survey, and the many papers citing that work. None of these achieve context-sensitive query suggestions at interactive speed as in our user interface (for a data set as large as Freebase); see also [1, Section 2].

Concerning our data curation pipeline, we do not claim particular novelty for any of the components. Our contri-

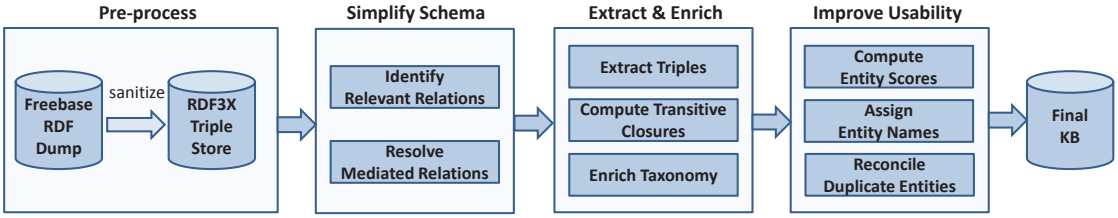

Figure 2: Architectural overview of our pipeline for a more easy-to-use version of the Freebase data set.

bution is that we have identified the major issues for the (widely used) Freebase data set, and provide a version that is much more easily accessible, and a ready-to-use demo application. We know of no comparable effort to date.

2. DESCRIPTION OF OUR PIPELINE

Freebase provides raw data dumps in the form of RDF-triples. As explained above, working with this raw data is complex for a variety of reasons. We therefore seek to simplify and enrich it in several ways. Figure 2 illustrates the general pipeline of our architecture. The various steps of the pipeline are described in the following subsections.

2.1 Data Sanitization

The raw RDF data contains redundant information as well as information which is undesired or even annoying in most use cases. We therefore first load the raw RDF data into RDF-3X [5], a fast triple store, and then extract the relations we are interested in using appropriate SPARQL queries. Namely, we omit relations with few facts (< 5) and relations that are not part of the core data in Freebase (e.g., facts in the domains *user* and *base*). Further, Freebase contains many (but not all) relations in two directions, e.g., *place-of-birth* and *people-born-here*. For all those, we only extract one direction (the one with more subjects than objects)[2].

2.2 Entity Scores

Scores indicating the prominence of entities are essential when ranking result entities (rank prominent entities first) and when resolving naming conflicts (assign the most prominent entity the canonical name, see Section 2.3). Intuitively, the more people talk (or write) about an entity the more prominent it is. We utilize the mentions of Freebase entities in the ClueWeb'12 Corpus[3] (733M web pages) from [3] to count the number of mentions of each entity and use it as a score. Given a set of mentions M_{CW_e} of an entity e we use s_{CW} as the resulting score:

$$s_{CW}(e) = |M_{CW_e}|$$

About 4.5 million distinct entities were recognized in ClueWeb, but our knowledge base contains a total of 39.6 million distinct entities. Therefore, we additionally compute a score based on the knowledge base and its relations in the following way:

$$s_{KB}(e) = \sum_{r \in R} \log(\max(1, |\{x \mid (e, r, x) \in KB\}|)) + \sum_{r \in R} \log(\max(1, |\{x \mid (x, r, e) \in KB\}|))$$

R is the set of all relation types in the knowledge base and KB denotes its set of relational triples (x, r, y). The above is the sum of the log of a per-relation out-degree and in-degree with the intuition that an entity with many incoming and outgoing relations is more prominent. The main effect of this score is as a tie-braker, when two entities have the same number of occurrences in the ClueWeb collections or were not mentioned or recognized at all. As final score for an entity we use the sum of the ClueWeb and knowledge based score:

$$s(e) = s_{CW}(e) + s_{KB}(e)$$

2.3 Entity names

As discussed in the introduction, a unique meaningful name for each entity is highly desirable in many applications. However, the raw Freebase data only provides alphanumerical ids and highly ambiguous names. In Wikipedia, this problem is solved manually as follows. For an ambiguous name, the most prominent entity gets the name without further additions. For example, the director from our example query in Figure 1 is called *Michael Moore*. Other contestants for the same name are distinguished by a meaningful suffix, e.g. *Michael Moore (Australian politician)*.

For the Freebase data, we automatically assign unique names as follows.[4] If there is no name at all, use the alphanumerical id from Freebase. Otherwise, there will be a set of candidates that compete for a name. Note that these candidates can be types (e.g. *Director*) as well as entities (e.g. *Michael Moore*) Also note that a type and an entity can have the same name in Freebase (e.g., there is a type *Person* and several entities with that name). The score for an entity is simply the score from Section 2.2. The score for a type is simply the maximum score of an entity plus the number of instances of that type. The literal name (without suffixes) then goes to the candidate with the highest score.

The remaining candidates are disambiguated as follows. If they are located in a country, they compete for the name *<name> (<country>)*. Again, the entity with the highest score gets that name. The others get an additional numerical suffix, e.g. *Berlin (United States) #2*. Entities without locational information are disambiguated using their *notable-for* relation, e.g. *Michael Moore (Soccer Forward)*. If that is not enough to achieve unique names, again a numerical suffix is

[2]Most applications, including our own here, can handle queries for the reverse direction without requiring a copy of it.
[3]http://lemurproject.org/clueweb12/
[4]Note that for most Freebase entries, there is no associated Wikipedia entry.

added. Entities that have neither locational nor *notable-for* information are disambiguated using their Freebase ids, e.g. *Maria (m/0760g8)*.

2.4 Mediators

As explained in the introduction, Freebase realizes multi-way relations using so-called mediator objects. For example, for a fact from Freebase's *Awards won* relation, the object is such a mediator object of type *award_honor*. This object is then related to the actual award, but also to supplementary information such as the winning work or the date of the award.

For each mediator type m (e.g. *award_honor*), we do the following. Intuitively, there are two types of mediators, which require a different approach. Namely, m either mediates between two entities in different roles (e.g. a musician and a group) or in the same role (e.g. siblings). We found the following strategy to differentiate very well between these two cases.

Consider the k relations that have m as subject.[5] Let $n_1 \geq \ldots \geq n_k$ be the number of facts in each of these k relations, sorted in decreasing order. Let r be the relation pertaining to n_1. If $k \geq 2$ and $n_2 \geq n_1/2$, let r' be the relation pertaining to n_2, otherwise let $r' = r$. Intuitively, r and r' are hence the most "frequent" relations, with $r = r'$ for a relation like "sibling". It remains to "merge" r and r' to the desired binary relation and give it a proper name.

Let n be the name of the reverse direction of the relation r according to Freebase. If no such name can be obtained, try the reverse relation of r'. In the very rare event that this fails too, fall back to $n = r$. We then extract a binary relation r_m in the following way.

$$r_m = \{(s, n, o) \mid (x, r, s) \in KB \wedge (x, r', o) \in KB \wedge s \neq o\}$$

This gives us exactly one binary relation for each mediator type m.

2.5 Transitivity

Freebase does not provide the transitive closure of transitive relations. Given R_1 and R_2, the tuples of two relations r_1 and r_2, we compute the transitive closure of tuples to be added during extraction as follows:

$$R_t = R_1 \circ R_2^+$$

Where R_2^+ is the transitive of relation r_2 and $\circ$ is relation composition. This allows computing the transitive closure over two relations, e.g., *profession* and *is-specialization-of* to ensure that a person with the profession *physicist* also has the profession *scientist* (because the profession *physicist* is a specialization of *scientist*). The classic transitive closure is a special case where r_1 equals r_2. We currently provide a manually compiled list of relations for which the transitive closure should be computed.

2.6 Duplicate Classes

A common problem in knowledge bases is that of duplicate entities or classes, often with identical or slightly different names. We follow a simple approach and merge two classes if they have the same name, ignoring case, and if the instances of one class are included in the other by a threshold. Let I_A and I_B be the instances/entities of some class A and B, respectively. We only merge class A into class B if the instances of class A are contained to at least 70% in class B, that is when:

$$\frac{|I_A \cap I_B|}{|I_A|} \geq 0.7$$

and vice versa.

[5] We always have $k \geq 1$ and for few relations, like "sibling", we indeed have $k = 1$.

3. USER INTERFACE

We provide a convenient user interface for performing complex searches on our version of the Freebase dataset, as described in the previous section. The main features are as follows. We encourage the reader to try our demo under `http://freebase-easy.cs.uni-freiburg.de`.

(1) A single input field, as in standard text search.

(2) Incremental query construction with suggestions (for matching types, instance and relations) after each keystroke.

(3) Example tooltips for each relation (shown on mouse over), to help understand what the relation is about (relation names in Freebase are sometimes opaque).

(4) Visual editing of the current query graph (e.g., removing a part or double-clicking a node to make it the new root).

(5) Meaningful names (following Section 2.3) and images (loaded from Freebase, if available).

(6) Proper ranking of results, using the scores from Section 2.2 where appropriate.

(7) Sort by an arbitrary query element, i.p. dates and values.

(8) Interactive query times, using the index from [1].

4. CONCLUSION

We provide a curated version of the Freebase data set that fixes several major usability issues with the original data set. We also provide a convenient user interface for interactive search and exploration, making good use of the various features we added. Several of the problems we addressed are major research problems in their own right. The solutions we provided here are simple and effective, yet by no means perfect. For example, our entity scores (derived from counts in the ClueWeb'12 corpus) work very well to bring the prominent entities to the top, but in some cases show an undesirable topic drift (e.g., Celine Dion is the fourth most prominent person). Our canonical entity names work like a charm for the more frequent entities, while names like *Berlin (United States) #2* could be improved.

5. REFERENCES

[1] H. Bast and B. Buchhold. An index for efficient semantic full-text search. In *CIKM*, pages 369–378, 2013.

[2] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[3] E. Gabrilovich, M. Ringgaard, and A. Subramanya. FACC1: Freebase annotation of ClueWeb corpora, Version 1. (Release date 2013-06-26, Format version 1, Correction level 0).

[4] E. Kaufmann and A. Bernstein. How useful are natural language interfaces to the semantic web for casual end-users? In *ISWC*, pages 281–294, 2007.

[5] T. Neumann and G. Weikum. Scalable join processing on very large RDF graphs. In *SIGMOD*, pages 627–640, 2009.

Open Information Extraction via Contextual Sentence Decomposition[1]

Hannah Bast, Elmar Haussmann
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{*bast,haussmann*}*@informatik.uni-freiburg.de*

***Abstract*—We show how contextual sentence decomposition (CSD), a technique originally developed for high-precision semantic search, can be used for open information extraction (OIE). Intuitively, CSD decomposes a sentence into the parts that semantically "belong together". By identifying the (implicit or explicit) verb in each such part, we obtain facts like in OIE. We compare our system, called CSD-IE, to three state-of-the-art OIE systems: ReVerb, OLLIE, and ClausIE. We consider the following aspects: accuracy (does the extracted triple express a meaningful fact, which is also expressed in the original sentence), minimality (can the extracted triple be further decomposed into smaller meaningful triples), coverage (percentage of text contained in at least one extracted triple), and number of facts extracted. We show how CSD-IE clearly outperforms ReVerb and OLLIE in terms of coverage and recall, but at comparable accuracy and minimality, and how CSD-IE achieves precision and recall comparable to ClausIE, but at significantly better minimality.**[1]

***Keywords*-open information extraction; contextual sentence decomposition; semantic search;**

I. Introduction

Information extraction (IE) is the task of automatically extracting relational tuples from natural language text. Such relational tuples typically take the form *subject predicate object (SPO)*, for example: *(Ruth Gabriel) (was born) (in San Fernando)*. In early IE systems, the desired relations (predicates) were part of the input, for example *born in*. Such a system was then typically given, for each such relation, a set of correct triples from which it could learn. In recent years, the trend has been towards open information extraction (OIE), where identifying the predicate and hence the relation is part of the problem [1]. Many systems for OIE have been developed in recent years; we describe the most recent ones in Section II.

A classical use case for information extraction is to obtain fact triples for a formal ontology. This use case requires that the S, P, and O parts are disambiguated, that is, different formulations referring to the same entity are mapped to the same identifier. For example, the S part of the triple above should be mapped to the actress Ruth Gabriel, regardless of whether in that part she is referred to as *Ruth Gabriel* (like above), *R. Gabriel*, *Gabriel*, or *she* (assuming, of course, that all these references actually mean her). In OIE, this disambiguation is typically not considered part of the problem. More than that, many facts extracted by traditional OIE systems are not easily disambiguated, because the S and O part often contain references to more than one entity. We come back to this important aspect below, when we discuss the aspect of *minimality* of a triple.

The motivation for our approach comes from an application called semantic full-text search (SFTS) [2]. SFTS combines formal ontology search with classical full-text search. A typical query would be *class:person word:writer*, searching for co-occurrences of a reference to a *person* with the word *writer*. The intention of the query is to find people who are writers. For results of good quality, it is crucial that the two occurrences (or more for a longer query), actually "belong together" semantically. For example, consider the following sentence, which will be our running example throughout the paper:

> *(S): Ruth Gabriel, daughter of the actress and writer Ana Maria Bueno, was born in San Fernando.*

The sentence contains two references to a person, *Ruth Gabriel* and *Ana Maria Bueno*, as well as the word *writer*. However, only the fact that Ana Maria Bueno is a writer is supported by the sentence. Returning Ruth Gabriel as a hit for the query above would be a mistake. In [2], we therefore proposed *contextual sentence decomposition* (CSD). The goal of CSD is to compute, for a given sentence, all sub-sequences of words in that sentence that semantically "belong together". The sub-sequences are then called the *contexts* of the sentence. A correct decomposition of the sentence above would yield the following four contexts (in any order):

#1: *Ruth Gabriel was born in San Fernando*
#2: *Ruth Gabriel, daughter of Ana Maria Bueno*
#3: *actress Ana Maria Bueno*
#4: *writer Ana Maria Bueno*

Note that not splitting *actress and writer Ana Maria Bueno* into #3 and #4 would be considered a mistake in CSD, because the words *actress* and *writer* do not "belong together" semantically. Rather, in this sentence, they are two unrelated attributes related to the same person.

[1] An extended version of the paper is available via the authors' website: http://ad.informatik.uni-freiburg.de/publications.

In this paper we explore the use of CSD for OIE. In fact, the contexts above already look close to the kind of triples expected from an OIE system. All that is missing is the distinction into the subject, predicate, and object part. Since CSD is computed from a full parse of the sentence, with explicit markup denoting the verb phrases, this is relatively straightforward. Also note that some of the contexts above are missing a verb. In that case the verb is implicit, but can (typically) be easily deduced from the context, e.g. *is* for contexts #3 and #4 (noun phrase with pre-modifying noun phrase). Our system, called CSD-IE is described in detail in Section III.

A. Quality Aspects

In this paper, we evaluate OIE systems with respect to the following three quality aspects.

1. The accuracy of the extracted facts. Two aspects are important here. First, whether the fact actually has the form of a meaningful relational triple. For example, the triple *(Ruth) (Gabriel was) (San Fernando)* would be considered inaccurate for two reasons: (1) the P part contains words which do not belong to the verb (but to the S part in this case), and (2) the P and the O parts do not fit together. Second, what is expressed by the triple should also be expressed by the sentence. For example, *(Ruth Gabriel) (is) (actress)* would be accurate according to the criterion just mentioned, but it's not expressed in our sentence (S) above.

Accuracy is typically assessed by human judges, see Section IV. The percentage of the extracted facts deemed accurate is typically referred to as the *precision*.

2. The number of extracted facts. An average sentence may express a lot of facts. For example, our example sentence from above expresses four facts:

#1: *(Ruth Gabriel) (was born) (in San Fernando)*
#2: *(Ruth Gabriel) (is) (daughter of Ana Maria Bueno)*
#3: *(Ana Maria Bueno) (is) (actress)*
#4: *(Ana Maria Bueno) (is) (writer)*

It is the explicit goal of our system to extract, from a given sentence, as many facts as possible, and lose as little information as possible. In Section IV we measure this by the *coverage*, that is, the percentage of all word occurrences that occur in at least one extracted triple. For the sentence above, the coverage is 100%.

3. The minimality of the extracted facts. An accurate fact may itself contain other (accurate) facts. For example, *(Ruth Gabriel) (is) (daughter of the actress and writer Ana Maria Bueno)* would be considered an accurate fact according to our definition above. However, we already explained above that this fact contains two other facts (namely, that Ana Maria Bueno is an actress and a writer), which are unrelated to the containing fact and that this mixture of unrelated facts is problematic for applications that require "semantic togetherness" of the words in a fact. It is therefore another explicit goal of our system to extract minimal facts. The four facts listed under 2. above are all minimal.

The goal of minimality comes with a challenge that is not apparent in our example sentences, but occurs in sentences of a more complex type. For example, consider the sentence:

The Embassy said that 6,700 Americans were in Pakistan.

This sentence contains two facts: that the Embassy made some statement, and that 6,700 Americans were in Pakistan. However, by simply extracting these two facts, we would lose the information what statement the Embassy made. We solve this by allowing the S and O part of a fact to contain *references* to other extracted facts. In this case, we would extract:

#1: *(The Embassy) (said) (that #2)*
#2: *(6,700 Americans) (were) (in Pakistan.)*

where the numbers are simply unique ids for each extracted triple. That way, no information is lost, and the application may choose to either keep the facts separate (and avoid mixing of facts) or substitute the references with the referred fact (and thus obtain output as in other OIE systems). A similar issue was addressed by OLLIE [3] using what they call additional context information; see Section II.

B. Our contribution

We present a new system for open information extraction, called CSD-IE, that is good in all three aspects above. We compare our approach to what we consider the three best previous approaches: ReVerb, OLLIE, and ClausIE. CSD-IE outperforms ReVerb and OLLIE in terms of coverage and recall, but at comparable precision and minimality. It also achieves precision and recall comparable to ClausIE, but at significantly better minimality. For some details about previous systems and how they relate to our approach see the next Section II. The details behind our CSD-IE are described in Section III. Our evaluation is provided in Section IV.

II. Related work

A large variety of OIE systems have been developed in recent years, starting with the original TextRunner [1], over WOE [4], R2A2 [5], ReVerb [6], OLLIE [3], to the very recent ClausIE [7]. A good and up-to-date overview over these and other systems is provided in [7]. In the following we will shortly describe how our strongest competitors (ReVerb, OLLIE, and ClausIE) relate to our approach.

ReVerb explicitly addressed the issues of *incoherent* extractions and *uninformative* extractions. Using shallow NLP and learned extraction patterns ReVerb achieves a significantly better precision than its predecessors. Our approach utilizes a full parse which helps with these problems; see the description of ClausIE below and our description of CSD-IE in Section III.

OLLIE improves over ReVerb by addressing two further issues important for extraction quality. The first issue are facts not mediated by verbs. The second issue is additional information about facts expressed in indirect speech (*He said that* ...) and the like. OLLIE relies on a dependency parse and achieves significantly better recall than ReVerb with basically the same good precision. Our contextual sentence decomposition deals with these issues by allowing triples to contain references to other triples and explicitly considering facts not mediated by verbs. This has the advantage of simultaneously addressing the issue of information loss and of minimality.

ClausIE is the most recent OIE system, and our strongest competitor. In fact, the basic approach of ClausIE is very similar to ours: decompose each sentence into its basic constituents (called "clauses" in ClausIE), and from those constituents derive triples. Our approach has been developed in independent work and the basic ideas behind our contextual sentence decomposition (CSD), as used for our semantic search, have already been published in [8] and [2], long before [7]. Still, there are some important differences between our CSD-IE and ClausIE.

First and foremost, we make minimality a primary goal of our system. This was motivated by our application to semantic full-text search. However, minimality is also important for the more universal task of transforming the OIE triples into disambiguated facts within a formal ontology.

Second, we provide a more principled description of how we obtain our sentence constituents, by first transforming the (fine-grained) parse tree into a (coarse-grained) constituent tree, and then treating that tree like an expression tree (with different operators) to obtain what we call our contexts (from which we then derive our triples). In particular, the second step is tricky when relative clauses are nested in enumerations or vice versa. This aspect is not addressed in the description of [7, Section 4.2].

III. CSD AND CSD-IE

We describe our system CSD-IE. The main idea behind CSD-IE is *contextual sentence decomposition* (CSD). CSD is performed in two steps. First, basic building blocks of our contexts are identified in the *sentence constituent identification* (SCI) phase. A tree expressing the semantics is derived. In the second step, *sentence constituent recombination* (SCR), the tree constituents are combined to form our contexts. To derive triples from resulting contexts we introduce a (relatively trivial) third phase. Throughout this section we use the running sentence (S) from Section I.

The next sub-sections define the conceptual ideas of SCI and SCR. We describe an implementation of SCI based on constituent parse trees in section III-C. The triple generation phase is described in section III-D.

A. Sentence Constituent Identification

The task of SCI is to identify the basic "building blocks" of our contexts in a sentence and arrange them in a tree. It turns out that, for our purposes, mainly *relative clauses* and what we call *enumeration items* are important, because they usually contain separate facts that have no direct relationship to the other parts of the sentence. In our sentence (S) from above, the (reduced) relative clause *daughter of the actress and writer Ana Maria Bueno* refers to *Ruth Gabriel* but has nothing to do with the rest of the sentence. Furthermore, the relative clause contains the nominal (pre-)modifier *the actress and writer* modifying *Ana Maria Bueno*, which we consider to be another type of relative clause. The nominal modifier itself contains an enumeration of two coordinated *enumeration items*: *the actress* and *writer*. These have nothing to do with each other, except that they both refer to Ana Maria Bueno.

More specifically now, SCI computes a tree with the following types of nodes:

ENUM: an enumeration. Each child corresponds to an enumeration item and belongs to a different context.

CONC: a group of child-nodes (constituents) that belong to the same context.

REL: a relative clause with a link to its head (the noun phrase or clause it closer describes).

SUB: a clause or sub-sequence corresponding to a self-sufficient context, e.g., a prepositional phrase describing a complex circumstance representing some fact on its own.

LEAF: a leaf that contains words (terminals) of the sentence.

Nodes can be nested in an arbitrary fashion and arbitrarily deep. Figure 1 depicts an SCI tree of our example sentence.

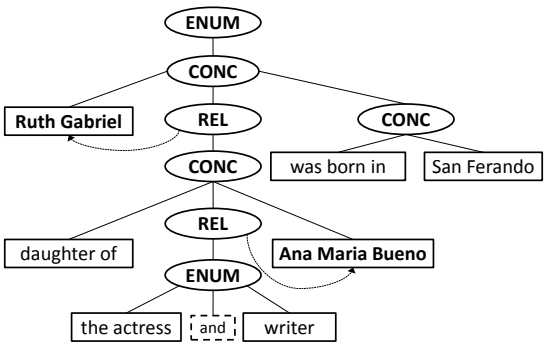

Figure 1. The SCI tree for our example sentence (S). The head of each relative clause is printed in bold, filler words in striped rectangles.

B. Sentence Constituent Recombination

The SCR phase recombines the constituents identified by the SCI phase to form the final *contexts*. SCR recursively computes contexts from an SCI tree or subtree as follows:

(SCR 0) Take out each subtree labeled REL or SUB and change the root of this new tree to CONC. For REL, add the head as the leftmost child (but leave it in the SCI tree, too). For SUB, leave a reference to the newly created tree in the original tree. This is the only place references to other contexts need to be considered. Then process each such subtree and the remaining part of the original SCI tree (each of which then only has ENUM and CONC nodes left) separately as follows:

(SCR 1) For a leaf, there is exactly one context: the part of the sentence stored in that leaf.

(SCR 2a) For an inner node, first recursively compute the set of contexts for each of its children.

(SCR 2b) If the node is marked ENUM, the set of contexts for this node is computed as the *union* of the sets of contexts of the children.

(SCR 2c) If the node is marked CONC, the set of contexts for this node is computed as the *cross-product* of the sets of contexts of the children.

Applying these rules to the SCI tree in Figure 1 yields the desired contexts shown in section I:
In a final step we generate triples from the extracted contexts, see Section III-D. We note that, given the SCI tree and the definition above, SCR is straightforward and fully defined. Therefore, the challenging part of CSD is computing the SCI tree.

C. Sentence Constituent Identification Based on Deep Parse Trees

We present an approach to SCI that is based on the output of a state-of-the-art constituent parser. Figure 2 depicts the parse tree for our example sentence (S).

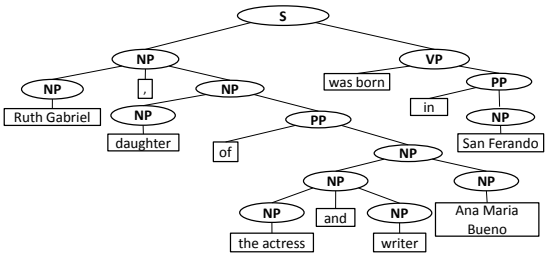

Figure 2. Constituent parse tree for our example sentence. For the sake of readability, the parse tree has been simplified.

We manually created a small set of rules with which we can derive an SCI tree from a parse tree. In the following description when we speak of, for example, an *NP* (noun phrase) we refer to nodes in the parse tree with that tag.

(SCI 1) Mark as ENUM each node, for which the children are all of the same type (e.g. all *VP*), but interleaved by punctuation or conjunctive constructions. (This avoids splitting objects of di- and complex-transitive verbs).

(SCI 2a) If the sequence consist of only two *NPs* split by a comma (and not some conjunction) this is an apposition. Mark the second *NP* as REL and the first *NP* as its head.

(SCI 2b) Mark as REL each *SBAR* and *PP*, if it starts with a phrase in *WHPP*, *WHADV*, *WHNP* or with a word from a positive-list (e.g., *such as* or *who*) but not with one from a negative-list indicating temporal relations (e.g., *before*, *when*).

(SCI 2c) Mark as REL each *PRN* contained in round brackets *"("*, *")"*.

(SCI 2d) Mark as REL each *S* that is preceded by an *NP* followed by a comma.

(SCI 2e) Mark as REL each *VP* that is preceded by a comma, if it has not previously been marked as part of an enumeration and starts with a word with part-of-speech tag of *VBN*, *VBG* or *VBD*. The verb can optionally be preceded by an adverbial phrase *ADVP*.

(SCI 2f) Mark as REL each *VP* below a *WHNP* or *NP*, starting with a word with a part-of-speech tag of *VBN*, *VBG*, or *VBD*. These are participial clauses acting as relative clauses.

(SCI 2g) For all REL from above define the closest left sibling *NP* as the *head*. If there is no left sibling *NP* move down towards the leaves and use the closest first *NP* to the left as head (low/local attachment).

(SCI 3) Mark as REL each *NP*, which has an *NP* as parent and which has exactly one right sibling *NP*. This *NP* is pre-modifying the right sibling *NP*. Mark the right sibling *NP* as head.

(SCI 4) Mark as REL each *NP*, which has as a first (or last) word with part-of-speech tag *PRP$*. This indicates a possessive relation. Mark the last (or first) part of the *NP* as head. If the *NP* also has a right sibling mark it as additional head.

(SCI 5a) Mark as SUB each *PP* starting with a preposition from a positive-list of temporal indicators (e.g., *before* or *while*), each *PP* enclosed in commas and all *PP*s at the beginning of a sentence. These often describe circumstances of events and should be considered separately.

(SCI 5b) Mark as SUB each *SBAR* if it is not already marked REL.

(SCI 5c) Mark as SUB each *S* if it is below an *SBAR* and contains *NP* as well as *VP*, but only if it is not directly below a node of type *PP* or *PRN* (these were treated separately).

(SCI 6) Mark as CONC all remaining nodes. Contract away each CONC with only text nodes in its sub-tree (by merging the respective text) and merge CONC nodes that only have CONC nodes as children.

Applying these rules to the parse tree in Figure 2 produces the SCI-tree displayed in Figure 1. As our quality evaluation in Section IV shows, in general, this small set of rules already works reasonably well.

D. Triple Generation

The triple generation from extracted contexts is (relatively) straight-forward. For each context, we identify the first explicit verb phrase and surrounding adverbs to be the predicate. Everything before that belongs to the subject, and everything after that to the object. For the contexts from our example sentence (S) this gives us the triple #1 from the desired triples shown under 2 in Section I-A.
For relations resulting from SCI 2a-g and SCI 3 rules we add the verb *is* between the head and its attachment, but only if the attachment does not begin with a verb phrase (we then use this verb phrase as predicate). This gives us the remaining triples #2, #3 and #4 for our example sentence from above.

To deal with possessive relations identified by the SCI 4 rule (see section III-C above) we use the predicate *has* between head and attachment.

In some cases the object consists of a list of noun or prepositional phrases (that are not enumeration items), for example in the sentence:

Soubry graduated in law from the University of Birmingham in 1979.

Here, the object consists of three parts: *(in law)*, *(from the University of Birmingham)* and *(in 1979)*. We identify these noun and prepositional phrases in the object when creating our contexts in the SCR-phase, utilizing information from the parse tree. From *n* identified phrases in the object we derive triples, by in turn appending zero or one of the *n - 1* last phrases to the first phrase:

#1: *(Soubry) (graduated) (in law)*
#2: *(Soubry) (graduated) (in law from the University of Birmingham)*
#3: *(Soubry) (graduated) (in law in 1979)*

For *n* identified phrases in the object, this results in *n* triples.

IV. Evaluation

We compare CSD-IE against the three OIE systems ReVerb, OLLIE, and ClausIE. CSD-IE was implemented as described in Section III, using the Stanford Constituent Parser [9]. ClausIE was run in its default mode to extract triples.

We evaluated the systems on two datasets: 200 random sentences from the English *Wikipedia*, and 200 random sentences from the *New York Times*. For comparability, we used the exact same datasets as in [7], which the authors made available on their web site[2]. The labels of their evaluation are also available, however, we chose to label all extracted triples (by all four systems) again, for the following reasons. First, for reasons of consistency (there is always room for interpretation when it comes to assessing accuracy and minimality). Second, to avoid a bias in favor or against a certain system (all extractions were labeled together). Third, because we also wanted to explicitly assess minimality for each extraction.

[2]http://www.mpi-inf.mpg.de/departments/d5/software/clausie.

	ReVerb	OLLIE	ClausIE	CSD-IE
#facts	249	408	610	677
#facts correct	188	230	421	474
prec-a	75.5%	56.4%	69.0%	70.0 %
prec-m	87.2%	80.4%	57.0%	76.8 %
coverage	47.2%	62.7%	95.4%	97.5 %
triple length	7.3	9.7	11.0	8.4

	ReVerb	OLLIE	ClausIE	CSD-IE
#facts	271	358	644	743
#facts correct	177	200	412	531
prec-a	65.3%	55.9%	64.0%	71.5 %
prec-m	93.8%	79.5%	71.4%	90.4 %
coverage	43.8%	61.0%	89.0%	91.8 %
triple length	7.0	10.4	11.6	7.8

Table I
RESULTS OF OUR QUALITY EVALUATION FOR ALL FOUR SYSTEMS FOR THE TWO DATASETS WIKIPEDIA (ABOVE) AND NEW YORK TIMES (BELOW).

The evaluation of [7] also comprises the ReVerb dataset from [6], which consists of 500 random sentences from the Web. However, the relative quality differences between evaluated systems on that dataset were essentially the same. Preliminary experiments on the ReVerb dataset have confirmed this impression, and we therefore excluded it from our (already cumbersome) evaluation.

As explained in Section I, we labeled each extraction (from each of the four systems) with two labels: one for accuracy (yes or no) and one for minimality (yes or no). From these labels, we computed the following accumulated measures for each system:

precision wrt accuracy (prec-a) = the percentage of triples labeled as accurate; see Section I
precision wrt minimality (prec-m) = the percentage of correct triples labeled as minimal; see Section I
coverage = the percentage of word occurrences that occur in at least one extracted triple for that system
average triple length in words = average length of extracted triples for that system in words (ignoring special characters)

Table I shows the results for the two datasets. We note that the numbers for both datasets closely agree with those reported in [7], confirming the reasonability of our labels. We first discuss the results for the Wikipedia dataset (Table I, upper part). From all systems CSD-IE extracts the largest number of facts overall (*#facts*), as well as the largest number of correct facts (*#facts correct*). It also provides the highest coverage of all systems (*coverage*). The precision wrt accuracy (*prec-a*) and coverage of CSD-IE is comparable

to that of ClausIE, but our extracted facts are shorter on average (*triple length*). Furthermore, the precision wrt minimality (*prec-m*) is 20% higher - a drastic improvement. This is because we explicitly consider minimality in our approach and, as a result, our extracted facts are shorter. ReVerb, the only system not utilizing a deep (constituent or dependency) parse, uses patterns that match relatively short facts. These seem to be very accurate, resulting in the best precision and precision wrt minimality scores. In contrast, coverage and the number of extracted facts are not nearly as good as those of CSD-IE or ClausIE. OLLIE improves upon ReVerb by extracting more facts and providing higher coverage. However, overall precision drops drastically. This has also been observed before in [7]. A common problem for deep-parse-based OpenIE systems is the large influence of parser errors. CSD-IE uses the Stanford Constituent Parser [9], OLLIE uses the MaltParser [10] and ClausIE uses the Stanford Dependency Parser [9]. One possible reason for the comparably low precision of OLLIE might be that the MaltParser was trained on data from a different domain and had problems with the sometimes ungrammatical Wikipedia sentences.

The results for the New York Times dataset, (Table I, lower part) are similar to those of the Wikipedia dataset. Sentences are typically longer and deeply nested, causing more broken parses, resulting in slightly worse coverage for all systems. The sentences also contain more facts - almost all systems extract a slightly larger number of facts. Because indirect speech and nested facts are very common in news articles, many triples extracted by CSD-IE contain references to other facts and are therefore more compact and minimal.

A preliminary investigation of errors for CSD-IE on both datasets revealed that most of the inaccurate extractions were caused by mistakes in the parse trees. We consider this an important direction for future research, see the next section.

V. Conclusions and Future Work

We have presented CSD-IE, a new system for open information extraction (OIE). CSD-IE is based on contextual sentence decomposition (CSD), a technique originally developed for semantic full-text search (SFTS). Our evaluation has shown that CSD-IE simultaneously achieves good precision, high recall and very good coverage and minimality. The aspects of coverage and minimality are particularly important for applications such as semantic search, where precise facts and small information loss are desirable. Our work has raised some interesting directions for future research.

A preliminary error analysis shows that most inaccurate extractions are due to a mistake in the (constituent) parse. Since parsing is a complex problem, it would be interesting to make the sentence constituent identification (SCI, see Section III-A) more robust against errors in the parse. Or, alternatively, pre-process the sentence, such that certain common errors are avoided.

A more ambitious plan would be to avoid the "detour" via a deep parse altogether, and try to identify the sentence constituents in a more direct manner. Since our SCI tree III-A is significantly more coarse-grained than the original parse tree, there is hope that this problem can be solved more efficiently and with higher quality.

A common phenomenon in more complex sentences is that of *transitivity*. For example, consider the sentence

> *The ICRW is a non-profit organization headquartered in Washington.*

The following two triples would be considered both accurate and minimal in our approach:

#1: *(The ICRW) (is) (a non-profit organization)*
#2: *(a non-profit organization) (headquartered) (in Washington).*

However, triple #2 would be significantly more informative, if the S part were replaced by *The ICRW*, that is, the concrete entity to which it actually refers. It would be desirable to deal with transitivity of this and more complex kinds as a part of the OIE process.

References

[1] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, "Open information extraction from the web," in *IJCAI*, 2007, pp. 2670–2676.

[2] H. Bast, F. Bäurle, B. Buchhold, and E. Haussmann, "Broccoli: Semantic full-text search at your fingertips," *CoRR*, 2012.

[3] Mausam, M. Schmitz, S. Soderland, R. Bart, and O. Etzioni, "Open language learning for information extraction," in *EMNLP-CoNLL*, 2012, pp. 523–534.

[4] F. Wu and D. S. Weld, "Open information extraction using wikipedia," in *ACL*, 2010, pp. 118–127.

[5] O. Etzioni, A. Fader, J. Christensen, S. Soderland, and Mausam, "Open information extraction: The second generation," in *IJCAI*, 2011, pp. 3–10.

[6] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *EMNLP*, 2011, pp. 1535–1545.

[7] L. D. Corro and R. Gemulla, "Clausie: clause-based open information extraction," in *WWW*, 2013, pp. 355–366.

[8] E. Haussmann, "Contextual sentence decomposition with applications to semantic full-text search," Master's thesis, University of Freiburg, July 2011.

[9] D. Klein and C. D. Manning, "Accurate unlexicalized parsing," in *ACL*, 2003, pp. 423–430.

[10] J. Nivre, J. Hall, and J. Nilsson, "Memory-based dependency parsing," in *Proceedings of CoNLL*, 2004, pp. 49–56.

More Informative Open Information Extraction via Simple Inference

Hannah Bast and Elmar Haussmann

Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast, haussmann}@informatik.uni-freiburg.de

Abstract. Recent Open Information Extraction (OpenIE) systems utilize grammatical structure to extract facts with very high recall and good precision. In this paper, we point out that a significant fraction of the extracted facts is, however, not *informative*. For example, for the sentence *The ICRW is a non-profit organization headquartered in Washington*, the extracted fact *(a non-profit organization) (is headquartered in) (Washington)* is not informative. This is a problem for semantic search applications utilizing these triples, which is hard to fix once the triple extraction is completed. We therefore propose to integrate a set of simple inference rules into the extraction process. Our evaluation shows that, even with these simple rules, the percentage of informative triples can be improved considerably and the already high recall can be improved even further. Both improvements directly increase the quality of search on these triples.[1]

1 Introduction

Information extraction (IE) is the task of automatically extracting relational tuples, typically triples, from natural language text. In recent years, the trend has been towards Open Information Extraction (OpenIE), where identifying the predicate and hence the relation is part of the problem. For example, from the sentence

The ICRW is a non-profit organization headquartered in Washington.

the following triples might be extracted:

#1: *(The ICRW) (is) (a non-profit organization)*
#2: *(a non-profit organization) (is headquartered in) (Washington)*

Extracted triples are an important source of information for many information retrieval (IR) systems, in particular in the area of semantic search. For example, systems for the semantic search challenges in the SemSearch 2010/2011 [1] and TREC Entity Track 2009/2010 [2] perform search on triples. A public demo of a search on triple extractions of the ReVerb OpenIE system [3] is available at: http://openie.cs.washington.edu/. Semantic search systems like Broccoli [4] search in triples or triple-like excerpts extracted from the full text as well. All these approaches rely on the usefulness of extracted triples, usually indicated by how much facts were extracted (*recall*) and whether they are correct (*precision*).

[1] A demo of our system is available via http://ad.informatik.uni-freiburg.de/publications

Early approaches to OpenIE focused on extracting triples with high precision but comparably low recall [5]. Later systems focused on improving recall at best possible precision. Newer systems also addressed other important quality aspects, for example in [3] *incoherent* extractions were addressed and [6] considers the *context* of triples. The by far highest recall (at reasonably good precision) was recently achieved with rule-based approaches utilizing grammatical structure, namely ClausIE [7] and CSD-IE [8]. They rely on grammatical rules based on deep parses of a sentence to extract *direct* facts, as e.g., triples #1 and #2 above. The facts are direct in the sense that subject, predicate (possibly implicit) and object are in some form directly connected via a grammatical relation. With respect to these direct facts, the systems achieve almost perfect recall which makes them suitable for a wide range of applications in IR. However, these systems ignore various quality aspects from earlier work.

In this paper, we show that a significant amount of the extracted facts is not *informative*. In the example above, triples #1 and #2 can both be considered correct, but only triple #1 is, by itself, informative. For all practical purposes, the fact that some non-profit organization is headquartered in Washington is useless. This is a serious problem for systems utilizing these triples for search. For example, a search for where the ICRW is headquartered would not be possible to answer from the extracted triples above. An informative extraction that instead can be inferred is:

#3: *(The ICRW) (is headquartered in) (Washington)*

With this, the extracted triple #2 becomes superfluous - all information of the sentence is covered in a precise form in triples #1 and #3. Note that inferring this is only possible while processing the sentence, when individual subjects, objects and their relations are uniquely identified. Afterwards, multiple facts extracted from different sentences mentioning (say) *a non-profit organization* cannot be guaranteed to refer to the same organization.

Based on the observation that this phenomenon is frequent we propose to integrate simple inference into the extraction process of an OpenIE system. Our approach utilizes some of the generic rules used in large scale inference systems, see Section 3. The process is simple and fast, only uses few inference rules and already shows good results. We provide a brief overview of related work in the next section, describe our approach in Section 3 and provide an evaluation in Section 4.

2 Related Work

For an elaborate overview of recent OpenIE systems we refer to [7]. To the best of our knowledge no existing OpenIE system addresses the issue of inference during its extractions process. Some earlier systems, e.g. [9], extract "indirect" facts, similar to inferred facts, using learned patterns. This only works if the text pattern learned for extraction is part of the used training set. In contrast, our inference rules are generic and independent of the exact text surface of a relation.

A lot of work on inferring new information from triples or knowledge bases exists, e.g. [10, 11]. The goal is usually to infer facts from triples extracted from different sources in order to, for example, extend knowledge bases or perform question answering. Our goal is to improve the informativeness of extracted triples in the first place, not

to perform elaborate inference. Furthermore, as we argued in Section 1, some information can only be inferred while extracting triples, and is irrevocably lost afterwards.

Informativeness of extracted triples has previously been addressed in [3]. Triples were considered uninformative if they omit critical information, for example, when relation phrases are underspecified. The informativeness of extracted triples was evaluated as part of correctness, i.e. uninformative triples were labeled incorrect. We take this one step further and consider a triple uninformative if there is a more precise triple that should be extracted instead, e.g., if the subject should be different (as in the example in Section 1). In our evaluation we explicitly label informativeness as well as correctness of extracted triples.

3 Simple Inference for OpenIE

Our approach consists of three straightforward steps, which are comparably simple yet effective (as our quality evaluation in section 4 shows). The steps are performed after subjects, predicates and objects of all triples in a sentence have been identified, but when all other information (underlying parse tree, supporting data structures etc.) is still available. Given the predicate of each triple in a sentence we first classify the predicate into one of several *semantic relation classes*. Based on the semantic relation we apply a set of inference rules to derive new triples. In the final step we remove existing triples that we consider uninformative depending on whether and how they were used to derive new triples. The next subsections each describe one of the three steps.

3.1 Identifying Semantic Predicate Class

We first classify the predicate of each triple into one of five semantic relation classes shown in Table 1. The relations have previously been successfully used for inference [10] and allow deriving generic, domain-independent inference rules.

To identify the relations we match simple indicator words and patterns. The patterns are implemented as regular expressions over text or parse tree fragments using Tregex [12].

Table 1. Semantic relation classes and patterns for identification

	Semantics	Pattern
SYN	synonymy	*is, was, has|have been, are, nicknamed, known as*
IS-A	hyponymy	*(has|have been, are, is) a|an*
PART-OF	meronymy	*part of, consist* of*
IN	containment or placement	** in*
OTHER	all other relations	*

3.2 Inferring New Triples

Given the triples with identified semantic predicates we infer new triples using a set of generic inference rules. Table 2 shows the rules used. For our example from the

introduction, the last rule matches because the semantic relation IS-A holds between *The ICRW* (A) and *a non-profit organization* (B) and C can be bound to *Washington*. As a result, it is inferred that *The ICRW is headquartered in Washington*.

These rules are similar to the up-ward monotone rules from [10], but have been extended with an additional rule to reason over IS-A relations. The implementation differentiates between lexically identical subjects and objects that occur in different places of a sentence. This is a fundamental difference to approaches inferring information after triple extraction, where this information is no longer available.

Table 2. Inference rules for new triples

$\textit{OTHER}(A', B)$	$\leftarrow \textit{OTHER}(A, B) \wedge \textit{SYN}(A, A')$
$\textit{OTHER}(A', B)$	$\leftarrow \textit{OTHER}(A, B) \wedge \textit{SYN}(A', A)$
$\textit{OTHER}(A, B')$	$\leftarrow \textit{OTHER}(A, B) \wedge \textit{SYN}(B, B')$
$\textit{OTHER}(A, B')$	$\leftarrow \textit{OTHER}(A, B) \wedge \textit{SYN}(B', B)$
$\textit{IN}(A, C)$	$\leftarrow \textit{IN}(A, B) \wedge \textit{PART-OF}(B, C)$
$\textit{IN}(A, C)$	$\leftarrow \textit{IN}(A, B) \wedge \textit{IS-A}(B, C)$
$\textit{OTHER}(A, C)$	$\leftarrow \textit{IS-A}(A, B) \wedge \textit{OTHER}(B, C)$

Table 3. Rule for deleting triples

$$remove(\textit{OTHER}(B, C)) \leftarrow \begin{array}{r} \textit{IS-A}(A, B) \\ \wedge \textit{OTHER}(B, C) \\ \wedge \textit{OTHER}(A, C) \end{array}$$

3.3 Removing Uninformative Triples

As described in Section 1 some triples become redundant after they were used to infer additional information. These triples should not be part of the output of the system. This is often the case for IS-A relations and we use a single rule shown in Table 3 to remove triples from our result list.

4 Evaluation

We evaluate the quality of extracted triples with respect to correctness and informativeness. A system similar to the OpenIE system in [8] was used to integrate inference as described above. We compared it against the OpenIE system without inference.

As dataset we used 200 random sentences from Wikipedia. The sentences contain only few incorrect grammatical constructions and cover a wide range of complexity and length. This is the exact same dataset that has already been used in [7].

For each extracted triple we manually assigned two labels: one for correctness (yes or no) and one for informativeness (yes or no). We follow the definition of [5] and consider a triple correct if it is consistent with the truth value of the corresponding sentence. A correct triple is considered informative if there is no extraction that is more precise, according to the sentence it was extracted from. For example, in the sentence from the introduction, triples #1 and #2 would be considered correct, but only triple #1 would be considered informative and triple #3 would be considered both, correct and informative. From the labeled triples we calculated precision of correct triples and estimate recall using the number of extracted correct triples. We also calculated corresponding breakdown statistics for triples that are informative (inf.) as well as correct (corr.). Table 4 shows overall results and Table 5 provides detailed information about the inferred triples.

We first discuss the results in Table 4. Without inference, a large fraction of 10% of correct triples is not informative (prec-corr. inf.). This means that, on average, every 10th extracted correct triple is more or less useless. Using inference the overall number of extracted facts increases from 649 to 762, a relative increase of 17%. The number of correct facts (#facts corr.) also increases: from 429 to 484, corresponding to a relative increase of 13%. The relative increase in correct triples is smaller, because a small number of incorrect triples are inferred (see next paragraph). This is also the reason for the small decrease in the percentage of correct triples (prec corr.) from 66% to 64% (at a 13% higher recall, however). Overall, the number of triples that are both correct and informative (#facts corr. + inf.) increases from 385 to 444: a 15% increase. This is a major improvement, caused by the large number of correct informative triples that were inferred and the uninformative triples removed. Correspondingly, the percentage of correct triples that are also informative (prec-corr. + inf.) increases from 90% to 92%.

Table 4. Quality evaluation results with inference (top row) and without inference (bottom row) over the labels correct (corr.) and informative (inf.). *prec corr.* refers to the percentage of all triples labeled correct, *prec-corr. inf.* to the percentage of correct triples labeled informative.

	#facts	#facts corr.	#facts corr. + inf.	prec corr.	prec-corr. inf.
No Inference	649	429	385	66%	90%
Inference	762	484	444	64%	92%

Table 5. Detailed statistics for inferred triples. *prec inf.* refers to the percentage of inferred triples labeled correct and *prec-corr. inf.* to the percentage of inferred correct triples also labeled informative.

#inferred	#inferred corr.	prec inf.	#inferred corr. + inf.	prec-corr. inf.
127	69	54%	59	85%

Table 5 shows the statistics for inferred triples. Note that, as described in Section 3.3, during inference previously extracted triples may be removed. Therefore, the number of extracted facts with inference does not equal the sum of facts extracted without inference and inferred facts (see Table 4). Overall, about 54% of inferred triples are correct (prec. inf.). A preliminary investigation shows that this is mainly caused by mistakes in preceding phases, in particular wrong parses, wrong identification of objects or predicates and wrong mapping of predicates to their semantic class (see Section 3.1). Eliminating these errors should be part of our next steps (see Section 5). About 85% of correct inferred triples are also informative (prec-corr. inf.). Closer analysis shows that, due to inference, 32% of the triples that were correct but uninformative were removed and replaced with informative triples. Together with the inferred triples, this causes the increase in the percentage of correct triples that are also informative (prec-corr. + inf. in Table 4).

In some cases uninformative triples were inferred. For example, from the sentence *She joined WGBH-TV, Boston's public television station* it is inferred that *(She) (joined) (Boston's public television station)*. Given that our current approach does not differen-

tiate between concrete and abstract subjects and objects (and therefore the "direction" of inference) the high percentage of informative triples derived is remarkable. Further work should, however, try to prevent these extractions.

5 Conclusions

We have presented a simple yet effective way to increase the informativeness of extracted triples for a recent OpenIE system. Using only a few simple inference rules integrated into triple extraction can increase the number of extracted informative triples by 15%. There are a lot of promising directions to improve our work.

A preliminary error analysis shows that most mistakes happen in preceding extraction stages, in particular the precise identification of predicates and objects. Improvements in these areas will likely obviate the small negative effect on precision. To improve the recognition of semantic relations, utilizing existing collections of semantic patterns, such as provided in [13], seems promising. Our manually designed inference rules could be exchanged for automatically derived rules, e.g., as suggested in [14]. Finally, to derive additional facts and distinguish between abstract and concrete facts utilizing information from named entity recognition seems promising.

References

1. Tran, T., Mika, P., Wang, H., Grobelnik, M.: Semsearch'11: the 4th Semantic Search Workshop. In: WWW. (2011)
2. Balog, K., Serdyukov, P., de Vries, A.P.: Overview of the TREC 2010 Entity Track. In: TREC. (2010)
3. Fader, A., Soderland, S., Etzioni, O.: Identifying relations for open information extraction. In: EMNLP. (2011) 1535–1545
4. Bast, H., Bäurle, F., Buchhold, B., Haussmann, E.: Broccoli: Semantic full-text search at your fingertips. CoRR (2012)
5. Banko, M., Cafarella, M.J., Soderland, S., Broadhead, M., Etzioni, O.: Open information extraction from the web. In: IJCAI. (2007) 2670–2676
6. Mausam, Schmitz, M., Soderland, S., Bart, R., Etzioni, O.: Open language learning for information extraction. In: EMNLP-CoNLL. (2012) 523–534
7. Corro, L.D., Gemulla, R.: ClausIE: clause-based open information extraction. In: WWW. (2013) 355–366
8. Bast, H., Haussmann, E.: Open information extraction via contextual sentence decomposition. In: ICSC. (2013)
9. Banko, M., Etzioni, O.: The tradeoffs between open and traditional relation extraction. In: ACL. (2008) 28–36
10. Schoenmackers, S., Etzioni, O., Weld, D.S.: Scaling textual inference to the web. In: EMNLP. (2008) 79–88
11. Lao, N., Mitchell, T.M., Cohen, W.W.: Random walk inference and learning in a large scale knowledge base. In: EMNLP. (2011) 529–539
12. Levy, R., Andrew, G.: Tregex and Tsurgeon: tools for querying and manipulating tree data structures. In: LREC. (2006) 2231–2234
13. Nakashole, N., Weikum, G., Suchanek, F.M.: PATTY: A taxonomy of relational patterns with semantic types. In: EMNLP-CoNLL. (2012) 1135–1145
14. Schoenmackers, S., Davis, J., Etzioni, O., Weld, D.S.: Learning first-order horn clauses from web text. In: EMNLP. (2010) 1088–1098

Relevance Scores for Triples from Type-Like Relations

Hannah Bast, Björn Buchhold, Elmar Haussmann
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast, buchhold, haussmann}@cs.uni-freiburg.de

ABSTRACT

We compute and evaluate relevance scores for knowledge-base triples from type-like relations. Such a score measures the degree to which an entity "belongs" to a type. For example, Quentin Tarantino has various professions, including Film Director, Screenwriter, and Actor. The first two would get a high score in our setting, because those are his main professions. The third would get a low score, because he mostly had cameo appearances in his own movies. Such scores are essential in the ranking for entity queries, e.g. "american actors" or "quentin tarantino professions". These scores are different from scores for "correctness" or "accuracy" (all three professions above are correct and accurate).

We propose a variety of algorithms to compute these scores. For our evaluation we designed a new benchmark, which includes a ground truth based on about 14K human judgments obtained via crowdsourcing. Inter-judge agreement is slightly over 90%. Existing approaches from the literature give results far from the optimum. Our best algorithms achieve an agreement of about 80% with the ground truth.

1. INTRODUCTION

Knowledge bases allow queries with a well-defined result set. For example, we can easily formulate a precise query that gives us a list of all *american actors* in a knowledge base. Note the fundamental difference to full-text search, where keyword queries are only approximations of the actual search intent, and thus result lists are typically a mix of relevant and irrelevant hits.

But even for well-defined result sets, a ranking of the results is often desirable. One reason is similar as in full-text search: when the set of relevant hits is very large, we want the most "interesting" results first. But even if the result set is small, an ordering often makes sense. We give two examples. The numbers refer to Freebase [7], the largest general-purpose knowledge base to date.

Example 1 (american actors): Consider the query that returns all entities that have *Actor* as their profession and *American* as their nationality. On our Freebase dataset, this query has 64,757 matches. A straightforward ranking would be by popularity, as measured, e.g., by counting the number of occurrences of each entity in a reference text corpus. Doing that, the top-5 results for our query look as follows (the first item means the younger Bush):

George Bush,*Hillary Clinton*,*Tim Burton*,*Lady Gaga*,*Johnny Depp*

All five of these are indeed listed as actors in Freebase. This is correct in the sense that each of them appeared in a number of movies, and be it only in documentary movies as themselves or in short cameo roles. However, Bush and Clinton are known as politicians, Burton is known as a film director, and Lady Gaga as a musician. Only Johnny Depp, number five in the list above, is primarily an actor. He should definitely be ranked before the other four.

Example 2 (professions of a single person): Consider all professions by Arnold Schwarzenegger. Our version of Freebase lists 10 entries:

Actor, *Athlete*, *Bodybuilder*, *Businessperson*, *Entrepreneur*, *Film Producer*, *Investor*, *Politician*, *Television Director*, *Writer*

Again, all of them are correct in a sense. For this query, ranking by "popularity" (of the professions) makes even less sense than for the query from Example 1. Rather, we would like to have the "main" professions of that particular person at the top. For Arnold Schwarzenegger that would be: *Actor*, *Politician*, *Bodybuilder*. Note how we have a very-ill defined task here. It is debatable whether Arnold Schwarzenegger is more of an actor or more of a politician. But he is certainly more of an actor than a writer.

1.1 Problem definition

In this paper, we address the following problem, which addresses the issues behind both of the examples above.

Definition: Given a type-like relation from a knowledge base, for each triple from that relation compute a score from $[0, 1]$ that measures the degree to which the subject belongs to the respective type (expressed by the predicate and object). In the remainder of this paper we often refer to these scores simply as *triple scores*.

Here are four example scores, related to the queries above:

Tim Burton has-profession Actor	0.3
Tim Burton has-profession Director	1.0
Johnny Depp has-profession Actor	1.0
Arnold Schwarzenegger has-profession Actor	0.6

An alternative, more intuitive way of expressing this notion of "degree" is: how "surprised" would we be to see *Actor* in a list of professions of Johnny Depp. We use this formulation in our crowdsourcing task when acquiring human judgments.

Note that the "degree" in the definition above is inherently ill-defined, much like "relevance" in full-text search. In particular, different users might have different opinions on the correct "degree" (for example, on the four scores from above). However, it is one of the results of our crowdsourcing-based experiments that there is

SIGIR'15, August 09 - 13, 2015, Santiago, Chile.

DOI: http://dx.doi.org/10.1145/2766462.2767734.

broad general consensus. Note that we do not take user preferences into account in this paper; we consider this an interesting topic on its own.

1.2 Variants and related problems

The reader may wonder about variants of the problem definition above, and related problems from the literature. We briefly discuss these here.

Degree of Belonging vs. Correctness vs. Accuracy

Our scores are different from correctness or confidence scores. For example, in knowledge fusion, multiple sources might provide conflicting information on the birth date of a person. In those situations we need assessments on the confidence of a particular information and which information is probably correct [12].

Our scores are also different from scores that measure the accuracy. For example, an estimate for the population of a country might be off by a certain margin but not completely wrong, or only a range for the value is given.

We consider a single knowledge base with non-conflicting and precise information. Indeed, for a knowledge base like Freebase, over 99% of all triples are reported to be correct [7]. Even if there are incorrect triples, it is a pleasant side effect of our framework that they would get low scores.

Graded scores vs. Binary scores

We started our research on this topic with binary scores, which we called *primary* and *secondary*, in accordance with existing research on graded relevance in full-text search.[1] We later found that both our crowdsourcing experiments and our algorithms naturally provide finer grades that also make sense in practice. We hence use a continuous score in our definition above. The range $[0, 1]$ is merely a matter of convention.

In our crowdsourcing tasks in Section 3 we still ask each judge for only a binary judgment. However, aggregation of multiple judgments for each triple leads again to a graded score.

Which kind of relations

In principle, our algorithms work for all relations. However, we found that for relations other than type-like, the kind of scores we are interested in are either trivial or can be obtained using other, simpler methods. We distinguish three types of relations.

- Functional relations like *birth date* or *weight*, where each subject has exactly one object. Since we assume a single knowledge base where (almost) all triples are correct, we could simply assign a score of 1.0 to all such triples.

- Non-functional relations between two concrete entities, like *acquired* (between two companies) or *acted in* (between an actor and a movie). For such triples there are two simpler options for a good relevance score, which we do not explore further in this paper. The first option is to take the value from another relation of the knowledge base, like the date of the acquisition or the rank of the actor in the cast listing. This relation could be assigned manually (the number of relations in a knowledge base is relatively small). Or it could be determined heuristically (interesting, but technically completely different from the methods we present). The second option is to adapt our method for type-like relations algorithms also for these relations. The main challenge for finding witnesses in the full text is then to recognize variations of the (fixed) predicate name and not of the (varying) object name. This makes our problem much easier; see Section 4.

[1]We assume that all triples in our knowledge base are correct, so there are no triples expressing a relationship that is *not relevant*.

- Non-functional relations between an entity and an abstract group or type, like *profession* (between a person and a profession) or *nationality* (between a person and a nationality). We know of no knowledge base with explicit values for the "degree" to which an entity belongs to a certain type. In particular, this is true for *profession* and *nationality*. As we will see in Section 4, such relations are also the hardest for our approaches, because they require a separate classifier for each distinct type (e.g., each distinct profession and nationality), instead of just one for the whole relation. This is why we selected only type-like relations for our benchmark.

We further remark that type information is central to many entity queries. For example, all three tasks from the TREC 2011 Entity Track [2] ask for lists of entities of a particular type. Also, most of the entity queries currently supported by Google are for entities of a certain type, like in the two use cases from our introduction; presumably because of their popularity amongst users.

Triple Scores vs. Full Ranking

The reader may ask why we study scores for individual triples when the motivation is the ranking of results for entity queries. We give two main reasons.

First, the triple scores we consider are a crucial component of any such ranking, and the problem by itself is hard enough to warrant separate study. In fact, in Section 2 we discuss related works on ranking for entity queries that require such scores as input.

Second, for many popular entity queries (like our two use cases from the introduction), our triple scores are basically all that is needed to obtain a good ranking. For the first use case ("american actors"), ranking is a simple matter of combining our triple scores with a popularity measure for each entity. Popularity measures are easily obtained, e.g., by counting the number of mentions in a reference corpus. For the second use case ("professions of a single person"), triple scores are exactly what is needed. Note that for the second use case, a proper ranking is still missing on Google.

1.3 Basic Idea and Challenges

Our basic idea is to find "witnesses" for each triple in a given large text corpus. The more witnesses we find and the more significant they are, the larger the score for that triple.

For example, consider the *profession* relation. In a "learning" step, we compute words that are associated with each profession. For example, for the profession *Actor* these could be: actor, actress, film, cast. For each entity (e.g., *Johnny Depp*) we identify occurrences of these words semantically related to that entity in the text (e.g., *Paramount decided to cast Depp ...*). Then we compute weighted sums of the number of such occurrences for each possible profession. In a final step, we normalize these sums to obtain our scores. This sounds simple, but there are a number of challenges:

- Learning must be unsupervised, given that we have to learn different words for every different possible "type". For example, there are more than 3,552 different professions in Freebase.

- When is a word "semantically related" to an entity in a text? It turns out that considering co-occurrence in the same sentence works, but that a deeper natural language processing helps.

- How much weight to give to each occurrence of a word? It turns out that results can be boosted by graded scores for different words.

- How to get from weighted sums to normalized scores? It turns out that a simple linear mapping works but is often not optimal.

- Why use text and not the knowledge base? For a long time, we also experimented with other facts from the knowledge base as complementary or even as the sole information source. For example, the fact that someone is or once was president of the U.S.

implies a high score for the profession Politician. However, we found this knowledge to be consistently less available (especially for less popular entities) and nowhere more reliable than full text. More details about this from our experiments in Section 5.4.

1.4 Contributions and Overview

We consider the following as our main contributions:

- We identify the computation of relevance scores for triples from type-like relations as an interesting research problem, which so far has not achieved much attention. These scores are essential for properly ranking many popular kinds of entity queries.
- We designed and make available a first benchmark for this problem. The required large number of relevance judgments (over 14K) were obtained via crowdsourcing. Given the hard-to-define nature of our scores, designing the task such that untrained human judges provide proper and consistent judgments was a challenging task on its own. Inter-judge agreement is about 90%, which confirms the soundness of our problem definition. See Section 3.
- We introduce a variety of methods (partly new, partly adapted) to compute these scores. All our methods work with an arbitrary knowledge base and text corpus. See Section 4.
- We implemented and evaluated all methods against three baselines, using Freebase as knowledge base and Wikipedia as text corpus. Our best methods achieve an agreement of about 80% with the ground truth. We considered many variants and ideas for improvement, none of which gave better results. See Section 5.
- We make our evaluation benchmark as well as all our code publicly available (see Section 5). In particular, this allows full reproduction of our results.

2. RELATED WORK

There exists a large amount of work about ranking and other problems based on the kind of relevance scores we study here. However, in all these works such scores are assumed to be given. We know of no work that addresses how to compute these scores in the first place. We give a short overview of the various approaches.

Triple Scores for Ranking Structured Queries Several pieces of work deal with integrating scores in a framework for ranking structured queries. For example, in [8], the authors propose an extension to RDF they call Ranked RDF, [13] proposes a ranking model for SPARQL queries with possible text extensions based on Language Models, and [11] discusses how to combine several kinds of scores associated with triples into a meaningful ranking. In all these frameworks, scores that are similar to our triple scores are assumed to be given.

Fuzzy Databases / Knowledge bases This is an old and well-established research area, considering all kinds of "fuzziness" in an information system, including: uncertainty, imprecision, and fractional degrees of membership in sets; see [20] for a survey. However, this body of work is almost entirely about modeling this fuzziness, and how to solve standard tasks (like queries or reasoning) based on fuzzy information. The fuzzy information itself is, again, assumed to be given. We know of no work in this area, where fuzzy scores of the kind we consider are actually computed.

Ranking for Relational Databases SQL allows explicit ordering of the query results (using the *order by* keyword). Still, there are many meaningful queries to databases that return a huge number of results. When confronting users with those results, ranking plays an important role. This problem is tackled in [9]. Apart from the many-answers problem, ranking for databases becomes important when adding a keyword-search component. BANKS [5] has the goal of ranking possible interpretations of a keyword query. Similarly, ObjectRank [1] also takes keyword queries, but ranks "database objects" (e.g., a paper, an author, etc.) according to a query. These approaches share the fact that they work exclusively on the data (or knowledge) base. Going from there, they try to use the structure induced by foreign keys to provide a ranking. For our specialized use case, this is not well suited. Even in a perfect world, where such an approach is able to find the perfect connections to influence the ranking (a hard task in itself), it is restricted to data in the knowledge base. However, as discussed in Section 1.3, we have found structured knowledge to be less available than that from full text for computing our relevance scores.

Ranking Semantic Web Resources In the Semantic Web, one is confronted with data from many differences sources, and of highly varying quality. This gives rise to the problem of ranking these data sources with respect to a given query or topic. For example, TripleRank [16] achieves this by representing the Semantic Web as a 3D tensor and then performs a tensor decomposition. Despite the related-sounding name, this is very different from our scenario, which is about ranking of entities from a single knowledge base.

Topic Models On a technical level, some of our methods are related to *topic models*. These derive high-level topics of documents by inferring important words for each topic and the topic distribution for each document. In our setting, a document could be seen as a subject (e.g., person) and topics as different types (e.g., her professions). One state-of-the-art unsupervised topic model is Latent Dirichlet Allocation (LDA) [6]. LDA (and related methods) infer topics given only the text as input. In a supervised extension called Labeled LDA (L-LDA) [21], topic labels (e.g., our professions) can be provided for each document as part of the input. We compare our methods against L-LDA in Section 5.

3. ESTABLISHING A BENCHMARK

Ranking problems and the notion of relevance are inherently subjective and vague. Human relevance judgments are not only necessary to evaluate algorithms that attempt to solve the problem, but also help understanding it. This has happened for keyword search, where ranking became an established problem with a universally shared understanding.

We decided to use crowdsourcing to create a suitable benchmark for our triple scores for exactly these two reasons: evaluation and problem refinement. Since this helps to understand the problem at hand, we discuss our benchmark before we describe our algorithms to solve the problem.

Recall our use case example from Section 1. We want humans to judge to what extent a person has a certain profession. In this section, we discuss how we select a representative sample of entities, present two ways to set up the task, and explain why one is superior. We then analyze the result of the crowdsourcing experiment. The ground truth files are available for download together with all reproducibility material under the URL given in Section 5.

3.1 Selecting Persons for Evaluation

Our benchmark should feature a representative sample of persons that contains all levels of popularity. This is important for two reasons. First, it can be expected that more information is available for popular persons, so that some approaches might work better (or even only) for those. Second, as discussed in the motivation, while the ranking problem we address also exists for less popular persons, it is more pronounced for popular persons.

Selecting persons from Freebase with more than one profession leaves us with about 240K persons. We use the number of times

a person is mentioned in Wikipedia to approximate popularity and restrict our choice to persons having Wikipedia articles. This has practical reasons. In principle, any text collection could be used, but Wikipedia is easy to obtain and, due to hyperlinks, no hard Entity Linking problem has to be solved. Apart from that, we can point human judges directly to the Wikipedia article and hence enable them to make an informed decision without much effort. We observe that (as could be expected) there are lots of people with none or very few mentions and few people with lots of mentions (up to almost 100K mentions). Therefore, a random sampling would almost exclusively contain unpopular entities.

We define buckets of popularity and take a uniform random sample from each bucket. The buckets used were for a popularity (number of mentions) $< 2^4$, between 2^i and 2^{i+1} for $4 \leq i \leq 13$, and $\geq 2^{14}$. In total, we took about 25 samples from each of the 12 buckets. This left us with 298 persons or a total of 1225 person-profession tuples that nicely cover different levels of popularity.

3.2 Evaluate a Single Profession of a Person

An obvious approach is to present a person and his or her profession to a judge and ask whether the profession is primary or secondary for that person. For example:

Arnold Schwarzenegger and the profession *Bodybuilder*. Is the profession primary or secondary for this person?

The task was enriched with instructions, including definitions of the factors listed above. However, this definition is extremely hard to communicate precisely. Also, there is a lot of subjectivity involved: what "feels" primary to one judge does not to another.

Eventually, it turned out that judges mostly resorted to the following strategy: label the profession that is first mentioned in Wikipedia as primary, and all others as secondary. Indeed this is one of our simple baselines in Section 5. Obviously, this cannot work for persons with more than one primary profession.

3.3 Evaluate All Professions of a Person

An improved version that turned out to work well is the following. Instead of labeling a single profession of a person, we ask to label all professions of a person. Additionally, we provide simpler instructions but enrich them with a set of diverse examples of labeled persons. An example is depicted in Figure 1. All professions of *Barack Obama* must be dragged into the box for primary or secondary professions.

Figure 1: Condensed illustration of the crowdsourcing task. All professions must be dragged into the box for primary or secondary professions. For the complete version including instructions and examples see, go the URL from Section 5.

Below that, we show four examples illustrating a diverse set of possible labelings: Michael Jackson (many professions, some primary some secondary), Ronald Reagan (many professions, two of them primary), Mike Tyson (three professions, only one primary), James Cook (two professions, both primary).

This worked well, and we assume this is mainly due to two reasons:

(1) People could do research on the person, and thus judge all the professions of that person in relation to each other.

(2) The diverse examples helped to form an intuitive theory of what we expected.

3.4 Crowdsourcing Results

We had the professions of each person labeled by seven different judges. As final score for each triple we sum up the number of primary labels, which is then in the range of 0 (all judges label secondary) to 7 (all judges label primary).

On the 298 persons (1225 person-profession tuples) the inter-annotator agreement according to Fleiss' Kappa [15] is 0.47 for the binary judgments. This can be considered moderate agreement. However, what our experiment actually does is to determine a hidden probability p with which a human judge finds a person's profession to be primary. In fact, we use a binary experiment to get gradual scores. Note that, for certain triples, the desired result may actually be a probability around 0.5. Measuring inter-annotator agreement does such a scenario no justice. Instead, we want to make sure that (1) all judges base their decision on the task and the data, and (2) the obtained probabilities are significantly different from random decisions.

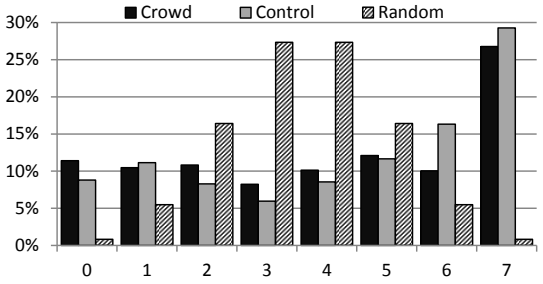

Figure 2: Histogram of score distribution of our crowdsourcing task, the control run, and expected results for randomly (with $p = 0.5$) guessing judges (rounded).

For (1), we have run the same task again (with different annotators) on a subset of every third person (386 person-profession tuples) from our initial sample. Figure 2 depicts how much the scores differ between this control run (*control*) and the full run (*crowd*).

For (2), looking at the distribution of scores shows that they are very far from a random decision. Figure 2 shows a distribution that prefers border cases (especially primary professions) and contains only reasonably few triples with average score (3 or 4). In contrast, a random or unclear task would lead to a distribution with mostly average scores and rare border cases.

In Section 5, we report more details on the results of the control run. The human judges based 95% of their decisions only on the Wikipedia page of the respective person. As we report in Section 5.4, our automatic methods perform much worse when we restrict them in this way. This shows how hard the judgment task is: the relative relevance of the various professions mentioned on a person's Wikipedia page is often highly implicit in the text.

4. COMPUTING TRIPLE SCORES

We propose a variety of algorithms to compute triple scores. For illustration purposes we describe the algorithms using the *profession* relation of a person, but the algorithms are not specific to that relation and should work with any type-like relation. In Section 5 we show that they are equally effective to derive scores for nationalities of persons. Likewise, we use Wikipedia as our source of text and Freebase as knowledge base, but the approaches are not specific to that data. None of our algorithms requires manually labeled data. Instead, we make use of existing facts in an unsupervised or distantly supervised fashion. The crowdsourcing judgments are only used for evaluation.

The different algorithms provide different output: binary classifications, weighted sums and probabilities. In Section 4.6, we describe how we map their output to the range $[0, 1]$.

4.1 Training Data

We want to avoid manually labeled training data. Instead, we use the following definition to obtain positive and negative training examples for each profession:

Positive: people with only the given profession or any specialization of it, i.e. those for which the profession has to be primary.
Negative: people that do not have that profession at all

Thus, Humphrey Bogart is a positive example for the profession *Actor* (because this is his only profession according to Freebase) and Barack Obama, who is listed as politician, lawyer and more (but not as actor), is a negative example for that profession.

We also experimented with other criteria (e.g., allowing the persons used as *positive* examples to also have other professions) but found this selection to work best across all approaches.

4.2 Selecting Text for each Person

All approaches analyze text about a person to derive triple scores. As text we use the English Wikipedia[2], utilizing the fact that persons in Freebase often have a link to their Wikipedia article. We have pre-processed Wikipedia and performed Entity Recognition and Linking, as well as anaphora resolution as described in [3]. With each person we associate all words that co-occur with a linked mention of the person in the same *semantic context*, as provided by the pre-processing. Each semantic context is a sub-sequence of the containing sentence that expresses one fact from the sentence. This yields slight improvements over using full sentences (where numerous facts can be mixed) as contexts. Further, we have found that stemming has almost no effect. These and other variations are discussed in more detail in Section 5.4.

4.3 Binary Classifier per Profession

We train a binary classifier to decide whether a profession is primary or secondary for a given person. For each person we extract features from her associated text. We use word counts as features, which we normalize by the total number of word occurrences in the person's associated text. Thus, feature values are between 0 and 1 (much closer to 0). Weighting feature values by their tf-idf score (instead of the normalization) had no positive effect.

Before training the classifier, we balance positive and negative training examples by randomly selecting an appropriate subset of the negative training examples (there are always more negative than positive training examples).

A logistic regression classifier with L2-regularization is then trained on the balanced training data. As a linear model, logistic regression has the benefit of an intuitive interpretation of features weights. Table 1 lists the features with top weights learned for profession *Actor*.

[2]downloaded in August 2013

1. cast	28.21	5. directed	-6.64	9. university	-5.16
2. actor	12.46	6. starring	6.22	10. written	-4.97
3. actress	11.50	7. role	5.62	11. voiced	4.50
4. played	7.19	8. stars	5.26	12. actors	4.48

Table 1: Top features (positive and negative) and their weight learned by the logistic regression classifier for the profession *Actor*.

The learned feature weights form a model for each profession. These models can then be used for persons with multiple professions to make a binary decision if each of his or her professions is primary or not (secondary).

4.4 Count Profession Words

We want to mimic the behavior of humans that want to solve our problem. Therefore, we look at the text associated with an entity and find out how big the portion of text is that discusses profession matters. In the simplest way, we could count how often a profession is mentioned in text associated with the person.

We cannot count exact occurrences, however, because of professions that consist of more than one word, e.g., Film Score Composer. Looking for mentions of such professions already puts them at a disadvantage (or advantage depending on full vs. partial match counting) compared to one-word professions like Composer. Additionally, often slight variations of a profession are mentioned, e.g., actor and actress.

To overcome this issue, we define a suitable prefix for each profession. We can do this automatically (using an off-the-shelf stemmer and the longest common prefix of the stem and the original word) or manually once for each profession. In our experiments (Section 5), we compare manually chosen prefixes as a strong baseline.

Besides the profession names themselves (or their prefixes), we have found that there are many other words that indicate, that some part of the text is about a given profession. For example, consider the positive features learned by the binary classifier as presented in Table 1. We extend our counting based approach by automatically computing indicator words in the following way: For each profession, take all words in the associated text for persons in the *positive* training data (see Section 4.1), compute their tf-idf value and rank the words by that value. During prediction, we set the weight of an indicator mention to 1/rank and build the sum over the weights of all indicator mentions.

1. cast	1.00	5. role	0.20	9. television	0.11
2. actor	0.50	6. starring	0.16	10. appeared	0.10
3. actress	0.30	7. played	0.14	11. born	0.09
4. film	0.25	8. best	0.13	12. series	0.08

Table 2: Top words (by tf-idf) and their weight $(1/\text{rank})$ for the profession *Actor*.

Table 2 shows the top words and their weights for the profession *Actor*. There is high overlap with the top features learned by the binary classifier shown in Table 1. Note, that the word weights were computed only from the text associated with *positive* training examples and idf values based on the whole corpus. The *negative* examples were not needed.

4.5 Generative model

We formulate a generative model where each person is associated with text that can be generated as follows. For a person with k professions and n word occurrences in the associated text: Pick one of the k professions with probability $P(p_i)$; generate word w_j with $P(w_j|p_i)$; repeat until all n words are generated. The joint probability for word w and profession p is then $P(w,p) = P(w|p)P(p)$.

Note that for a given text, the professions selected in the first step above are unobserved. We would like to infer those, because, intuitively, they represent the amount of text that can be associated with a certain profession. We derive these using a maximum likelihood estimate.

Let tf_j be the term frequency of word j, $P(p_i)$ be the probability of profession i and let $P(w_j|p_i)$ be the probability for word j under profession i. The log-likelihood of producing text consisting of n words for k professions is:

$$\log \mathcal{L} = \sum_{j=1}^{N} [tf_j \cdot \log \sum_{i=1}^{k} (P(w_j|p_i)P(p_i))]$$

Training We use the *positive* training examples to derive word probabilities $P(w|p)$. We distinguish two ways. (1) use the term frequency of terms in the text associated with all training examples and assign probabilities accordingly. (2) use the tf-idf values as term frequencies to avoid common, unspecific terms with high probabilities. When using tf-idf values, we use them for both: $P(w|p)$ and as new tf_j when calculating LP. For efficiency reasons, we only keep the probabilities for the top 30K words. We have found that tf-idf values work better and restrict our examples to this setup for the remainder of the paper.

Text that is associated with a person often contains information that is not related to professions, e.g., family and personal life. Therefore, we add a pseudo profession to each person in order to account for these words. We use all text in the English Wikipedia to calculate $P(w|p)$ for the pseudo profession.

1. cast	0.023	5. role	0.007	9. appeared	0.004
2. actor	0.009	6. starring	0.005	10. television	0.004
3. actress	0.008	7. played	0.005	11. born	0.004
4. film	0.008	8. best	0.004	12. roles	0.004

Table 3: Top word probabilities for the profession *Actor*.

Naturally, the top word probabilities depicted in Table 3 are in line with the top word weights presented in Table 2. This is not surprising, since both values are based on the tf-idf scores of words in text associated with the *positive* training samples. The probabilities, however, do not differ as much as the weights we have assigned for the counting approach.

Prediction To derive profession probabilities $\vec{p}$, which have to sum to 1, we maximize the log-likelihood:

$$\vec{p} = \arg\max_{\vec{p}} \sum_{j=1}^{N} [tf_j \cdot \log \sum_{i=1}^{k} (P(w_j|p_i)P(p_i))], \text{s.t.} \sum_{i=1}^{k} p_i = 1$$

To find the maximum likelihood estimate we use Expectation Maximization [10]. Similar to the generative model for pLSI [17] we treat the topic (profession) assignments to word-occurrences as hidden variables. EM iteratively computes posterior probabilities given the hidden variables in the expectation step (E) and then maximizes parameters for the previously computed probabilities in the maximization step (M). The E and M steps are identical to the steps in [17], with the difference that we treat $P(w|p)$ as fixed, because we already computed those from the *positive* examples as described above.

4.6 Mapping to Triple Scores

The above approaches yield a variety of results. Binary classifications, weighted sums and probabilities. However, we actually want to compute scores for triples. While this is trivial for binary classifications (assign minimum and maximum), we distinguish two approaches for sums and probabilities. Keep in mind that we assume there is at least one primary profession for each person. For comparison with the crowdsourcing judgments, we want scores from 0 to 7 but the methods apply for any desired magnitude and, without rounding, naturally allow continuous scores as well.

Maplin Linearly scale computed values to the range 0 to 7. In practice, just divide all sums or probabilities by the highest one. Then multiply by 7 and round to the nearest integer.

Maplog Scale computed values to the range 0 to 7 such that the next highest score corresponds to twice the sum or probability. In practice, divide all sums or probabilities by the highest one. Then multiply by 2^7, take the logarithm to base 2, and finally take the integer portion of the result.

We have found that the intuitive *Maplin* works much better with the sums of weights obtained by counting triples. *Maplog*, however, is stronger when mapping probabilities to triples scores. This is true for both, our generative model, and the topic model, Labeled LDA, we compare against. The probabilistic models tend to assign high probabilities to the top professions, leaving small probabilities to all others. Differences between debatable professions and definite secondary are small in total probability difference value but still observable when comparing magnitudes.

5. EVALUATION

We first discuss the experimental setup: data + ground truth, algorithms compared and quality measures. In Section 5.2, we present and discuss our main results. In Section 5.4, we discuss many variants (algorithms and quality measures) that we also experimented with but which did not turn out to work well.

All of the data and code needed to reproduce our experiments are available under `http://ad.informatik.uni-freiburg.de/publications`.

5.1 Experimental Setup

Data We extracted all triples from the Freebase relations *Profession* and *Country of nationality*. In all experiments we only consider persons with at least two different profession or nationalities, i.e., we only consider the non-trivial cases. Files with all these triples are available under the above URL.

Ground truth For the *profession* relation, we randomly selected a total of 1225 triples pertaining to 298 different people entities, as described in Section 3.1. Each of these triples was then labeled independently by 7 human judges, using crowdsourcing, as described in Section 3.1. This gives a total of 8.575 labels. We repeated the task for all 386 triples of a random subset of 98 from the 298 people. This gives us another 2.702 labels. We used these as control labels to assess the quality of our main set above; see Section 5.2 below. We have presented the distribution of the scores in the ground truth in Section 3.4 in Figure 2. For the *nationaliy* relation, we randomly selected a total of 162 triples pertaining to 77 different people entities and ran the same experiment. This gives us another 1134 judgments.

Algorithms Compared We compare the following: three baselines, six algorithms, and the output of two control experiments.

We normalized all approaches to yield an integer score from the range 0..7 for each triple.

First For each person, take the entity's description from Freebase[3] Look for the first literal mention of one of that person's profession. That profession gets a score of 7, all other professions get a score of 0. This may look overly simplistic, but actually this is what most of the judges did in the previous version of our task as presented in Section 3.2.

Random Make 7 independent judgments for each triple, where for each judgment primary and secondary are equally likely. That is, pick the score at random from a binomial distribution with $n = 7$ and $p = 0.5$. This simulates human judges that guess randomly.

Prefixes For each triple, count the number of occurrences of a hand-picked prefix of the profession (same for all professions) in the Wikipedia article of the person. Map these counts to scores (per person) using *Maplin*, as described in Section 4.6.

Labeled LDA (LLDA) The closest approach from previous work, as described in Section 2. We use a topic label for each profession and label each person with all of her professions. The Dirichlet prior, α, is set to 2.0 and we run Gibbs sampling for 100 iterations. Other parameter settings for α gave much worse results and more iterations showed no improvement. We map the probabilities to scores using *Maplog*.

Words Classification For each triple, make a binary decision (score 0 or 7) using the logistic-regression classifier (see Section 4.3).

Words Counting For each profession, learn a weighted list of words, as described in Section 4.4. For each triple, add the weights of all profession words in all contexts containing the entity. Map these weight sums to scores using *Maplin*.

Words MLE For each triple, we derive a score using the generative model described in Section 4.5 and the *Maplog* mapping.

Counting Combined For each triple, use the *Words Counting* method. Additionally look at the decision of the *Words Classification* method. If the binary classification is positive (score 7), increase the score to the average of the two. The intuition behind this is, that for strongly related professions (e.g., Poet and Playwright), it is hard to attribute a text passage to one of the two. The binary decisions made by the *Words Classification* method do not have this problem.

MLE Combined Similar to *Counting Combined* but starting with the *Words MLE* method instead of the counting.

Control Judges Take the labels from the human control judges, as described above.

Control Expected The expected values when comparing two scorings, where in each scoring each of the seven judgments for a triple is primary with probability p, where p is chosen from the posterior distribution given score s, where s is the score given by the crowdsourcing judges.[4]

Evaluation measures We use two kinds of measures: score-based and rank-based.

Score-based The score-based measures directly compare the scores of two sequences s and s' of triple scores for the same sequence of n triples. We consider two measures: *accuracy* and *average score deviation*. The accuracy has an integer parameter δ and measures the percentage of triple scores that deviate by at most δ. That is, Acc-$\delta = |\{i : |s_i - s'_i| \leq \delta\}|$. The average score deviation is simply the average over all absolute score differences. That is, $\text{ASD} = \sum_i |s_i - s'_i|/n$. We will see that the relative performance of the various methods shows in both measures, but that the Acc figures are more intuitive and insightful.

Rank-based The rank-based measures compare two rankings of the same sequence of triples. We will use them to compare two rankings of all professions / nationalities of a single person, and the average over all persons. We consider three standard ranking measures: Kendall's Tau, the footrule distance, and nDCG.

Because scores in the gold standard are discrete, items often share a rank. Such a ranking with ties constitutes a *partial ranking* [14]. To compare partial rankings we use adapted versions of Kendall's τ from [14]: $\tau_p = \frac{1}{Z}(n_d + p \cdot n_t)$, where n_d is the number of *discordant* (inverted) pairs, n_t is the number of pairs that are tied in the gold standard but not in the predicted ranking or vice versa, p is a penalization factor for these pairs which we set to 0.5, and the normalization factor Z (the number of ordered pairs plus p times the number of tied pairs in the gold standard).

The Spearman footrule distance counts the displacements of all elements. It is $f = \frac{1}{Z}\sum |\sigma_p(i) - \sigma_g(i)|$, where i ranges over the items of the list, and $\sigma_p(i)$ and $\sigma_g(i)$ is the rank of item i in the predicted partial ranking and gold standard partial ranking, respectively. The normalization factor Z corresponds to the maximum possible distance and causes f to be in the interval $[0, 1]$.

For nDCG, we remark that it also takes the exact scores into account: ranking a triple lower than it should be is punished more, the higher the score of that triple is.

5.2 Main results

Score-based evaluation of the *profession* triples We first discuss the results for our score-based measures for the *profession* relation, shown in Table 4.

Method	Accuracy (Acc)			Average Score Diff
	$\delta = 1$	$\delta = 2$	$\delta = 4$	
First	41%	**53%**	71%	2.71
Random	31%	**55%**	91%	2.39
Words Classification	47%	**61%**	78%	2.09
Prefixes	50%	**64%**	83%	2.07
LLDA	50%	**68%**	89%	1.86
Words Counting	57%	**75%**	94%	1.61
Words MLE	57%	**77%**	95%	1.61
Count Combined	58%	**77%**	95%	1.52
MLE Combined	63%	**80%**	96%	1.57
Control Expected	75%	**91%**	99%	0.93
Control Judges	76%	**94%**	99%	0.92

Table 4: Accuracies and ASD for the *profession* relation, best methods last.

The last line (Control Judges) shows that the human judges rarely disagree by more than 2 on the 0-7 scale. The Acc-4 measure shows that there are almost no stark disagreements, that is, by more than 4.[5] A disagreement up to 1 is not unusual though. The average score deviation is 0.92. As the next to last line (Control Expected) shows, this is essentially what would be expected from random

[3]For most entities, this is just the abstract of the corresponding Wikipedia page.

[4]That is, $Pr(p = k/7) \propto \Pr(X = k)$, for $k = 0..7$, where X is from the binomial distribution $B(7, s/7)$.

[5]Note that such disagreements can only happen for scores 0-2 and 5-7.

fluctuation. Acc-2 hence seems to be the single most intuitive measure (for integer scores on a scale 0-7). We therefore emphasized the results for that measure in Table 4.

Our binary classification algorithm achieves an Acc-2 of 61%, which gradually improves to 80% for our most sophisticated algorithm. There is hardly a difference between the MLE approach and word counting with proper normalization. Our best approach makes glaring mistakes (Acc-4) for only 4% of all triples. For the Acc-2 measure, our best approach is still more than 10% away from the ideal. In Section 5.4, we discuss various options for improvement, none of which actually gives a significant further improvement though. We conclude that our problem is what could be called "NLP-hard". Under that condition, we consider an Acc-2 of 80% a very good result.

The simple "First" baseline performs similarly bad as the "Random" baseline, which simulates a random guess for each judge. Note that the "Random" baseline makes glaring mistakes (Acc-4) only for 9% of the triples. This is because the scores of this baseline follow a binomial distribution, with a probability of 0.55 that the score is 3 or 4. For these two scores, the deviation from the "true" value cannot be larger than 4. For the more extreme scores, the probability of being off by more than 4 is also low. Acc-2, however, is only about 55% for "Random". Note that a more extreme version of "Random", which picks only one of the scores 3 or 4 at random for each triple, would achieve an Acc-4 of 100%, but an Acc-2 of only 52%.

The relative performance of the various approaches is reflected by the average score difference (ASD), shown in the last column of Table 4. The various Acc measures provide a more refined picture though.

Rank-based results for the *profession* triples We next discuss the results for our rank-based measures for the *profession* relation, shown in Table 5.

Method	**Kendall**	Footrule	nDCG
Random	**0.51**	0.58	0.80
First	**0.40**	0.47	0.92
LLDA	**0.32**	0.38	0.88
Words Classification	**0.32**	0.35	0.88
Prefixes	**0.31**	0.35	0.88
Words MLE	**0.23**	0.28	0.94
Words Counting	**0.24**	0.29	0.94
Counting Combined	**0.24**	0.28	0.94
MLE Combined	**0.22**	0.27	0.94
Control Judges	**0.18**	0.21	0.97

Table 5: Average rank differences for the *profession* relation, best methods last.

We observe that the relative quality of the various baselines and algorithms is about the same in all three measures. Since Kendall is the simplest and perhaps most intuitive measure, we have highlighted the results for that measure in Table 5.

Footrule is always slightly larger than Kendall. The nDCG values are relative large already for the simple Random baseline. This is an artifact of the short lists we are comparing here (for each person, his or her professions, which are often only two or three). Indeed, for a list with only two items, there are only two possible values for nDCG: 1 and $1/\log_2 3 \approx 0.63$. The average of the two is ≈ 0.82.

The relative order of the methods is similar as for the score-based evaluation of Table 4. The only notable exception is that in the rank-based evaluation, the simple "First" baseline is significantly better than "Random", while in the score-based evaluation, it was the other way round. The reason is simple: "First" almost always gets the first item in the list right (the first profession listed in Wikipedia, is almost always the most obvious profession of the person in question). For persons with only two professions, this already gives the perfect ranking. For persons with three professions, this already gets 2 of the 3 possibly transpositions right. In the score-based evaluation, the score for the triple with this "first" profession is just one out of k, when the person has k professions. Apart from that, "First" (and "Words Classification") make binary decisions for one of the most extreme scores. This is punished by the Accuracy measures. Random, on the other hand, tends to select the middle ground scores 3 and 4 which are not punished as strongly.

Note that the compared rankings are invariant under score mapping (*Maplin* or *Maplog*, see Section 4.6). Hence, score mapping affects neither Kendall nor Footrule. It does, in principle, affect nDCG, since that measure also takes the actual scores into account. We found the effect to be insignificant though (at most 0.02 difference for affected approaches).

Score- and ranked-based evaluation of the *nationality* triples We repeated the experiments above for the triples from the *nationality* relation. We focused on the baselines and the best-performing methods.

Method	Accuracy (Acc)			Average Score Diff
	$\delta = 1$	$\delta = 2$	$\delta = 4$	
First	28%	**36%**	51%	3.89
Random	31%	**55%**	91%	2.83
Prefixes	41%	**52%**	71%	2.21
Words MLE	58%	**78%**	95%	1.71
Words Counting	63%	**78%**	96%	1.34
Control Expected	76%	**92%**	99%	0.82

Method	**Kendall**	Footrule	nDCG
Random	**0.51**	0.58	0.80
First	**0.41**	0.42	0.90
Prefixes	**0.51**	0.53	0.88
Words MLE	**0.44**	0.45	0.92
Words Counting	**0.36**	0.37	0.94

Table 6: Score-based and ranked-based results for the *nationality* relation, best methods last.

Table 6 shows the results for our experiments with triples from the *nationality* relation. Sophisticated approaches have performance similar to the run on triples of the *profession* relation (see Tables 4 and 5). This is good news, because it was not guaranteed that Wikipedia texts provided a sufficiently strong signal for our algorithms to perform well. Interestingly, baselines "First" and "Prefixes" are weaker on this task. Apparently, direct mentions of a nationality in the Wikipedia article are not as useful as they are for professions. A typical example could be the term "German-American" to describe (primary) Americans with German roots. Not only is the secondary nationality mentioned first, the article may also discuss the person's ancestry further, frequently mentioning that country. Statistical signals for the primary nationality, like

mentions of the place of residence or typical institutions are only considered by our more advanced approaches.

5.3 Refined analysis

We also conducted an analysis of the "clear cases", that is, the triples, where the score from the crowdsourcing judges were either 6 or 7 or 1 or 0 (all judges, except at most one, agree). We analyzed the percentage of triples, which got the "reverse" scores for these triples, that is 0-1 for 6-7 and 6-7 for 0-1. The figures were very similar to those already captured by the Acc-4 figure in Table 4. Recall that a score deviation of more than 4 can only happen for the extreme scores. Thus, no new insights were provided by this analysis.

We also analyzed the dependency between accuracy and popularity of a person.

Method	Popularity Bucket					
	1	2	3	4	5	6
Words Counting	**58%**	76%	71%	71%	76%	81%
Count Combined	**63%**	81%	72%	68%	76%	81%
Words MLE	**68%**	84%	76%	79%	84%	76%
MLE Combined	**74%**	84%	75%	79%	84%	79%

Method	Popularity Bucket					
	7	8	9	10	11	12
Words Counting	77%	75%	70%	72%	82%	83%
Count Combined	86%	78%	76%	76%	78%	78%
Words MLE	76%	74%	69%	77%	79%	82%
MLE Combined	82%	75%	76%	80%	80%	82%

Table 7: Accuracy-2 for the *profession* relation, breakdown by person popularity. Popularity Bucket i contains persons with a number of occurrences between 2^{i+2} and 2^{i+3}. Buckets 1 and 12 are special cases and contain persons with less than 2^4, resp. more than 2^{14}, occurrences.

The buckets of popularity used in Table 7 pertain to the buckets from our selection for the crowdsourcing task (see Section 3.1). The breakdown shows that approaches function well on all persons with more than 16 occurrences in the text. This corresponds to buckets 2 and above. While more text is always better for our statistical models, we observe that the minimum required to work reasonably well, is already very low. The persons in bucket 1 with very little associated text are harder to get correct. We can observe two things: *Words Counting* has more problems than *Words MLE*. Further, the combination with the classifier specifically helps to even out the problems with entities in this bucket.

5.4 Variants

Our main results show a gap of 11-14% between our best method, and what human judges can achieve. We have experimented with numerous variations of the methods described in Section 4 and 5.1. None of these variations turned out to give an improvement.

Stemming. All approaches from Section 4 can be used with and without stemming. We tried both the Porter [22] and the Lancaster stemmer [19]. For all methods, stemming (in either variant) did more harm than good.

Word pairs instead of just words. One significant source of errors in the methods from Section 4 is that *single* words are ambiguous indicators for certain triples. For example, the word *film* is a strong indicator for both *Actor* and *Film Director*. Some persons have both professions. To distinguish between the two, the context of the word *film* is important. One obvious approach to address this is to consider pairs of words as features, instead of just single words. For the *profession* relation, the influence on both the score-based and the rank-based measure was insignificant though. For the *nationality* relation, results became significantly worse when using word pairs. This is understandable, since for the majority of nationalities single words are perfectly suited to uniquely identify them (e.g., *Canadian*, *Italian*, *German*).

TF versus TF-IDF In our evaluation, we used tf-idf for all our features. We also experimented with tf features, as well as with variations of tf-idf that give more emphasis to the idf part or that dampened the tf factor like in BM25. Results were either unaffected or became slightly worse.

Non-linear score mappings. In Section 3, we discussed two score mappings: Maplin and Maplog. The latter uses the mapping $x \mapsto \log_2 x$. The rationale was that the MLE approach produces raw scores (the probabilities) that grow super-linearly with the actual score. We experimented with a variety of other non-linear score mappings to capture this behavior, in particular, $x \mapsto x^\alpha$, for various value of α. The results were similar but never significantly better than for our Maplog.

Sentences vs. contexts In our methods in Section 4, we consider only those words that occur "in the same context" as the entity in question. We experimented with two realizations of this context: co-occurrence in the same *sentence* and co-occurrence in the same *context*, as described in Section 4. Contexts consistently outperformed sentences by a few percent. The improvement was more pronounced for popular entities, where we have many occurrences of the indicator words. This is in accordance with the theory from [4]. Co-occurrence in the same sentences does not necessarily mean that the two entities / words that co-occur have something to do with each other. When they co-occur in the same context, they have, by the definition of context from [4].

Whole corpus vs. only the Wikipedia article Almost all human judges indicated that they only used the Wikipedia article of a person to make their decision (95% of all decisions). For our methods from Section 4, we considered co-occurrences anywhere in the text corpus. When restricting to only the Wikipedia page of the respective person, like the human judges did, results consistently became much worse. This is easy to understand, since all our methods are statistics-based, and the whole corpus simply provides more information, and hence also more certainty to make a good decision.

Treating hierarchy / specialization In Freebase, a person's professions may include generalizations of another profession that is listed. For example, for John Lennon there are (among other professions) singer-songwriter, keyboard player, guitarist and also musician. Like for Hierarchical Classification [18] problems, two obvious approaches are to (1) ignore the hierarchy and classify each triple and to (2) only classify specializations and infer a score for parent professions. We have experimented with both approaches and found that there is only little difference. If the same inference rules from (2) are used on both, our scores and the human judgments (human judgments are corrected towards a higher score if the human judges have given that higher score to a specialization of the profession), this increases the final scores by a few percent (e.g., 82% for *MLE Combined*). Otherwise, there is not much difference.

Knowledge-base facts An obvious alternative to using text for computing triple scores is to use information from the knowledge base. However, we have made the experience that 1) only few properties

are reflected by other facts in the knowledge base and 2) even in cases where they are, they are typically only available for popular instances. For example, an indication for the relevance of an actor may be in how many movies he acted, or how many awards he won for them. But there are no such facts in the knowledge base for a geologist, a surgeon, or a firefighter. Furthermore, in Freebase, out of 612K persons with more than one profession, 400K have less than 10 and 243K less than 6 other facts (besides type and profession information).

5.5 Manual Error Analysis

For all our methods from Section 4, as well as for all the variants discussed in Section 5.4, we conducted a manual error analysis. We discovered two main sources of errors.

Wrong reference An indicator word sometimes co-occurs with the person in question, but is actually relating to another person. For example, Michael Jackson is listed as a Film Director, which is certainly not one of his main professions. The Wikipedia article about him contains many mentions of the word *directed* or *director* though. But most of them relate not to him but to a person directing one of his shows.

Wrong meaning An indicator word is sometimes used with another meaning. For example. John F. Kennedy is listed as a Military Officer, which is certainly not one of his main professions. The Wikipedia article contains many mentions of variants of the word *military* though. But most of them relate to political actions during his presidency with respect to the military.

Some of the variants discussed in Section 5.4 mitigate the effect of these problems somewhat. For example, restricting co-occurrence to semantic contexts instead of full sentences helps the wrong-reference problem in several instances. Or, considering word pairs instead of single words helps with the wrong-meaning problem in several instances. However, a significant number of instances remain, where deep natural language understanding appears to be required. Consider the sentences "<Screenwriter X> was very happy with how his script was visualized" vs "<Film Director Y> was praised a lot for how he visualized the script". It is hard to understand the roles of the persons in these sentences. But it is necessary to correctly decide if the sentence is evidence for a profession *Screenwriter* or *Film Producer*.

6. CONCLUSION

We have studied the problem of computing relevance scores for knowledge-base triples from type-like relations. We have shown that we can compute good such scores in an unsupervised manner from a text corpus with an accuracy of about 80%.

We have described various attempts to further improve the accuracy of our triple scores, towards the over 90% accuracy achieved by human judges. These attempts, together with a manual error analysis, suggest that deep natural language understanding is required to close this gap. In particular, we identified two main error sources: *wrong reference* and *wrong meaning* of indicator words. We consider these a hard but worthwhile task for future work.

The focus of this work is on scores for individual triples. This problem is practically relevant on its own and also hard enough on its own. Still, it would be interesting to follow up on previous work about integrating such triple scores into a meaningful ranking for complex knowledge-base queries.

7. REFERENCES

[1] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[2] K. Balog, P. Serdyukov, and A. P. de Vries. Overview of the TREC 2011 Entity Track. In *TREC*, 2011.

[3] H. Bast, F. Bäurle, B. Buchhold, and E. Haussmann. Broccoli: Semantic full-text search at your fingertips. *CoRR*, abs/1207.2615, 2012.

[4] H. Bast and E. Haussmann. Open information extraction via contextual sentence decomposition. In *ICSC*, pages 154–159, 2013.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. In *NIPS*, pages 601–608, 2001.

[7] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[8] J. P. Cedeño and K. S. Candan. R^2DF framework for ranked path queries over weighted RDF graphs. In *WIMS*, page 40, 2011.

[9] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Soc.*, pages 1–38, 1977.

[11] R. Q. Dividino, G. Gröner, S. Scheglmann, and M. Thimm. Ranking RDF with provenance via preference aggregation. In *EKAW*, pages 154–163, 2012.

[12] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From data fusion to knowledge fusion. *PVLDB*, 7(10):881–892, 2014.

[13] S. Elbassuoni, M. Ramanath, R. Schenkel, M. Sydow, and G. Weikum. Language-model-based ranking for queries on RDF-graphs. In *CIKM*, pages 977–986, 2009.

[14] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and aggregating rankings with ties. In *PODS*, pages 47–58, 2004.

[15] J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[16] T. Franz, A. Schultz, S. Sizov, and S. Staab. TripleRank: Ranking semantic web data by tensor decomposition. In *ISWC*, pages 213–228, 2009.

[17] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, pages 50–57, 1999.

[18] C. N. S. Jr. and A. A. Freitas. A survey of hierarchical classification across different application domains. *Data Min. Knowl. Discov.*, 22(1-2):31–72, 2011.

[19] C. D. Paice. Another stemmer. *SIGIR Forum*, 24(3):56–61, 1990.

[20] S. Parsons. Current approaches to handling imperfect information in data and knowledge bases. *IEEE Trans. Knowl. Data Eng.*, 8(3):353–372, 1996.

[21] D. Ramage, D. L. W. Hall, R. Nallapati, and C. D. Manning. Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. In *EMNLP*, pages 248–256, 2009.

[22] C. J. Van Rijsbergen, S. E. Robertson, and M. F. Porter. *New models in probabilistic information retrieval*. Computer Laboratory, University of Cambridge, 1980.

WSDM Cup 2017: Vandalism Detection and Triple Scoring*

Stefan Heindorf
Paderborn University
heindorf@uni-paderborn.de

Martin Potthast
Bauhaus-Universität Weimar
martin.potthast@uni-weimar.de

Hannah Bast
University of Freiburg
bast@informatik.uni-freiburg.de

Björn Buchhold
University of Freiburg
buchholb@informatik.uni-freiburg.de

Elmar Haussmann
University of Freiburg
haussmann@informatik.uni-freiburg.de

ABSTRACT

The WSDM Cup 2017 was a data mining challenge held in conjunction with the 10th International Conference on Web Search and Data Mining (WSDM). It addressed key challenges of knowledge bases today: quality assurance and entity search. For quality assurance, we tackle the task of vandalism detection, based on a dataset of more than 82 million user-contributed revisions of the Wikidata knowledge base, all of which annotated with regard to whether or not they are vandalism. For entity search, we tackle the task of triple scoring, using a dataset that comprises relevance scores for triples from type-like relations including occupation and country of citizenship, based on about 10,000 human relevance judgments. For reproducibility sake, participants were asked to submit their software on TIRA, a cloud-based evaluation platform, and they were incentivized to share their approaches open source.

Keywords: Knowledge Base; Vandalism; Data Quality; Search

1. TASK ON VANDALISM DETECTION

Knowledge is increasingly gathered by the crowd. Perhaps the most prominent example is Wikidata, the knowledge base of the Wikimedia Foundation that can be edited by anyone, and that stores structured data similar to RDF triples. Most volunteers' contributions are of high quality, whereas some vandalize and damage the knowledge base. The latters' impact can be severe: integrating Wikidata into information systems such as search engines or question-answering systems bears the risk of spreading false information to all their users. Moreover, manually reviewing millions of contributions every month imposes a high workload on the community. Hence, the goal of this task is to develop an effective vandalism detection model for Wikidata:

> Given a Wikidata revision, the task is to compute a quality score denoting the likelihood of this revision being vandalism (or similarly damaging).

*We thank Adobe Systems Inc. for sponsoring the event, and Wikimedia Germany for supporting it.

WSDM 2017 February 06-10, 2017, Cambridge, United Kingdom

ACM ISBN 978-1-4503-4675-7/17/02.

DOI: http://dx.doi.org/10.1145/3018661.3022762

Table 1: The vandalism detection evaluation datasets in terms of time period covered, revisions, sessions, items, and users as per Heindorf et al. [7]. Numbers are given in thousands.

Dataset	From	To	Revisions	Sessions	Items	Users
Training	Oct 1, 2012	Feb 29, 2016	65,010	36,552	12,401	471
Validation	Mar 1, 2016	Apr 30, 2016	7,225	3,827	3,116	43
Test	May 1, 2016	Jun 30, 2016	10,445	3,122	2,661	41

Revisions were to be scored in near real time as soon as a revision arrives, allowing for immediate action upon potential vandalism. Moreover, a model should hint at vandalism across a wide range of precision/recall points to enable use cases such as fully automatic reversion of damaging edits at high precision, as well as pre-filtering revisions at high recall and ranking them with respect to importance of being reviewed.

For the challenge, we constructed the Wikidata Vandalism Corpus 2016 (WDVC-2016),[1] an up-to-date version of the Wikidata Vandalism Corpus 2015 (WDVC-2015) [6]: it consists of user-contributed edits, excluding edits by bots, alongside annotations whether or not an edit has been reverted via the administrative rollback feature, which is employed at Wikidata to revert vandalism and similarly damaging contributions. This way, we obtained a large-scale corpus ranging from October 2012 to June 2016, containing over 82 million revisions, 198,147 of which are labeled as vandalism. The corpus also supplies meta information that is not readily available from Wikidata, such as geolocalization data of all anonymous edits as well as Wikidata revision tags originating from both the Wikidata Abuse Filter and semi-automatic editing tools. Table 1 gives an overview of the corpus. Participants were provided training data and validation data while the test data was held back until the final evaluation. To prevent teams from using information that emerged after a revision was made, we sorted all revisions by time and employed the evaluation-as-a-service platform TIRA [4][2] in combination with a newly developed data server that only provides new revisions after a participant's software has reported scores for previous revisions. The setup, datasets, rules, and measures, are described in detail on http://www.wsdm-cup-2017.org/vandalism-detection.html.

As our main evaluation metric, we employ the area under curve of the receiver operating characteristic because it is the de facto standard for imbalanced learning tasks and enables a comparison to state-of-the-art vandalism detectors [7]. For informational purposes, we compute the area under the precision-recall curve, too.

The final evaluation results will be published in the workshop proceedings of the WSDM Cup 2017 [5].

[1] Available from http://www.wsdm-cup-2017.org/vandalism-detection.html
[2] http://www.tira.io

2. TASK ON TRIPLE SCORING

Knowledge bases allow queries that express the search intent precisely. For example, we can easily formulate a query that gives us precisely a list of all *American actors* in a knowledge base. Note the fundamental difference to full-text search, where keyword queries are only approximations of the actual search intent, and thus result lists are typically a mix of relevant and irrelevant hits.

But even for result sets containing only relevant items, a ranking of the contained items is often desirable. One reason is similar as in full-text search: when the result set is very large, we cannot look at all items and thus want the most "interesting" items first. But even for small result sets, it is useful to show the inherent order of the items in case there is one. We give two examples. The numbers refer to a sanitized dump of Freebase from June 29, 2014; see [1].

Example 1 (American actors): Consider the query that returns all entities that have *Actor* as their profession and *American* as their nationality. On the latest version of the Freebase dataset, this query has 64,757 matches. A straightforward ranking would be by popularity, as measured, e.g., by counting the number of occurrences of each entity in a reference text corpus. Doing that, the top-5 results for our query look as follows (the first result is G. W. Bush):

George Bush,*Hillary Clinton*,*Tim Burton*,*Lady Gaga*,*Johnny Depp*

All five of these are indeed listed as actors in Freebase. This is correct in the sense that each of them appeared in a number of movies, and be it only in documentary movies as themselves or in short cameo roles. However, Bush and Clinton are known as politicians, Burton is known as a film director, and Lady Gaga as a musician. Only Johnny Depp, number five in the list above, is primarily an actor. He should be ranked before the other four.

Example 2 (professions of a single person): Consider all professions by Arnold Schwarzenegger. Freebase lists 10 entries:

Actor, *Athlete*, *Bodybuilder*, *Businessperson*, *Entrepreneur*, *Film Producer*, *Investor*, *Politician*, *Television Director*, *Writer*

Again, all of them are correct in a sense. For this query, ranking by "popularity" (of the professions) makes even less sense than for the query from Example 1. Rather, we would like to have the "main" professions of that particular person at the top. For Arnold Schwarzenegger that would be: *Actor*, *Politician*, *Bodybuilder*. Note how we have an ill-defined task here: it is debatable whether Arnold Schwarzenegger is more of an actor or more of a politician. But he is certainly more of an actor than a writer.

2.1 Task Definition

The task is to compute relevance scores for triples from type-like relations. The following definition is adapted from [2]:

> Given a list of triples from two type-like relations (profession and nationality), for each triple compute an integer score from 0..7 that measures the degree to which the subject belongs to the respective type (expressed by the predicate and object).

Here are four example scores, related to the example queries above:

Tim Burton	*profession*	*Actor*	2
Tim Burton	*profession*	*Director*	7
Johnny Depp	*profession*	*Actor*	7
A. Schwarzenegger	*profession*	*Actor*	6

An alternative, more intuitive way of expressing this notion of "degree" is: how "surprised" would we be to see *Actor* in a list of professions of, say, Arnold Schwarzenegger (a few people would be, most would not). This formulation is also used in the crowdsourcing task which we designed to acquire human judgments for the ground truth used in our evaluation.

2.2 Datasets

Participants were provided a knowledge base in the form of 818,023 triples from two Freebase relations: *profession* and *nationality*. Overall, these triples contained 385,426 different subjects, 200 different professions, and 100 different nationalities.

We constructed a ground truth for 1,387 of these triples (1,028 profession, 359 nationality). For each triple we obtained 7 binary relevance judgments from a carefully implemented and controlled crowdsourcing task, as described in [2]. This gives a total of 9,709 relevance judgments. For each triple, the sum of the binary relevance judgments yields the score.

About half of this ground truth (677 triples) was made available to the participants as training data. This was useful for understanding the task and the notion of "degree" in the definition above. However, the learning task was still inherently unsupervised, because the training data covers only a subset of all professions and nationalities. Participants were allowed to use arbitrary external data for unsupervised learning. For convenience, we provided 33,159,353 sentences from Wikipedia with annotations of the 385,426 subjects. For each subject from the ground truth, there were at least three sentences (and usually many more) with that subject annotated.

The setup, datasets, rules, and measures, are described in detail on http://www.wsdm-cup-2017.org/triple-scoring.html.

2.3 Performance Measures

Three quality measures were applied to measure the quality of participating systems with respect to our ground truth:

Accuracy: the percentage of triples for which the score (an integer from the range 0..7) differs by at most 2 (in either direction) from the score in the ground truth.

Average score difference: the average (over all triples in the ground truth) of the absolute difference of the score computed by the participating system and the score from the ground truth.

Kendall's Tau: a ranked-based measure which compares the ranking of all the professions (or nationalities) of a person with the ranking computed from the ground truth scores. The handling of items with equal score is described in [2, Section 5.1] and under the link above.

Note that the Accuracy measure can only increase (and never decrease) when all scores 0 and 1 are rounded up to 2, and all scores 6 and 7 are rounded down to 5. For reasons of fairness, we therefore applied this simple transformation to all submissions when comparing with respect to Accuracy.

The final evaluation results will be published in the workshop proceedings of the WSDM Cup 2017 [3].

References

[1] H. Bast, F. Bäurle, B. Buchhold, and E. Haußmann. Easy access to the freebase dataset. In *WWW*, pages 95–98, 2014.

[2] H. Bast, B. Buchhold, and E. Haussmann. Relevance scores for triples from type-like relations. In *SIGIR*, pages 243–252, 2015.

[3] H. Bast, B. Buchhold, and E. Haussmann. Overview of the Triple Scoring Task at WSDM Cup 2017. *To appear*, 2017.

[4] T. Gollub, B. Stein, and S. Burrows. Ousting Ivory Tower Research: Towards a Web Framework for Providing Experiments as a Service. In SIGIR, pages 1125–1126, 2012.

[5] S. Heindorf, M. Potthast, G. Engels, and B. Stein. Overview of the Wikidata Vandalism Detection Task at WSDM Cup 2017. *To appear*, 2017.

[6] S. Heindorf, M. Potthast, B. Stein, and G. Engels. Towards Vandalism Detection in Knowledge Bases: Corpus Construction and Analysis. In *SIGIR*, pages 831–834, 2015.

[7] S. Heindorf, M. Potthast, B. Stein, and G. Engels. Vandalism Detection in Wikidata. In *CIKM*, pages 327–336, 2016.

More Accurate Question Answering on Freebase

Hannah Bast, Elmar Haussmann
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast, haussmann}@informatik.uni-freiburg.de

ABSTRACT

Real-world factoid or list questions often have a simple structure, yet are hard to match to facts in a given knowledge base due to high representational and linguistic variability. For example, to answer "who is the ceo of apple" on Freebase requires a match to an abstract "leadership" entity with three relations "role", "organization" and "person", and two other entities "apple inc" and "managing director". Recent years have seen a surge of research activity on learning-based solutions for this method. We further advance the state of the art by adopting learning-to-rank methodology and by fully addressing the inherent entity recognition problem, which was neglected in recent works.

We evaluate our system, called *Aqqu*, on two standard benchmarks, Free917 and WebQuestions, improving the previous best result for each benchmark considerably. These two benchmarks exhibit quite different challenges, and many of the existing approaches were evaluated (and work well) only for one of them. We also consider efficiency aspects and take care that all questions can be answered interactively (that is, within a second). Materials for full reproducibility are available on our website: `http://ad.informatik.uni-freiburg.de/publications` .

1. INTRODUCTION

Knowledge bases like Freebase have reached an impressive coverage of general knowledge. The data is stored in a clean and structured manner, and can be queried unambiguously via structured languages like SPARQL. However, given the enormous amount of information (2.9 billion triples for Freebase), mapping a search desire to the right query can be an extremely hard task even for an expert user. For example, consider the (seemingly) simple question *who is the ceo of apple*. The answer is indeed contained in Freebase, and the corresponding SPARQL query[1] is:

```
select ?name where {
  Managing_Director  job_title.people_with_this_title  ?0 .
  ?0  employment_tenure.company  Apple_Inc .
  ?0  employment_tenure.person  ?name
}
```

It would clearly be preferable, if we could just ask the question in natural language, and the machine automatically computes the corresponding SPARQL query. This is the problem we consider in this paper.

We focus on "structurally simple" questions, like the one above. They involve k entities (typically two or three, in the example above: *ceo* and *apple* and the result entity), which are linked via a single k-ary relation in the knowledge base. For languages like SPARQL, k-ary relations for $k > 2$ can be represented by a special entity (one for each k-tuple in the relation) and $k - 1$ binary relations (in the example above: the three binary relations in the where clause, all connected to the ?0 entity).

The challenge for these questions is to find the matching entities and relations in the given knowledge base. The entity-matching problem is hard, because the question may use a variant of the name used in the knowledge base (synonymy), and the knowledge base may contain many entities with the same name (polysemy). For example, there are 218 entities with the name *apple* in Freebase, but the right match for the question is actually *Apple Inc.* The relation-matching problem has the same problem, which is even more difficult for k-ary relation with $k > 2$. As a further complication, questions like the above do not contain any word that matches the relations from the sought for query.[2] Note how these problems exacerbate for very large knowledge bases. If we restrict to lexical matches, we will often miss the correct query. If we allow weaker matches, the number of possibilities becomes very large. This will become clearer in Section 3.

1.1 Contributions

We consider the following as our main contributions:

- A new end-to-end system that automatically translates a given natural-language question to the matching SPARQL query on a given knowledge base. Several previous systems factor out part of the problem, for example, by assuming the right entities for the query to be given by an oracle. See Section 3 for an overview of our system.
- An evaluation of our system on two standard benchmarks, Free917 and WebQuestions, where it outperforms all pre-

[1]For the sake of readability, prefixes are omitted from the entity and relation names.

[2]This is typical when the verb *to be* is used in the question.

CIKM'15, October 19–23, 2015, Melbourne, Australia.
 ISBN 978-1-4503-3794-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2806416.2806472.

vious approaches significantly. These two benchmarks exhibit quite different challenges, and many of the existing approaches were evaluated (and work well) only for one of them. See Section 2 for an overview of the existing approaches, and Section 5 for the details of our evaluation.

• Integration of entity recognition in a learning-based approach. Previous learning-based approaches treated this sub-problem in a simplistic manner, or even factored it out by assuming the right entities to be given as part of the problem.

• Using learning-to-rank techniques to learn pair-wise comparison of query candidates. Previous approaches often use parser-inspired log-linear models for ranking.

• We also consider efficiency aspects and take care that all questions can be answered interactively, that is, within one second. Many of the previous systems do not consider this aspect, and take at least several seconds and longer to answer a single query. Again, see Section 5 for some details.

• We make the code of our system publicly available under `http://ad.informatik.uni-freiburg.de/publications` . In particular, this allows reproducing our results. The website also provides various additional useful materials; in particular, a list of mistakes and inconsistencies in the Free917 and WebQuestions benchmarks.

Throughout this paper, we focus on Freebase as the currently largest general-purpose knowledge base. However, there is nothing in our approach specific to Freebase. It works for any knowledge base with entities and (possibly k-ary) relations between them.

2. RELATED WORK

Much recent work on natural-language queries on knowledge bases has focused on two recent benchmarks, both based on Freebase: *Free917* and *WebQuestions*. Section 2.1 gives an overview over this body of work, introducing the two benchmarks on the way. In Section 5, we compare our new method against *all* methods from this section. Section 2.2 briefly discusses work using other benchmarks.

2.1 Work on Free917 and WebQuestions

We consider the works in chronological order, briefly highlighting the relative innovations to previous works and the corresponding gain in result quality. A more technical description of each of the methods is provided in Section 5.3.

In [7], the Free917 benchmark was first introduced. The benchmark consists of 917 questions along with the correct[3] knowledge-base query. All queries have exactly one (possibly k-ary) relation. The basic approach of [7] is to extend an existing semantic parser with correspondences between natural-language phrases and relation names in the knowledge base. The correspondences are learned using weak supervision techniques and from the training portion of the benchmark (70% = 641 questions).

In [15], query candidates are derived by transforming an underspecified logical form of a CCG [21] parse. This form is grounded to Freebase using a set of collapsing and expansion operators that preserve the type of the expression. This has the advantage that it leverages grammatical structure in the question and can adjust knowledge base mismatches, and the disadvantage that it relies on well-formed questions. A linear model is learned to score derivations, which are built using a dynamic programming based parser.

In [2], the WebQuestions (WQ) benchmark was introduced. This benchmark is much larger (5,810 question) but only provides the result set for each question, not the knowledge-base query. This allows gathering more training data more easily (the results were obtained via crowdsourcing). The WQ questions are also more realistic (they were obtained via the Google Suggest API) and language-wise more diverse than the Free917 questions, and hence also harder (e.g. *who runs china in 2011* asking for the former Chinese Premier). The basic approach of [2] is to generate query candidates by recursively generating logical forms. The generation is guided by a mapping of phrases to knowledge base predicates and a small set of composition rules. Candidate scores are learned with a log-linear model.

In a follow-up work [3], the process from [2] is "turned on its head" by again generating a natural-language question from each query candidate. Scores are then learned (again with a log-linear model) based on the similarity between the question representing the query candidate and the original question. This allows leverage of text-similarity information (paraphrases) from large text corpora (unrelated to the queried knowledge base).

In [25], the authors go another step further by not even generating query candidates. Instead their approach tries to identify the central entity of the question, and then iterates over each entity connected (via a single relation) to that central entity in the knowledge base. It is then decided (via a learned model) separately for each such entity whether it becomes part of the result set. In principle, this allows correct answers even when no single relation from the knowledge base matches the question (e.g., asking for a brother of someone, when the knowledge base only knows about siblings). On the downside, this adds a lot of additional features to the learning process (the attributes of the result entities). Quality-wise, the approach does not improve over [2] and [3].

In [19], the authors go yet a step further by not even using the training data. Instead, weak-supervision is used to generate learning examples from natural language sentences. The parsing step itself is conceptualized as a graph-matching problem between the graph of a CCG parse and graphs grounded in Freebase entities and relations. However, their approach was evaluated only on small (and topically narrow) subsets of the two benchmarks.

In [4], the authors try to solve the problem without any natural-language processing (not even POS-tagging). They match the results from [3] but do not improve them.

2.2 Other benchmarks

Another recent notable effort in open-domain question answering is the QALD (Question Answering over Linked Data) series of evaluation campaigns, which started in 2011. See [22] for the latest report. So far, five benchmarks have been issued, one per year. The challenges behind these benchmarks are somewhat different than those behind the Free917 and WebQuestions benchmarks from Section 2.1:

• The biggest and most diverse knowledge base used is DBpedia, which is more than an order of magnitude smaller than Freebase (about 4M vs. about 40M entities).

[3]Actually, a small portion of the queries are incorrect, but this is not a deliberate feature of the benchmark.

• A significant fraction of the questions involves more than one relation or non-trivial comparatives. For example, *what are the capitals of all countries that the himalayas run through* or *which actor was cast in the most movies*.

• The training sets are relatively small (50-100 queries for QALD 1-3). This is mainly due to the fact, discussed in Section 2.1 above, that the ground truth provides not just the correct result sets but also the corresponding SPARQL queries, which requires expensive human expert work. The benchmarks thus give relatively little opportunity for supervised learning. Indeed, most of the participating systems are unsupervised. It is one of the insights from our evaluation in Section 5 that supervised learning is key for results of the quality we achieve.

• QALD 3 and 4 contain multi-lingual versions of the datasets and questions. For QALD 5, the dataset is a combination of RDF data and free text.

For these reasons, and because there is such a substantial body of very recent work on Free917 and WebQuestions with a series of better and better results, we did not include QALD in our evaluation. We consider it a very worthwhile endeavor for future work though, to extend our approach to the QALD benchmarks.

3. SYSTEM OVERVIEW

We first describe our overall process of answering a natural language question from a knowledge base (KB). In the next sections we describe each of the steps in detail. Assume we are trying to answer the following question (from the WebQuestions benchmark):

what character does ellen play in finding nemo?

Entity identification. We begin by identifying entities from the KB that are mentioned in the question. In our example, *ellen* refers to the tv host *Ellen DeGeneres* and *finding nemo* refers to the movie *Finding Nemo*. However, like for the example in the introduction, this is not obvious: *ellen* could also refer to the actor *Ellen Page* and *finding nemo* to the video game with the same name (besides others). Instead of fixing a decision on which entities are mentioned, we delay this decision and jointly disambiguate the mentioned entities via the next steps. Hence, the result of this step is a set of (possibly overlapping) entity mentions with attached confidence scores.

Template matching. Next, we match a set of query templates to the question. Figure 1 shows our templates. Each template consists of entity and relation placeholders. A matched template corresponds to a *query candidate* which can be executed against the KB to obtain an answer.

Our simplest template consists of a single entity and an answer relation (template 1 in Figure 1). One of the query candidates for our example is generated by matching the entity for the tv host *Ellen DeGeneres* and the relation *parents* [4]:

<*Ellen DeGeneres*> <*parents*> <*T*>

This has the (wrong) interpretation of asking for her parents. A slightly more complex template contains two relations connected to the entity via a *mediator object* (template 2 in Figure 1). In our example, this matches a query candidate connecting *Ellen Page* to abstract *film performance* objects, via a *film performance* relation, and from there to all the films she acted in via a *film* relation:

<*Ellen Page*> <*performance*> <*M*>
<*M*> <*film*> <*T*>

This asks for all films *Ellen Page* acted in. Yet another template combines two entities via relations and a mediator entity (m in template 3 in Figure 1). In our example, *Ellen DeGeneres* and *Finding Nemo* are connected via two relations and a film-performance mediator.

<*Ellen DeGeneres*> <*performance*> <*M*>
<*M*> <*film*> <*Finding Nemo*>
<*M*> <*character*> <*T*>

We find this connection using an efficient inverted index (see Section 4.2) and continue matching from the mediator. In particular, we create query candidates asking for the *character* (Dory) and *performance type* (Voice) of *Ellen DeGeneres* in *Finding Nemo*. The final result of this step is a set of all the matched query candidates.

Relation matching. The query candidates still miss the fundamental information about which relations were actually mentioned and asked for in the question. We distinguish three ways of matching relations of the query candidate to words in the question: 1) via the name or description of the relation in the KB, 2) via words learned for each relation using distant supervision, 3) via supervised learning on a training set. Each match has a confidence score attached.

In our example, a word learned for the relations *performance* and *film* connecting an actor to the film she acted in is *play*. This matches in the query candidates asking for all films of *Ellen Page* and for the performance type or character of *Ellen DeGeneres* in *Finding Nemo*. Furthermore, the word *character* matches the relation with the same name, whereas the relation *performance type* doesn't match. Continuing this way, all relations in all query candidates are enriched with information about what words were matched in which way.

Ranking. We now have a set of query candidates, where each candidate is enriched with information about which of its entities and relations match which parts of the question how well. It remains to rank the candidates in order to find the best matching candidate. Note that performing ranking at this final step has the strong benefit of jointly disambiguating entities and relations. A candidate can have a weak match for an entity, but a strong match for a relation, and vice versa. By deciding this at the final stage we can identify these combinations as correct, even when one of the matches seems unlikely when considered separately.

Intuitively, for our example, the candidate covering most words of the question is best. Matching *ellen* to *Ellen Page* does no longer allow matching *Finding Nemo* because these aren't actually related in the KB. On the other hand, asking for the performance type of *Ellen DeGeneres* in *Finding Nemo* doesn't match the word *character*. This leaves us with the correct interpretation of asking for her character in the movie.

4. SYSTEM DETAILS

In this section, we describe the details of our system, called *Aqqu*. Aqqu works by generating query candidates for each

[4]We use SPARQL-like triple (subject, predicate, object) notation, where uppercase characters indicate variables.

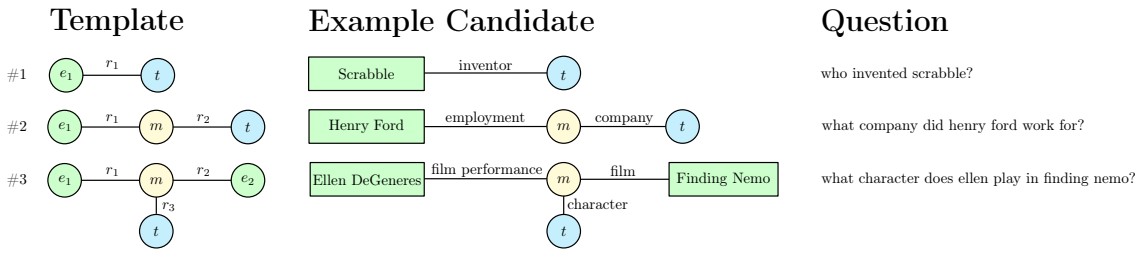

Figure 1: Query templates and example candidates with corresponding questions. A query template can consist of entity placeholders e, relation placeholders r, an intermediate object m and the answer node t.

question. These query candidates are then ranked using a learned model. The top-ranked query is then returned (or "no answer" in case the set of candidates was empty). The following subsections describe the candidate generation and ranking in detail. The previous section explained the process by an example.

4.1 Entity matching

The goal of the entity matching phase is to identify all entities from the knowledge base that match a part of the question. The match can be literal, or via an alias of the entity name.

POS-tagging We POS-tag the question using the Stanford tagger [17]. For entity matching (this subsection), we make use of the tags *NN* (noun) and *NNP* (proper noun). For relation matching (Section 4.3), we also make use of the tags *VB* (verb) and *JJ* (adjective).

Subsequence generation We generate the set S of all subsequences of words from the question, with the following two restrictions. First, a subsequence consisting of a single word must be tagged *NN*. Second, a subsequence must not "split" a sequence of words tagged *NNP*; that is, when it starts (ends) with a word tagged *NNP*, it must not be preceded (succeeded) by a word tagged *NNP*.

Find matching entities For each $s \in S$, we compute the list of all entities from the knowledge base that have s as their name or alias. We use a map from phrases (the aliases) to lists of entities (the entities with the respective alias) obtained from the CrossWikis dataset [20]. CrossWikis was built by mining the anchor text of links to Wikipedia entities (articles) from various large web-crawls. CrossWikis covers around 4 million entities from Wikipedia. Almost all of these entities also exist in Freebase, together with a link to the respective Wikipedia entity. For the remaining Freebase entities, we only consider the literal name match. Overall, we are able to recognize around 44 million entities with about 60 million aliases.

We have also experimented with the aliases provided by Freebase, but they tend to be much more noisy (wrong aliases) and less complete (important aliases missing).

Scores for the entity matches We compute a score for each match s, e computed in the previous step, where s is a subsequence of words from the question and e is an entity from Freebase with alias s. Consider a fixed alias s. CrossWikis also provides us with a probability distribution $p_{cross}(e|s)$ over the Wikipedia entities e with alias s. Let e' be a Freebase entity that is not contained in CrossWikis. Let e_{max} be the CrossWikis entity with the highest $p_{cross}(e|s)$. That is, e_{max} is the most likely Wikipedia entity for alias s. Let $p_{free}(e'|s) = p(e_{max}|s) \cdot pop(e')/pop(e_{max})$, where pop is the (alias-independent) popularity score of an entity, as described in the next subsection. Intuitively, $p_{free}(e'|s)$ estimates the probability that e' has alias s via its relative popularity to the most likely Wikipedia entity for s. We merge $p_{cross}(e|s)$ and $p_{free}(e'|s)$ into one probability distribution by simply normalizing the probabilities to sum 1.

Popularity scores for each entity For each entity, we also compute a (match-independent) popularity score. We simply take the number of times the entity is mentioned in the ClueWeb12 dataset [9], according to the annotations provided by Google [13]. The popularity scores are used for the entity match scores above. They also yield two features used in ranking each candidate; see Section 4.5.

4.2 Candidate generation

Based on the entity matches, we compute a set of query candidates as follows. We generate the query candidates in three (disjoint) subsets, one for each of the three templates shown in Figure 1. Each template stands for a query with a particular kind of structure. These three templates cover almost all of the questions in the Free917 and WebQuestions benchmarks.

Let E be the set of all entities matched to a subsequence of the question, as described in the previous section.

Template 1 For each $e \in E$, find all relations r such that there is some triple $(e, r, \cdot)$ in the knowledge base. We obtain these via a single SPARQL query for each e.

Template 2 For each $e \in E$, find all r_1, r_2, m such that there are two triples (e, r_1, m) and $(m, r_2, \cdot)$ in the knowledge base, where r_1 and r_2 are relations and m is a mediator entity. We obtain these as follows. For each e, we use a single SPARQL query to obtain all matching r_1. For each e, r_1, we then use another SPARQL query to obtain all matching r_2. Note that m remains a variable in the query candidate.

Template 3 For all pairs of entities $e_1, e_2 \in E$ such that the two subsequences matched in the question do not overlap, find all r_1, r_2, r_3 such that there are three triples (e_1, r_1, m), (m, r_2, e_2), and $(m, r_3, \cdot)$ in the knowledge base, where r_1, r_2, r_3 are relations and m is a mediator entity. We obtain these as follows. For each entity e, we precompute the list of all (r, m) such that m is a mediator entity and the triple (e, r, m) exists in the knowledge base. The list is sorted by the ids of the mediator entities. For given e_1, e_2 like above, we then intersect the lists for e_1 and e_2. For each

mediator m in the intersection, we then obtain all r_3 via a simple SPARQL query. In the query candidate, m remains a variable.

4.3 Relation matching

Let C be the set of query candidates computed in the previous subsection. For each query candidate $c \in C$, let RW_c be the set of lemmatized[5] words from the relations from c (there can be one, two, or three relations, depending on the template from which c was generated). We compute how well the words from RW_c match the subset QW of lemmatized words from the question that are not already matched by the entities from c.

We consider four kinds of matches, described in the following: literal, derivation, synonym, context. For each of these four kinds of matches, we compute a non-negative score (which is zero, if there is no match at all). It can happen that all four of these scores are zero. In the basis version of our system, we keep such candidates, in a variant we prune them; see Section 4.7.

Literal matches This score is simply the number of pairs w, q, where $w \in RW_c$ and $q \in QW$ and $w = q$. Almost all questions have no repeated words; in that case, this score is just the number of relation words that occur in the question (and are not already matched by an entity).

Derivation matches This score is the number of pairs w, q, where $w \in RW_c$ and $q \in QW$ and w is derivationally related to q. Here we also consider the POS-tag of w in the question. We precompute a map from POS-tagged words to derivations using WordNet [11]. We extract derivation links for verbs and nouns (e.g. *produce.VB - producer.NN* and vice versa). We also extract attribute links between adjectives and their describing attribute (e.g., *high.JJ - height.NN*). We extend these links with synonyms of the noun in WordNet (e.g. *high.JJ - elevation.NN*).

Synonym matches For each $w \in RW_c$ and $q \in QW$, add s to this score if w is a synonym of q with similarity s. We compute the similarity between two words by computing the cosine similarity between the associated *word vectors*. We use 300-dimensional word vectors that were computed with Google's *word2vec* on a news text corpus of size around 100 billion words.[6] We consider only synonyms, where the score is ≥ 0.4. This threshold is based on observation, but chosen very liberally: many word pairs with score above that threshold are not what humans would call "real synonyms", but almost all such "real synonyms" have a score above that threshold.

Context matches For this score, we precompute weighted *indicator* words for each relation from our knowledge base. These are words which are not necessarily synonyms of words in the relation name, but are used in text to express that relation; see below for an example. The score is then the sum of the weights of all words in QW that are indicators for one of the relations from the query candidate. For templates 2 and 3, we consider $r1.r2$ as one relation.

We learn indicator words using distant supervision [18] as follows. First, we identify entity mentions in Wikipedia using Wiki markup and a set of simple heuristics for coreference resolution, as described in [1]. We also identify dates and values using SUTime [8]. For the 23 million sentences that contain at least two entities (including dates or values), we compute a dependency parse using [17].

For each pair e_1, e_2 of entities occurring in a sentence, we look up all relations r in the knowledge base that connect them. We also treat relations $r1.r2$ that connect the entities via a mediator as a single binary relation r. If the shortest path between e_1 and e_2 in the dependency parse has length at most four, we consider all words along that path as indicator words for r. We also experimented with considering all words in the sentence, or words along longer paths, but these gave considerably worse results.

We find about 4.7 million sentences that match at least one relation this way. For example, we can thus learn that *born* is an indicator word for the relation *place of birth* from the following sentence (assuming that our knowledge base contains the respective fact):

Andy Warhol was born on August 6, 1928 in Pittsburgh.

Note that from the same sentence, we can also learn that *born* is an indicator word for the relation *date of birth*. To distinguish between the two, we need some kind of answer type matching; this is described in Section 4.4.

We compute the weights for the indicator words in the following IR-style fashion. Consider each relation as a document consisting of the words extracted for that relation. Then compute tf.idf scores for all the words in these (relation) documents in the usual way. For each relation, then only consider the top-1000 words and sum up their tf.idf scores. The weight for each word in a (relation) document is then its tf.idf score divided by this sum. This could also be interpreted as a probability distribution $p(w|r)$ over words w given a relation r.

4.4 Answer type matching

For each candidate, we perform a simple but effective binary check based on the relation leading to the answer ($r1$, $r2$, $r3$ for templates 1,2 and 3, respectively). We precompute a list of target types for each relation r by counting the types of objects o in all triples $(\cdot, r, o)$, keeping only the top ten percent of most frequent types. For questions starting with *who*, we check whether the computed target types contain the type person, character, or organization. For questions starting with *where*, we check whether the relation leads to a location or an event. For questions starting with *when* or *since when*, we check whether the type is a date; for all other questions, the check for target objects of type date is negative.

As our evaluation and error analysis shows, these simple heuristics work reasonably well for the Free917 and WebQuestions benchmarks. The reason is that our entity and relation matching already provide ample information for discriminating between candidates. However, as explained in Section 4.3, a question word like *born* alone does not permit discrimination between the two relations *place of birth* and *date of birth*. However, it is exactly those cases that can be easily discriminated with the simple answer-type check from above.

We leave elaborate answer-type detection (which has been addressed by many QA systems) to future work.

4.5 Candidate features

The previous subsections have shown two things. First, how we generate query candidates for a given question. Sec-

[5]For example, *founded* → *found* and *was* → *be*.
[6]`https://code.google.com/p/word2vec/`

ID	Description
1	number of entities in the query candidate
2	number of entities that matched exactly with their name, or with a high probability (> 0.8)
3	number of tokens of all entities that matched literally as per the previous feature
4-5	average (4) and sum (5) of entity match probabilities
6-7	average (6) and sum (7) of entity match popularities
8	number of relations in matched template
9	number of relations that were matched literally via their name
10-13	number of tokens that matched a relation of kind: literal (10), derivation (11), synonym (12), context (13)
14	sum of synonym match scores
15	sum of relation context match scores
16	number of times the answer relation (r_1, r_2, r_3 for templates 1, 2 and 3 respectively) occurs in the KB
17	a value between 0 and 1 indicating how well the relation matches according to n-gram features (Section 4.5)
18	sum of features 3 and 10; that is, the number of tokens matching a relation or entity literally
19	number of tokens that match an entity or relation divided by the total number of tokens in question
20-22	whether the result size is 0 (feature 20), 1-20 (feature 21), or larger than 20 (feature 22); all binary
23	binary result of the answer-type check (Section 4.4)

Table 1: Features used by our ranking approaches. Top/middle/bottom: features for entity matches/features for relation matches/combined or other features.

ond, how we compute various scores for each candidate that measure how well the entities and relations from the candidate match which parts of the question.

In this subsection, we show how we generate a feature vector from each candidate. Most of these features are based on the scores just mentioned. Another important feature, described below, serves to learn the correspondence between n-grams from the question and relations from query candidates. Table 1 provides an overview over all our features. In the description below, we refer to the features by their ID (first column in the table). In Section 4.6, we show how we rank candidates based on these feature vectors.

Entity/Relation matching features Features 1-7 are based on the results from the entity matching described in Section 4.1. Features 8-16 are based on the results from the relation matching step described in Section 4.3. Features 18 and 19 quantify the number of words in the question covered by entity or relation matches (feature 18 = literally, feature 19 = in any way). Features 20-22 quantify the result size. This is important, because some candidates produce huge result sizes or empty results sets, which are both rare. Feature 23 is the binary output of the simple answer-type check from Section 4.4.

N-Gram relation matching feature This feature considers correspondences between words (unigrams) or two-word phrases (bigrams) in the question and the relation in the query candidate. For example, in the WebQuestion benchmark, the question *who is ...* almost always asks for the *profession* of a person. Such a correspondence cannot be learned by any of the mechanisms described in Section 4.3. We learn this feature as follows.

For each query candidate, we generate all unigrams and bigrams of the lemmatized words of the question. The matched entities (Section 4.1) are replaced with a special word *entity*. For each n-gram, we then create an indicator feature by appending the n-gram to the relation names of the candidate. For example, for template 2 from Figure 1, one of the features would be *employment.company+work* for the uni-gram *work* and the relations *employment.company*. We then train an $L2$-regularized logistic regression classifier with all correct candidates as positive examples and all others as negative examples. The value of feature 17 is simply the (probability) output by this classifier.

This feature will be part of a subsequent step to learn a ranking that uses the same training data. To provide realistic feature values (that aren't overfit) we proceed as follows. Split the training data into six folds. In turn, leave out one fold and train the n-gram feature classifier on the remaining folds. Then, for each example in the left-out fold compute the n-gram feature value. Use this computed value as part of the training data for subsequent learning.

4.6 Ranking

For each question, we finally rank the query candidates using the feature vectors described in the previous subsection. The top-ranked query candidate is then used to provide the answer. We say "no answer" only when the set of candidates is emtpy; this is discussed in Section 4.7 below.[7]

We have experimented with state-of-the-art techniques for the learning-to-rank approach from IR [14] [16], including: RankSVM [14], RankBoost [12], LambdaRank [6] and AdaRank [23]. These only lead to moderate results and were outperformed by our approaches described below. We presume that this is because our ranking problem is degenerate. In particular, each query is only associated with a single *relevant* answer. This is different from a typical IR scenario where a query usually has several answers, sometimes with varying degrees of relevance.

We investigate two variants to obtain a ranking: *pointwise ranking* and *pairwise ranking*. These approaches are inspired by the learning-to-rank approaches from IR.

[7]Both benchmarks contain a considerable number of questions starting with *how many ...*, asking for a count. We simply replace *how many* by *what* in these questions, and count the size of the result set (unless the answer already is a count).

Pointwise ranking In the pointwise ranking approach we compute a score for each candidate. Candidates are sorted by this score to infer a ranking. The score is computed by a classifier learned on the candidate features (see Section 4.5) and training data. We create training data by using the correct candidate of each question as positive examples and all other candidates as negative examples.

A drawback of the pointwise approach is that the model "compares" question-independent examples. That is, correct (incorrect) query candidates of questions of different type and difficulty are in the same correct (incorrect) class, when in practice it is not necessary to compare or discriminate between them.

Pairwise ranking In the pairwise ranking approach, we transform the ranking problem into a binary classification problem. The idea is to learn a classifier that can predict for a given pair of candidates, whether one should be ranked before the other.

To infer a ranking, we sort the list of candidates using the learned preference relation. This works very well in practice, although our learning does not guarantee that the learned relation is transitive or anti-symmetric. We have experimented with two alternatives to sorting. Simply computing the maximum turned out to perform badly. This makes sense, because the maximum has to "survive" a larger number of comparisons. Following [10], we have also sorted the candidates by their number of "won" comparisons against all other candidates. The results were identical to those for sorting, but this method requires $\Theta(n^2)$ comparisons for n candidates.

To train the classifiers we create training examples in the following way. For a question with n query candidates, randomly select $n/2$, but at least 200 candidates (or n if $n/2 < 200$) . This is to guarantee that we have enough training examples for questions with few candidates and to avoid putting too much emphasis on questions that have more than 200 candidates.[8] Then, for each randomly selected candidate r_i and the correct candidate c, where $r_i \neq c$, create a positive example pair (c, r_i) and a negative example pair (r_i, c). The feature representation for a pair (a, b) is a tuple of the individual feature vectors and their difference: $\phi_{pair}(a, b) = (\phi(a) - \phi(b), \phi(a), \phi(b))$, where ϕ is a function extracting the features in Table 1.

Both ranking approaches, pointwise and pairwise, require a classifier. Here, we consider two different options.

Linear A logistic regression classifier. In initial experiments, other linear models, such as linear SVMs, have shown similar performance. Logistic regression is also known to output well calibrated probabilities and performs well in high-dimensional feature spaces. We train the model using L-BFGS-B [26]. To avoid over-fitting we apply $L2$-regularization choosing the regularization strength using 6-fold cross-validation on the training set.

Random forest We learn a forest of decision trees [5]. Random forests are able to learn non-linear decision boundaries, require few hyperparameters, are simple to train, and are known to perform very well on a variety of tasks.

[8]Our system generates around 200 candidates on average for a random question, but the exact value had little effect on performance in our evaluation.

4.7 Candidate pruning

Some questions may have no answers in the knowledge base. Our system, as described so far, returns "no answer" only when the set of query candidates is empty. However, as also described, this would rarely happen, since there are matching entities for every question, and we do not require that the relations match any of the words in the question.[9]

We consider two variants of our system to deal with this problem: (1) omitting the n-gram feature, and using hard pruning; and (2) keeping the n-gram feature, and using a pruning-classifier. Note that a nice side-effect of pruning is that it speeds up the ranking process because it needs to consider less candidates.

Without n-grams, with hard pruning When omitting the n-gram feature, there is no reason to keep candidates with the wrong answer type or where features 9-15 are all zero. The natural approach is then to prune such candidates before we do the ranking; this is what we call *hard pruning*. Hard pruning naturally leads to empty candidate sets for some queries. Indeed, on the Free917 benchmark, 10 questions have no answer, and our hard pruning yields an empty candidate set for 7 of them.

With n-grams, with a pruning classifier When keeping the n-gram feature, hard pruning as just described would be counterproductive. As explained in Section 4.5, the answers for the *who is ...* questions from the WebQuestions benchmark are professions. They would be eliminated when hard pruning by answer type. Also, the *profession* relation matches no words from these questions. They would hence also be eliminated when hard pruning if features 9-15 are all zero.

The goal of the pruning classifier is to weed out only the "obviously" bad candidates. For example, candidates that do not match the answer type, have bad relation matches, *and* a weak n-gram feature. We train the pruning classifier in the same way as the pointwise classifiers (see above) with the features from Table 1 using logistic regression. To optimize the classifier for recall we adjust example weights so that positive candidates have twice the weight of negative candidates. Before the ranking step, we apply the classifier to each candidate and only keep candidates classified positively.

5. EVALUATION

We perform an extensive evaluation of our system. In Section 5.1, we provide more details on our two benchmarks. In Section 5.2, we describe the evaluation measures used. In Section 5.3, we describe the systems we evaluate and compare to. In Section 5.4, we provide our main results followed by a detailed analysis in Section 5.5.

5.1 Data

We use all of Freebase as our knowledge base (2.9 billion facts on 44 million entities). Note that our approach is not tailored to Freebase and could easily be adapted to another knowledge base, e.g., WikiData[10].

Datasets We evaluate our system on two established benchmarks: *Free917* and *WebQuestions*. Each benchmark consists of a set of questions and their answers from Freebase.

[9]In that case, features 9-15 are all zero; however, the n-gram features could still be positive.

[10]`http://www.wikidata.org`

The benchmarks differ substantially in the types of questions and their complexity.

Free917 contains 917 manually generated natural language questions [7]. The questions cover a wide range of domains (81 in total). Two examples are *what fuel does an internal combustion engine use* and *how many floors does the white house have*. The most common domains, *film* and *business*, only make up 6% of the questions [7]. All questions are grammatical and tend to be tailored to Freebase. The dataset provides a translation of each question into a SPARQL-equivalent form. We execute the SPARQL queries to obtain a gold answer for each question. [7] also provide an entity lexicon: a mapping from exact text to the mentioned entity for all entities appearing in the questions. This lexicon consists of 1014 different entities. It was used for identifying entities by all systems reporting results on the dataset so far. We only make use of this lexicon where explicitly stated. To report results, we use the original split of the questions by [7] into 70% (641) questions to train and 30% questions (276) to test.

WebQuestions consists of 5,810 questions that were selected by crawling the Google suggest API [2]. Contrary to Free917, questions are not necessarily grammatical and are more colloquial. For example: *where did jackie kennedy go to college* and *what is spoken in czech republic*. Due to how they were selected, the questions are biased towards topics that are frequently asked from Google. According to [19], the *people* domain alone makes up about 7% of questions. Furthermore, the structure of questions tends to be simpler. Most questions only require a single entity with an answer relation [2]. Answers to the questions were obtained by using crowdsourcing. This introduces additional noise; in particular, for some questions only a subset of the correct answer is provided as gold answer. We use the original train-test split of the questions by [2] into 70% (3,778 questions) to train and 30% (2,032 questions) to test.

5.2 Evaluation measures

Given a benchmark and a system, denote the questions by $q_1...q_n$, the gold answers by $g_1...g_n$, and the answers from the system by $a_1...a_n$. Note that an answer can consist of a single value (in particular, a date or a literal) or a list of values. We consider the following two evaluation measures.

Accuracy The fraction of queries answered with the exact gold answer:

$$\text{accuracy} = \frac{1}{n}\sum_{i=1}^{n} I(g_i = a_i)$$

where $I(e)$ is an indicator function returning one if expression e is true and zero else. This is reasonable on Free917 which provides perfect gold answers.

Average F1 The average F1 across all questions:

$$\text{average F1} = \frac{1}{n}\sum_{i=1}^{n} F1(g_i, a_i)$$

where the function $F1$ computes F1 in the regular way. This accounts for partially correct results, which is reasonable for WebQuestions, where gold answers are sometimes incomplete.

In our evaluation we focus on accuracy for Free917 and average F1 for WebQuestions. These are the most reported and most intuitive measures for these datasets. We also performed the evaluation with other measures that were used in previous work, e.g., variants of F1 as defined in [15] and [25]. These provided no new insights and strongly correlated with the measures above.

	Free917		WebQuestions
Method	Accuracy+	Accuracy	Average F1
Cai+Yates	59 %	–	–
Jacana	–	–	35.4 %
Sempre	62 %	52 %	35.7 %
Kwiat. et al	68 %	–	–
Bordes et al	–	–	39.2 %
ParaSempre	68.5 %	46 %	39.9 %
Aqqu	**76.4 %**	**65.9 %**	**49.4 %**

Table 2: Results on the Free917 (267 questions) and WebQuestions (2032 questions) test set. For the results in the second column (Accuracy+) a manually crafted entity lexicon was used.

5.3 Systems evaluated

We evaluate and compare the following systems. See Section 2 for a brief description of the systems from previous work. If we (re-)produced results, we explicitly state so. Otherwise, we report existing results.

Cai+Yates The semantic parser developed by [7].

Kwiat. et al The semantic parser by [15].

Sempre The semantic parser by [2]. We produced results for Free917 without an entity lexicon using the provided code.[11]

ParaSempre The semantic parser suggested by [3]. We used the code provided by the authors[11] to produce results on Free917 without an entity lexicon.

GraphParser The semantic parser developed by [19]. We report results obtained from the code provided by the authors[12]. The results from their code slightly deviates from the results reported in their paper.

Jacana The information extraction based approach by [25]. We report updated results from [24].

Bordes et al The embedding-based model by [4].

Aqqu Our system, as described in Section 4. We want to stress that we use the exact same system on both benchmarks. As shown in Section 5.5 below, results can be further improved by adapting the feature set to the benchmark. However, we consider this overfitting. Note that all of the systems above, except Sempre and ParaSempre, were only evaluated on one of the two benchmarks.

5.4 Main results

Table 2 shows the results on the test sets for Free917 and WebQuestions for all the systems from Section 5.3. GraphParser is discussed separately below, because it was evaluated only on a subset of questions.

On Free917, Aqqu improves in accuracy over the best previous systems by 8% with an entity lexicon, and by 14% without entity lexicon. Performance drops considerably for all systems when not using an entity lexicon. This shows

[11] `http://github.com/percyliang/sempre`
[12] `http://github.com/sivareddyg/graph-parser`

	Top-2	Top-3	Top-5	Top-10
Free917	74.3 %	77.2 %	79.3 %	83.7 %
WebQuestions	67.1 %	72.7 %	77.5 %	82.3 %

Table 3: Top-k results on Free917 (top) and WebQuestions (bottom). Percentage of questions with the best answer in the top-k candidates.

	Free917		WebQuestions
Method	Acc+	Acc	Avg F1
Aqqu-point-lin	73.6 %	63.4 %	46.9 %
Aqqu-point-tree	74.3 %	63.0 %	47.9 %
Aqqu-pair-lin	76.4 %	65.2 %	48.3 %
Aqqu-pair-tree	76.4 %	65.9 %	49.4 %

Table 4: Results for different ranking variants on the test sets for Free917 and WebQuestions. For the results in the second column (Acc+) a manually crafted entity lexicon was used.

that addressing entity recognition is an integral part of the problem that cannot be ignored. Overall, we achieve an oracle accuracy (percentage of questions where at least one produced query candidate is correct) of 89.1% and 85.5%, with and without entity lexicon respectively. This indicates that there is still room for improvement for better matching and ranking.

On WebQuestions our system improves the state of the art by almost 10% in average F1. No system uses an entity lexicon. Note that the WebQuestions benchmark is much harder and contains a considerable amount of imperfect or wrong answers. Out of a random sample of 55 questions we found 9 questions that had a wrong answer, and 10 further question that had only a partially correct answer. This suggests that the upper bound for average F1 is roughly around 80%. Our oracle average F1 is at 68.5%. [2] and [3] report 48% and 63% respectively. Hence, we successfully identify most of the entities and relations. However, there is still much room for improvement in ranking and matching.
GraphParser was evaluated only on a subset of Freebase relations. The authors provide a train-test split of questions for WebQuestions. Note that we didn't restrict our system to the specific relations and that GraphParser requires an entity lexicon also on WebQuestions. Our system (without an entity lexicon) scores an average F1 of 66.1 % compared to 40.5 % reported for GraphParser. The selected subset of relations and thus questions seems to be considerably easier to answer for our system.

5.5 Detailed analysis

Top-k results Table 3 shows the top-k results on the two datasets. A large majority of questions can be answered from the top two or three candidates. By providing these interpretations and results (in addition to the top-ranked candidate) to a user, many questions can be answered correctly. Note that on WebQuestions some questions only have an imperfect gold answer with an F1 score smaller than one. Therefore, the percentage of best answers in the top candidates can be slightly larger than the resulting average F1.

Ranking variants As described in Section 4.6, we consider two possible ranking methods: pointwise (*point*) and pairwise (*pair*), each with two different ranking classifiers: logistic regression (*lin*) and random forests (*tree*). This gives a total of four different combinations.

Table 4 shows results of all four ranking variants with the full features of Table 1. On both benchmarks, pairwise ranking is more effective than pointwise ranking. This is consistent with our intuition that learning a pairwise comparator is better (see Section 4.6). Furthermore, random forests are slightly more effective than a weighted linear combination. We therefore use pairwise ranking with random forests as a standard choice.

	Free917	WebQuestions
best previous	52.0 %	39.9 %
best now	**69.2 %**	**49.4 %**
no n-grams, all other	69.2 %	39.6 %
no n-grams, no lit-match	65.2 %	39.6 %
no n-grams, no synonyms	61.6 %	28.2 %
n-grams, all other	65.9 %	49.4 %
n-grams, no pruning	64.4 %	49.3 %
n-grams, no synonyms	62.0 %	48.0 %
n-grams, nothing else	18.1 %	43.8 %

Table 5: Feature analysis for Free917 and WebQuestions. No synonyms disables features 11-15 and no lit-match features 2, 3, 9, 10 and 18. When not using the n-gram feature a different type of candidate pruning is performed (see text).

.

Feature analysis To gain insight into which features are helpful we evaluate our system with different combinations. Table 5 shows the results. Note that, as described in Section 4.7, without the n-gram feature hard pruning is applied. The following main observations can be made.

The n-gram feature is extremely helpful on WebQuestions but slightly detrimental on Free917. The WebQuestions benchmark contains many questions that are hard to answer without this kind of supervision, e.g., *where is reggie bush from?* (asking for the place of birth) or *what to do downtown san francisco?* (asking for tourist attractions). Our system is able to successfully learn important features for these from the training set. On the other hand, the small Free917 benchmark covers a wide range of domains and relations with only few repetitions. N-gram features aren't helpful on this dataset, which is shown by the low performance when only using the n-gram feature (18.1%). Note that the ranking and learning problem is inherently more difficult when the number of possible candidates increases. This is the case when not using hard pruning which goes along with using the n-gram feature (see Section 4.7). This disadvantage cannot be fully compensated by the weak n-gram feature and leaner pruning and, as a result, the score drops by about 3% for Free917. Still, we consider it more important to have a single approach that performs well on different kinds of datasets than to optimize for a single dataset.

Literal features provide a small benefit for Free917 but no benefit on WebQuestions. This is an artefact of the way Free917 was built. Free917 questions are tailored to Freebase, often using words from the relation name as part of the

question. Synonym features are important for both datasets. They give a huge benefit on WebQuestions without the n-gram feature but only a small benefit on top of it.

Finally, the pruning classifier used with the n-gram feature helps on Free917 because it allows to return "no answer" for some questions that have no answer in the knowledge base. The difference on WebQuestions (which always has an answer in the knowledge base) is not significant, and shows that the pruning classifier doesn't negatively affect performance.

Manual error analysis We manually inspected the errors our system makes. Many errors are due to mistakes in the benchmarks (partially or completely wrong gold answers) and inconsistencies in the knowledge base (different relations with contradicting answers on the same piece of information). We provide a list on our website, see the link in Section 1.1.

On that website, we also provide a list of errors due to our system. There is no single large class of errors worth pointing out though.

Efficiency We also evaluated the performance of our system. The average response time for a question is 644 ms for Free917 and 900 ms for WebQuestions.[13] None of the other system from Section 5.3 comes with an efficiency evaluation. For systems that provide code and for which we reproduced results, runtimes are (at least) several seconds per query. Training our system on the large WebQuestions benchmark takes about 90 minutes in total.

6. CONCLUSION

We have presented Aqqu, a new end-to-end system that automatically translates a given natural-language question to the matching SPARQL query on a knowledge base. The system integrates entity recognition and utilizes distant supervision and learning-to-rank techniques. We showed that our system outperforms previous state-of-the-art systems on two very different benchmarks by 8% and more. Aqqu answers questions interactively, that is, within one second.

For around 80% of the queries, the correct answer is among the top-5 candidates. This suggests that a more interactive approach, which asks the user's feedback for critical decisions (e.g., between two relations), could achieve a significantly further improved accuracy.

7. REFERENCES

[1] H. Bast, F. Bäurle, B. Buchhold, and E. Haussmann. Broccoli: Semantic full-text search at your fingertips. *CoRR*, abs/1207.2615, 2012.

[2] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic Parsing on Freebase from Question-Answer Pairs. In *EMNLP*, pages 1533–1544, 2013.

[3] J. Berant and P. Liang. Semantic Parsing via Paraphrasing. In *ACL*, pages 1415–1425, 2014.

[4] A. Bordes, S. Chopra, and J. Weston. Question Answering with Subgraph Embeddings. *CoRR*, abs/1406.3676, 2014.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[13] Answer times are averaged over three runs on a server with Intel E5649 CPUs, 90GB of RAM and warm SPARQL caches.

[6] C. J. C. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *NIPS*, pages 193–200, 2006.

[7] Q. Cai and A. Yates. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. In *ACL*, pages 423–433, 2013.

[8] A. X. Chang and C. D. Manning. Sutime: A library for recognizing and normalizing time expressions. In *LREC*, pages 3735–3740, 2012.

[9] ClueWeb, 2012. The Lemur Projekt.

[10] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *JAIR*, 10:243–270, 1999.

[11] C. Fellbaum. *WordNet*. Wiley Online Library, 1998.

[12] Y. Freund, R. D. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *JMLR*, 4:933–969, 2003.

[13] E. Gabrilovich, M. Ringgaard, and A. Subramanya. FACC1: Freebase annotation of ClueWeb corpora, Version 1.

[14] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[15] T. Kwiatkowski, E. Choi, Y. Artzi, and L. S. Zettlemoyer. Scaling Semantic Parsers with On-the-Fly Ontology Matching. In *EMNLP*, pages 1545–1556, 2013.

[16] T. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[17] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *ACL*, pages 55–60, 2014.

[18] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, pages 1003–1011, 2009.

[19] S. Reddy, M. Lapata, and M. Steedman. Large-scale Semantic Parsing without Question-Answer Pairs. *TACL*, 2:377–392, 2014.

[20] V. I. Spitkovsky and A. X. Chang. A Cross-Lingual Dictionary for English Wikipedia Concepts. In *LREC*, pages 3168–3175, 2012.

[21] M. Steedman. *The syntactic process*, volume 35. MIT Press, 2000.

[22] C. Unger, C. Forascu, V. Lopez, A. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter. Question answering over linked data (QALD-4). In *CLEF 2014*, pages 1172–1180, 2014.

[23] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR*, pages 391–398, 2007.

[24] X. Yao, J. Berant, and B. V. Durme. Freebase QA: Information Extraction or Semantic Parsing? In *ACL, Workshop on Semantic Parsing*, 2014.

[25] X. Yao and B. V. Durme. Information Extraction over Structured Data: Question Answering with Freebase. In *ACL*, pages 956–966, 2014.

[26] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, 1997.

Foundations and Trends® in Information Retrieval
Vol. 10, No. 2-3 (2016) 119–271

DOI: 10.1561/1500000032

now
the essence of knowledge

Semantic Search on Text and Knowledge Bases

Hannah Bast
University of Freiburg
bast@cs.uni-freiburg.de

Björn Buchhold
University of Freiburg
buchhold@cs.uni-freiburg.de

Elmar Haussmann
University of Freiburg
haussmann@cs.uni-freiburg.de

Contents

Abstract

This article provides a comprehensive overview of the broad area of semantic search on text and knowledge bases. In a nutshell, semantic search is "search with meaning". This "meaning" can refer to various parts of the search process: understanding the query (instead of just finding matches of its components in the data), understanding the data (instead of just searching it for such matches), or representing knowledge in a way suitable for meaningful retrieval.

Semantic search is studied in a variety of different communities with a variety of different views of the problem. In this survey, we classify this work according to two dimensions: the type of data (text, knowledge bases, combinations of these) and the kind of search (keyword, structured, natural language). We consider all nine combinations. The focus is on fundamental techniques, concrete systems, and benchmarks. The survey also considers advanced issues: ranking, indexing, ontology matching and merging, and inference. It also provides a succinct overview of natural language processing techniques that are useful for semantic search: POS tagging, named-entity recognition and disambiguation, sentence parsing, and word vectors.

The survey is as self-contained as possible, and should thus also serve as a good tutorial for newcomers to this fascinating and highly topical field.

H. Bast, B. Buchhold, E. Haussmann. *Semantic Search on Text and Knowledge Bases*. Foundations and Trends® in Information Retrieval, vol. 10, no. 2-3, pp. 119–271, 2016.

DOI: 10.1561/1500000032.

1

Introduction

1.1 Motivation for this Survey

This is a survey about the broad field of semantic search. Semantics is the study of meaning.[1] In a nutshell, therefore, it could be said that semantic search is *search with meaning*.

Let us first understand this by looking at the opposite. Only a decade ago, search engines, including the big web search engines, were still mostly *lexical*. By lexical, we here mean that the search engine looks for literal matches of the query words typed by the user or variants of them, without making an effort to understand what the whole query actually means.

Consider the query *university freiburg* issued to a web search engine. Clearly, the homepage of the University of Freiburg is a good match for this query. To identify this page as a match, the search engine does not need to understand what the two query words *university* and *freiburg* actually mean, nor what they mean together. In fact, the university homepage contains these two words in its title (and, as a

[1] The word comes from the ancient greek word *sēmantikós*, which means *important*.

matter of fact, no other except the frequent word *of*). Further, the page is at the top level of its domain, as can be seen from its URL: `http://www.uni-freiburg.de`. Even more, the URL consists of parts of the query words. All these criteria are easy to check, and they alone make this page a very good candidate for the top hit of this query. No deeper understanding of what the query actually "meant" or what the homepage is actually "about" were needed.[2]

Modern search engines go more and more in the direction of accepting a broader variety of queries, actually trying to "understand" them, and providing the most appropriate answer in the most appropriate form, instead of just a list of (excerpts from) matching documents.

For example, consider the two queries *computer scientists* and *female computer scientists working on semantic search*. The first query is short and simple, the second query is longer and more complex. Both are good examples of what we would call semantic search. The following discussion is independent of the exact form of these queries. They could be formulated as keyword queries like above. They could be formulated in the form of complete natural language queries. Or they could be formulated in an abstract query language. The point here is what the queries are asking for.

To a human, the intention of both of these queries is quite clear: the user is (most likely) looking for scientists of a certain kind. Probably a list of them would be nice, with some basic information on each (for instance, a picture and a link to their homepage). For the query *computer scientists*, Wikipedia happens to provide a page with a corresponding list and matching query words.[3] Correspondingly, the list is also contained in DBpedia, a database containing the structured knowledge from Wikipedia. But in both cases it is a manually compiled list, limited to relatively few better-known computer scientists. For the second query (*female computer scientists working on semantic search*), there is no single web page or other document with a corresponding

[2] In this simple example, we are leaving aside the important issue of *spam*. That is, someone deliberately putting misleading keywords in the title or even in the URL, in order to fool search engines, and thus users, to consider the web page relevant. Note that this query could also be solved using clickthrough data; see Section 1.2.2.

[3] `http://en.wikipedia.org/wiki/List_of_computer_scientists`

list, let alone one matching the query words. Given the specificity of the query, it is also unlikely that someone will ever manually compile such a list (in whatever format) and maintain it. Note that both lists are constantly changing over time, since new researchers may join any time.

In fact, even individual web pages matching the query are unlikely to contain most of the query words. A computer scientist does not typically put the words *computer scientist* on his or her homepage. A female computer scientist is unlikely to put the word *female* on her homepage. The homepage probably has a section on that particular scientist's research interests, but this section does not necessarily contain the word *working* (maybe it contains a similar word, or maybe no such word at all, but just a list of topics). The topic *semantic search* will probably be stated on a matching web page, though possibly in a different formulation, for example, *intelligent search* or *knowledge retrieval*.

Both queries are thus good examples, where search needs to go beyond mere lexical matching of query words in order to provide a satisfactory result to the user. Also, both queries (in particular, the second one) require that information from several different sources is brought together to answer the query satisfactorily. Those information sources might be of different kinds: (unstructured) text as well as (structured) knowledge bases.

There is no exact definition of what semantic search is. In fact, semantic search means a lot of different things to different people. And researchers from many different communities are working on a large variety of problems related to semantic search, often without being aware of related work in other communities. *This is the main motivation behind this survey.*

When writing the survey, we had two audiences in mind: (i) newcomers to the field, and (ii) researchers already working on semantic search. Both audiences should get a comprehensive overview of which approaches are currently pursued in which communities, and what the current state of the art is. Both audiences should get pointers for further reading wherever the scope of this survey (defined in Section 1.2

right next) ends. But we also provide explanations of the underlying concepts and technologies that are necessary to understand the various approaches. Thus, this survey should also make a good tutorial for a researcher previously unfamiliar with semantic search.

1.2 Scope of this Survey

1.2.1 Kinds of Data

This survey focuses on semantic search on text (in natural language) or knowledge bases (consisting of structured records). The two may also be combined. For example, a natural language text may be enriched with semantic markup that identifies mentions of entities from a knowledge base. Or several knowledge bases with different schemata may be combined, like in the Semantic Web. The types of data considered in this survey are explained in detail in Section 2.1 on *Data Types and Common Datasets*.

This survey does *not* cover search on images, audio, video, and other objects that have an inherently non-textual representation. This is not to say that semantic search is not relevant for this kind of data; quite the opposite is true. For example, consider a user looking for a picture of a particular person. Almost surely, the user is not interested in the precise arrangements of pixels that are used to represent the picture. She might not even be interested in the particular angle, selection, or lighting conditions of the picture, but only in the object shown. This is very much "semantic search", but on a different kind of data. There is some overlap with search in textual data, including attempts to map non-textual to textual features and the use of text that accompanies the non-textual object (e.g., the caption of an image). But mostly, search in non-textual data is a different world that requires quite different techniques and tools.

A special case of image and audio data are scans of text documents and speech. The underlying data is also textual[4] and can be extracted using optical character recognition (OCR) and automatic speech recognition (ASR) techniques. We do not consider these techniques in this

[4]Leaving aside aspects like a particular writing style or emotions when talking.

survey. However, we acknowledge that "semantic techniques", as described in this survey, can be helpful in the text recognition process. For example, in both OCR and ASR, a semantic understanding of the possible textual interpretations can help to decide which interpretation is the most appropriate.

1.2.2 Kinds of Search

There are three types of queries prevailing in semantic search: keyword, structured, and natural language. We cover the whole spectrum in this survey; see Section 2.2 on *Search Paradigms*.

Concerning the kind of results returned, we take a narrower view: we focus on techniques and systems that are *extractive* in the sense that they return elements or excerpts from the original data. Think of the result screen from a typical web search engine. The results are nicely arranged and partly reformatted, so that we can digest them properly. But it's all excerpts and elements from the web pages and knowledge bases being searched in the background.

We only barely touch upon the analysis of query logs (queries asked) and clickthrough data (results clicked). Such data can be used to derive information on what users found relevant for a particular query. Modern web search engines leverage such information to a significant extent. This topic is out of scope for this survey, since an explicit "understanding" of the query or the data is not necessary. We refer the user to the seminal paper of Joachims [2002] and the recent survey of Silvestri [2010].

There is also a large body of research that involves the complex synthesis of new information, in particular, text. For example, in *automatic summarization*, the goal is to summarize a given (long) text document, preserving the main content and a consistent style. In *multi-document summarization*, this task is extended to multiple documents on a particular topic or question. For example, *compile a report on drug trafficking in the united states over the past decade*. Apart from collecting the various bits and pieces of text and knowledge required to answer these questions, the main challenge becomes to compile these into a compact and coherent text that is well comprehensible for hu-

mans. Such non-trivial automatic content synthesis is out of scope for this survey.

1.2.3 Further inclusion criteria

As just explained, we focus on semantic search on text and knowledge bases that retrieves elements and excerpts from the original data. But even there we cannot possibly cover all existing research in depth.

Our inclusion criteria for this survey are very practically oriented, with a focus on fundamental techniques, datasets, benchmarks, and systems. Systems were selected with a strong preference for those evaluated on one of the prevailing benchmarks or that come with a working software or demo. We provide quantitative information (on the benchmarks and the performance and effectiveness of the various systems) wherever possible.

We omit most of the history and mostly focus on the state of the art. The historical perspective is interesting and worthwhile in its own right, but the survey is already long and worthwhile without this. However, we usually mention the first system of a particular kind. Also, for each of our nine categories (explained right next, in Section 1.3), we describe systems in chronological order and make sure to clarify the improvements of the newer systems over the older ones.

1.2.4 Further Reading

The survey provides pointers for further reading at many places. Additionally, we provide here a list of well-known conferences and journals, grouped by research community, which are generally good sources for published research on the topic of this survey and beyond. In particular, the bibliography of this survey contains (many) references from each of these venues. This list is by no means complete, and there are many good papers that are right on topic but published in other venues.

Information Retrieval: SIGIR, CIKM, TREC, TAC, FNTIR.

Web and Semantic Web: WWW, ISWC, ESWC, AAAI, JWS.

Computer linguistics: ACL, EMNLP, HLT-NAACL.

Databases / Data Mining: VLDB, KDD, SIGMOD, TKDE.

1.3 Overview of this Survey

Section 1.4 provides a *Glossary* of terms that are strongly related to semantic search. For each of these, we provide a brief description together with a pointer to the relevant passages in the survey. This is useful for readers who specifically look for material on a particular problem or aspect.

Section 2 on *Classification by Data Type and Search Paradigm* describes the two main dimensions that we use for categorizing research on semantic search:

Data type: text, knowledge bases, and combined data.

Search paradigm: keyword, structured, and natural language search.

For each data type, we provide a brief characterization and a list of frequently used datasets. For each search paradigm, we provide a brief characterization and one or two examples.

Section 3 on *Basic NLP Tasks in Semantic Search* gives an overview of: part-of-speech (POS) tagging, named-entity recognition and disambiguation (NER+NED), parsing the grammatical structure of sentences, and word vectors / embeddings. These are used as basic building blocks by various (though not all) of the approaches described in our main Section 4. We give a brief tutorial on each of these tasks, as well as a succinct summary of the state of the art.

Section 4 on *Approaches and Systems for Semantic Search* is the core section of this survey. We group the many approaches and systems that exist in the literature by data type (three categories, see above) and search paradigm (three categories, see above). The resulting nine combinations are shown in Figure 1.1. In a sense, this figure is the main signpost for this survey. Note that we use *Natural Language Search* and *Question Answering* synonymously in this survey. All nine subsections share the same sub-structure:

Profile ... a short characterization of this line of research
Techniques ... what are the basic techniques used
Systems ... a concise description of milestone systems or software
Benchmarks ... existing benchmarks and the best results on them

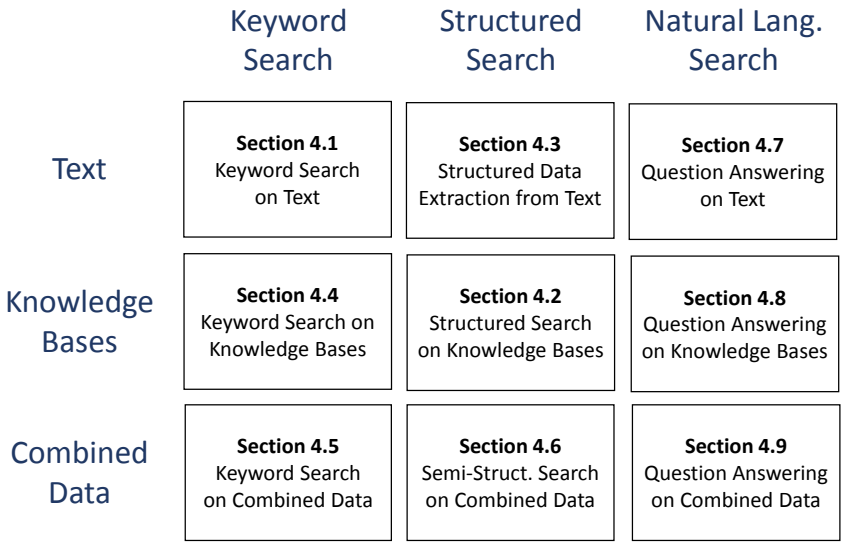

Figure 1.1: Our basic classification of research on semantic search by underlying data (rows) and search paradigm (columns). The three data types are explained in Section 2.1, the three search paradigms are explained in Section 2.2. Each of the nine groups is discussed in the indicated subsection of our main Section 4.

Section 5 on *Advanced Techniques for Semantic Search* deals with: *ranking* (in semantic entity search), *indexing* (getting not only good results but getting them fast), *ontology matching and merging* (dealing with multiple knowledge bases), and *inference* (information that is not directly contained in the data but can be inferred from it). They provide a deeper understanding of the aspects that are critical for results of high quality and/or with high performance.

Section 6 on *The Future of Semantic Search* provides a very brief summary of the state of the art in semantic search, as described in the main sections of this survey, and then dares to take a look into the near and the not so near future.

The article closes with a long list of 218 references. Datasets and standards are not listed as part of the References but separately in the Appendices. In the PDF of this article, all citations in the text are clickable (leading to the respective entry in the References), and so are

most of the titles in the References (leading to the respective article on the Web). In most PDF readers, *Alt+Left* brings you back to the place of the citation.

The reader may wonder about possible reading orders and which sections depend upon which. In fact, each of the six sections of this survey is relatively self-contained and readable on its own. This is true even for each of the nine subsections (one for each kind of semantic search, according to our basic classification) of the main Section 4. However, when reading such a subsection individually, it is a good idea to prepend a quick read of those subsections from Section 2 that deal with the respective data type and search paradigm: they are short and easy to read, with instructive examples. Readers looking for specific information may find the glossary, which comes right next, useful.

1.4 Glossary

This glossary provides a list of techniques or aspects that are strongly related to semantic search but non-trivial to find using our basic classification. For each item, we provide a very short description and a pointer to the relevant section(s) of the survey.

Deep learning for NLP: natural language processing using (deep) neural networks; used for the word vectors in Section 3.4; some of the systems in Section 4.8 on *Question Answering on Knowledge Bases* use deep learning or word vectors; apart from that, deep NLP is still used very little in actual systems for semantic search, but see Section 6 on *The Future of Semantic Search*.

Distant supervision: technique to derive labeled training data using heuristics in order to learn a (supervised) classifier; the basic principle and significance for semantic search is explained in Section 4.3.2 on *Systems for Relationship Extraction from Text*.

Entity resolution: identify that two different strings refer to the same entity; this is used in Section 4.3.4 on *Knowledge Base Construction* and discussed more generally in Section 5.4 on *Ontology Matching and Merging*.

Entity search/retrieval: search on text or combined data that aims at a particular entity or list of entities as opposed to a list of documents; this applies to almost all the systems in Section 4 that work with combined data or natural language queries[5]; see also Section 5.1, which is all about ranking techniques for entity search.

Knowledge base construction: constructing or enriching a knowledge base from a given text corpus; basic techniques are explained in Section 4.3.1; systems are described in Section 4.3.4.

Learning to rank for semantic search: supervised learning of good ranking functions; several applications in the context of semantic search are described in Section 5.1.

Ontology merging and matching: reconciling and aligning naming schemes and contents of different knowledge bases; this is the topic of Section 5.3.

Paraphrasing or synonyms: identifying whether two words, phrases or sentences are synonymous; systems in Section 4.8 on *Question Answering on Knowledge Bases* make use of this; three datasets that are used by systems described in this survey are: Patty [2013] (paraphrases extracted in an unsupervised fashion), Paralex [2013] (question paraphrases), and CrossWikis [2012] (Wikipedia entity anchors in multiple languages).

Question answering: synonymous with natural language search in this survey; see Section 2.2.3 for a definition; see Sections 4.7, 4.8, and 4.9 for research on question answering on each of our three data types.

Reasoning/Inference: using reasoning to infer new triples from a given knowledge base; this is the topic of Section 5.4.

Semantic parsing: finding the logical structure of a natural language query; this is described in Sections 4.8 on *Question Answering on Knowledge Bases* and used by many of the systems there.

Semantic web: a framework for explicit semantic data on the web; this kind of data is described in Section 2.1.3; the systems described

[5] A search on a knowledge base naturally returns a list of entities, too. However, the name *entity search* is usually only used when (also) text is involved and returning lists of entities is not the only option.

in Section 4.5 deal with this kind of data; it is important to note that many papers / systems that claim to be about semantic web data are actually dealing only with a single knowledge base (like DBpedia, see Table 2.2), and are hence described in the sections dealing with search on knowledge bases.

Information extraction: extracting structured information from text; this is exactly what Section 4.3 on *Structured Data Extraction from Text* is about.

XML retrieval: search in nested semi-structured data (text with tag pairs, which can be arbitrarily nested); the relevance for semantic search is discussed in Section 4.5.3 in the context of the INEX series of benchmarks.

2

Classification by Data Type and Search Paradigm

In this section, we elaborate on our basic classification of semantic search research and systems. The classification is along two dimensions:

Data type: text, knowledge bases, or combined data
Search paradigm: keyword, structured, and natural language search

In Section 2.1, we explain each of the three data types, providing a list of frequently used datasets for each type. In Section 2.2, we explain each of the three search paradigms along with various examples. The resulting nine combinations are shown in Figure 1.1.

Why this Classification

Coming up with this simple classification was actually one of the hardest tasks when writing this survey. Our goal was to group together research that, from a technical perspective, addresses similar problems, with a relatively clear delineation between different groups (much like in *clustering* problems). Most of the systems we looked at clearly fall into one of our categories, and no other classification we considered (in particular, refinements of the one from Figure 1.1) had that property. Of course, certain "gray zones" between the classes are inevitable;

these are discussed in the respective sections. For example, there is an interesting gray zone between *keyword* and *natural language* queries, which is discussed at the beginning of Section 2.2. Also, it is sometimes debatable whether a dataset is a single knowledge base or a combination of different knowledge bases, which counts as combined data in our classification; this is discussed in Section 2.1.2 on *Knowledge Bases*.

Also note that some other natural aspects are implicitly covered by our classification: for example, the *type of result* is largely implied by the type of data and the kind of search. Another complication (or rather, source of confusion) is terminology mixup. To give just one example, there is a huge body of research on the Semantic Web, but much of this work is actually concerned with a single knowledge base (like DBpedia, see Table 2.2), which requires mostly different techniques compared to true semantic web data, which is huge and extremely heterogeneous. Our Glossary in Section 1.4 should help to resolve such mixups, and, more generally, to locate (in this survey) material on a given technique or aspect.

Yet other aspects are orthogonal to our primary classification, for example: interactivity, faceted search, and details of the result presentation. These could be added with advantage to almost any system for semantic search. We briefly discuss such aspects in Section 2.3.

2.1 Data Types and Common Datasets

This section explains each of the three basic data types used in our classification above: natural language text, knowledge bases, and combinations of the two. For each type, we provide a list of frequently used datasets. All datasets are listed in a dedicated subsection of the References section. In the PDF of this article, the references in the tables below are clickable and lead to the corresponding entry in the Appendix.

2.1.1 Text

Definition 2.1. For the purpose of this survey, *text* is a collection of documents containing text, typically written in natural language. The

text need not be orthographically or grammatically correct. It may contain typical punctuation and light markup that exhibits some high-level structure of the text, like title and sections. There may also be hyperlinks between documents.

Remark: If there is markup that provides fine-grained annotations of parts of the text (e.g., linking an entity mention to a knowledge base), we count this as *Combined Data*, as discussed in Section 2.1.3.

Text is ubiquitous in the cyberworld, because it is the natural form of communication between humans. Typical examples are: news articles, e-mails, blog posts, tweets, and all kinds of web pages.

Web pages pose several additional challenges, like boilerplate content (e.g., navigation, headers, footers, etc., which are not actual content and can be misleading if not removed), spam, and dynamically generated content. We do not discuss these aspects in this survey. On the positive side, the hyperlinks are useful for search in general. Techniques for exploiting hyperlinks in the context of semantic search are discussed in Section 5.1.3 on *Ranking of Interlinked Entities*.

Commonly Used Datasets

Table 2.2 lists some collections of text documents that are often used in research on semantic search.

Reference	Documents	Size	zip	Type
[AQUAINT, 2002]	1.0 million	3.0 GB	n	news articles
[AQUAINT2, 2008]	0.9 million	2.5 GB	n	news articles
[Blogs06, 2006]	3.2 million	25 GB	n	blog posts
[ClueWeb, 2009]	1.0 billion	5.0 TB	y	web pages
[ClueWeb, 2012]	0.7 billion	5.0 TB	y	web pages
[CommonCrawl, 2007]	2.6 billion	183 TB	n	web pages
[Stream Corpus, 2014]	1.2 billion	16.1 TB	y	web pages[1]

Table 2.1: Datasets of natural language text used in research on semantic search.

The two AQUAINT datasets were heavily used in the TREC benchmarks dealing with question answering on text; see Section 4.7. The ClueWeb datasets are (at the time of this writing) the most used web-scale text collections. The CommonCrawl project provides regular snapshots (at least yearly) of a large portion of the Web in various languages. The Stream Corpus has been used in the TREC Knowledge Base Acceleration tracks (see Section 4.3 on *Structured Data Extraction from Text*) where knowledge about entities can evolve over time.

2.1.2 Structured Data / Knowledge Bases

Definition 2.2. For the purpose of this survey, a *knowledge base* is a collection of records in a database, which typically refer to some kind of "knowledge" about the world. By convention, records are often stored as triples in the form *subject predicate object*.

To qualify as a knowledge base, identifiers should[2] be used consistently: that is, the same entity or relation should have the same name in different records. Collections of records / triples from difference sources with different naming schemes are counted as *Combined Data*, which is discussed in Section 2.1.3.

Here are four example records from the Freebase dataset (see Table 2.2 below). The *ns:* is a common prefix, and the corresponding identifiers are URIs; see the subsection on data formats below.

ns:m.05b6w	*ns:type.object.name*	*"Neil Armstrong"*
ns:m.0htp	*ns:type.object.name*	*"Astronaut"*
ns:m.05b6w	*ns:people.person.profession*	*ns:m.0htp*
ns:m.05b6w	*ns:people.person.date_of_birth*	*"08-05-1930"*

Note that by the consistent use of identifiers we can easily derive information like *a list of all astronauts* or *astronauts born before a certain date*. We briefly discuss some related terminology and finer points.

[1] Web pages are timestamped, which allows treating the corpus as a stream of documents.

[2] A small fraction of inconsistencies are unavoidable in a large knowledge base and hence acceptable.

Ontologies: an ontology is the (typically hierarchical) system of types and relations behind a knowledge base. For example, the fact that astronauts are persons and that all persons are entities are typical ontological statements. WordNet [Miller, 1992] is a large ontology of the concepts of general-purpose real-world knowledge. In a sense, an ontology is therefore also a knowledge base, but on more "abstract" entities. The distinction is not always sharp, however. For example, WordNet also contains statements about "concrete entities", like in a typical knowledge base. Throughout this survey, we will consistently use the term knowledge base when referring to collections of records as defined above.

n-ary relations: It is easy to see that one can break down any structured data into triples, without loss of information. This is an instance of what is called *reification*. An example is given at the end of Section 2.1.3 (Christoph Waltz's Oscar).

n-tuples with $n > 3$: some knowledge bases also store tuples with more than three components. Typical uses are: adding provenance information (the data source of a triple), adding spatial or temporal information, assigning a unique id to a triple.

Triples vs. facts. vs. statements: the triples or n-tuples are sometimes referred to (somewhat optimistically) as facts or (more carefully) as statements. This does not mean that they are necessarily true. They may have entered the knowledge base by mistake, or they may just express an opinion. Still, very often they are "facts" in the common sense and it usually makes sense to think of them like that.

Graph representation: a knowledge base can also be thought of as a graph, where the nodes are the entities and the edges are the relations. When n-ary relations are involved, with $n > 2$, these edges become hyperedges (connecting more than two entities) and the graph becomes a hypergraph.

Commonly Used Datasets

Table 2.2 lists some often used knowledge bases. It is sometimes debatable when a dataset is a single knowledge base (as discussed in this

section) or a combination of different knowledge bases (as discussed in the next section). Our criterion, according to Definition 2.2 above, is whether the bulk of the data follows a consistent ontology / naming scheme. For example, the bulk of DBpedias's knowledge is stored in *dbpedia-owl:...* relations which are used consistently across entities. But there are also numerous *dbprop:...* relations, which correspond to a wide variety of properties from the Wikipedia infoboxes, which do not follow a strict underlying schema. Similarly, Freebase has numerous relations from its *"base"* domain, which partly fill in some interesting gaps and partly provide redundant or even confusing information.[3]

Reference	#Triples	#Entities	Size	Type
[YAGO, 2007]	20 M	2.0 M	1.4 GB	Wikipedia
[YAGO2s, 2011]	447 M	9.8 M	2.2 GB	Wikipedia
[DBpedia, 2007][4]	580 M	4.6 M	3.8 GB	Wikipedia
[GeoNames, 2006]	150 M	10.1 M	465 MB	geodata
[MusicBrainz, 2003]	239 M	45.0 M	4.1 GB	music
[UniProt, 2003]	19.0 B	3.8 B	130 GB	proteins
[Freebase, 2007]	3.1 B	58.1 M	30 GB	general
[Wikidata, 2012]	81 M	19.3 M	5.9 GB	general

Table 2.2: Knowledge bases used in research on semantic search. All sizes are of the compressed dataset.

We also remark that usually only a fraction of the triples in these knowledge bases convey "knowledge" in the usual sense. For example, the YAGO dataset contains about 3 million facts stating the length of each Wikipedia page. Freebase contains 10 million facts stating the keys of all Wikipedia articles. DBpedia has millions of *rdf:type* triples relating entities to the countless synsets from WordNet. Also, many facts are redundant. For example, in Freebase many relations have an

[3]For example, the type *base.surprisingheights.surprisingly_short_people* with only fifteen entities, including Al Pacino.

[4]The number of triples and entities are for the English version. The multilingual version features 3 billion triples and 38.8 million entities.

inverse relation with the same statements with subject and object reversed. According to Bordes and Gabrilovich [2015], the number of non-redundant triples in Freebase is 637 million, which is about one third of the total number stated in the table above.

On December 16, 2014 Google announced that it plans to merge the Freebase data into Wikidata and then stop accumulating new data in Freebase. Freebase became read-only on March 30, 2015. At the time of this writing, Wikidata was still relatively small, however.

Data Formats

A knowledge base can be stored in a general-purpose relational database management system (RDBMS), or in special-purpose so-called triple stores. The efficiency of the various approaches is discussed in Section 4.2 on *Structured Search on Knowledge Bases*.

On the Web, knowledge base data is often provided in the form of RDF (Resource Description Framework). RDF is complex, and we refer the reader to Wikipedia or W3C for a complete description. What is notable for this survey is that in RDF, identifiers are provided by a URI (Universal Resource Identifier) and hence globally unambiguous.

Also note that RDF is an abstract description model and language, not an explicit format. For practical purposes, many text serializations exist. The first such serialization was proposed in 1999 by the W3C and was based on XML, and thus very verbose. At the time of this writing, less verbose text serializations are commonly used:

N-triples: the triples are stored in a text file, with a space between subject, predicate, and object, and a simple dot to separate triples (usually one triple per line).

N-quads: like N-triples, but with one additional field per triple that provides an arbitrary context value (e.g., the source of the triple).

TSV: one triple or quad per line, with the three or four components separated by a tab. TSV is an abbreviation for tab-separated values.

Turtle: allows an explicit nested representation. Depending on the data, this can be more convenient for reading and producing than mere

triples or quads. The price is a more complex format that requires more complex parsers.

2.1.3 Combined Data

Text and knowledge bases are both natural forms to represent knowledge. Text is the most natural form for humans to produce information. A knowledge base is the most natural form to store information that is inherently structured in the first place. Therefore, it makes sense to combine data of these two types into a maximally comprehensive dataset It also makes sense to consider multiple knowledge bases, since a single knowledge base is usually limited to a certain scope. Of course, it also makes sense to combine different text collections, but that is trivial since there is no common structure or naming scheme to obey.

Definition 2.3. For the purpose of this survey, *combined data* is obtained by one or both of the following two principles:

link: link a text to a knowledge base by recognizing mentions of entities from the knowledge base in the text and linking to them

mult: combine multiple knowledge bases with different naming schemes (such that the same entity or relation may exist with different names)

Both of these are used extensively in research in order to obtain what we call *combined data* here. In the list of commonly used datasets in Table 2.3 below, it is indicated which dataset makes uses of which subset of these principles. Note that realizing "link" is equivalent to solving the named-entity recognition and disambiguation problem discussed in Section 3.2.

Commonly Used Datasets

Table 2.3 lists a number of popular datasets of the "combined" type. The number of "triples" for the Wikipedia LOD dataset, the two ClueWeb FACC datasets, and the FAKBA1 dataset is the number of entity mentions in the text that were linked to an entity from the knowledge base (YAGO and DBpedia for the Wikipedia LOD dataset, Freebase for the FACC and FAKBA1 dataset). Note that the ClueWeb

Reference	#Triples	Size	Type
[Wikipedia LOD, 2012]	70 million	61 GB	link
[ClueWeb09 FACC, 2013]	5.1 billion	72 GB	link
[ClueWeb12 FACC, 2013]	6.1 billion	92 GB	link
[FAKBA1, 2015]	9.4 billion	196 GB	link
[BTC, 2009]	1.1 billion	17 GB	mult
[BTC, 2010]	3.2 billion	27 GB	mult
[BTC, 2012]	1.4 billion	17 GB	mult
[WDC, 2012]	17.2 billion	332 GB	link+mult

Table 2.3: Commonly used datasets of the "combined" type. The last column indicates which combination principles were used, according to the typology explained at the beginning of the section.

FACC and FAKBA1 datasets only consist of the annotations and do not include the full text from ClueWeb or the Stream Corpus from the TREC Knowledge Base Acceleration track. The three BTC datasets were obtained from a crawl of the Semantic Web, started from a selection of seed URIs. Note that the BTC 2012 dataset contains all of DBpedia and a selection of Freebase, which are both listed in Table 2.2 as individual knowledge bases. The WDC (Web Data Commons) dataset is obtained by extracting structured data from CommonCrawl (see Table 2.1). Both BTC and WDC are considered "semantic web data", which is explained in more detail below.

Text Linked to a Knowledge Base

The natural format to encode *link* information in text is XML. Here is an example excerpt from the Wikipedia LOD collection from Table 2.3 above.

```
<paragraph> Mt. Morris is home of the <link>
<wikilink href="13135902.xml">Illinois Freedom Bell</wikilink>
<dbpedia href="http://dbpedia.org/.../Illinois_Freedom_Bell">
</dbpedia><yago ref="Illinois_Freedom_Bell"></yago>
</link>, which is located in the town square. [...]</paragraph>
```

The main tasks of the INEX (Initiative for the Evaluation of XML retrieval) series of benchmarks, discussed in Section 4.5.3, work with this collection.

However, note that for the purposes of annotation, XML is a mere convention, not a necessity. For example, the two FACC collections from Table 2.3 above provide links between the ClueWeb collections (from Table 2.1) and Freebase (from Table 2.2) as follows:

PDF	*21089*	*21092*	*0.9976*	*m.0600q*
FDA	*21303*	*21306*	*0.9998*	*m.032mx*
Food and Drug Administration	*21312*	*21340*	*0.9998*	*m.032mx*

The first column is the name of the entity in the text, the second and third columns specify the byte offsets in the file, the fourth column is the confidence of the link, and the fifth column is the Freebase id.

Semantic Web

The Semantic Web (SW) is an effort to provide "combined data" in the sense above at a very large scale. The data from the Semantic Web is often also called *linked open data (LOD)*, because contents can be contributed and interlinked by anyone, just like web pages (but in a different format, see below). With respect to search, these are secondary aspects. Throughout this survey, we therefore relate to this kind of data as simply *semantic web data*. It makes uses of both principles of combining data, as defined above:

link: provided by semantic markup for ordinary web pages.

mult: anyone can contribute + absence of a global schema.

The "mult" principle is realized via RDF documents that can link to each other, just like ordinary web pages can link to each other. For example, here is an excerpt from the RDF page for the French city Embrun (the showcase page of the GeoNames knowledge base from Table 2.2). Note the use of prefixes like *rdf:* and *gn:* in the URI to keep the content compact and readable also for humans.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
        xmlns:gn="http://www.geonames.org/ontology#" ... >
<gn:Feature rdf:about="http://sws.geonames.org/3020251/">
<gn:name>Embrun</gn:name>
<gn:countryCode>FR</gn:countryCode>
<gn:population>7069</gn:population>
```

The "link" principle is realized via semantic markup that is embedded into regular web pages. For example, here is an excerpt from an HTML page using so-called *Microdata* markup.

```
<title>Michael Slackenerny's Homepage</title>
<section itemscope itemtype="http://schema.org/Person">
Hi, my name is <span itemprop="name">Michael Slackenerny</span>.
I am a <span itemprop="jobTitle">postdoctoral student</span> at
<span itemprop="affiliation">Stanford Univ</span>.</section>
```

The four most widely used formats for semantic markup are as follows. The first three simply use HTML tags with existing or new attributes.

Microdata: uses the dedicated attributes *itemscope* and *itemprop*

Microformats: (ab)uses the *class* attribute

RDFa: uses RDF-style attributes *about*, *property*, and *contents*

JSON-LD: uses JavaScript to provide Microdata-like markup

The advantage of JSON-LD over the other formats is that it allows a cleaner separation of the ordinary content from the semantic content (much in the spirit of frameworks like CSS, or libraries like jQuery). As of this writing, all four of these formats are still widely used, and no clear winner has emerged yet.

The use of semantic markup has increased steadily over the last years. According to [Meusel, Petrovski, and Bizer, 2014], the number of RDF quads (all of the above, except JSON-LD) in the WDC dataset (see Table 2.3 below) has increased from 5.2 billion in the 2010 dataset, to 7.4 billion in 2012, to 17.2 billion in 2013. Between 2012 and 2013, the fraction of analyzed HTML pages and domains that use semantic markup has more than doubled, up to 26% of pages and 14% of domains in 2013. More recent statistics can be found in [Guha, Brickley, and

MacBeth, 2015]. Here, a sample of 10 billion pages from a combination of the Google index and Web Data Commons is analyzed. 31% of pages have Schema.org markup, as discussed in the next subsection.

Challenges of Semantic Web Data

It is important to understand that semantic web data is *not* a single knowledge base as defined in the previous section, because there is no consistent naming scheme. This is not by accident, but rather by design. Given the heterogeneity of contents and its providers on the Web, it seems illusory to establish standard names for everything and expect everyone to stick with it. The Semantic Web takes a minimalist approach in this respect: content providers can use whatever names they like, they only have to be globally unambiguous (URIs). We briefly discuss three resulting challenges.

Standards: One (mostly social) approach is to enable and encourage contributors to reuse existing schemes as much as possible. For common concepts, this is already happening. One of the earliest schemes was FOAF [2000] (Friend Of A Friend), which provided standard names for relations related to a person, like: given names, family name, age, and who knows who. A more recent and larger effort is Schema.org [2011], collaboratively launched by Bing, Google, and Yahoo. Schema.org provides URIs for concepts such as creative works, events, organizations, persons, places, product, reviews, which are relevant for many popular web search queries.

Explicit links: Another approach is to enable contributors to provide explicit links between different names for the same entities or concepts, via meta statements like *<owl:sameAs>*. An example for different names for the same entity is: *Barack Obama* and *Barack H. Obama* and *B. Obama*. Such links could also be identified by automatic methods. This is discussed in more detail in Section 5.3 on *Ontology Matching and Merging*.

Model mismatch: A more complex problem are relations, which can not only be named differently, but also modeled in different ways. For example, consider the YAGO and Freebase knowledge bases from Table

2.2. In YAGO, the relation *won award* is used to express who won which award. For example

Christoph Waltz *won-award* *Oscar for Best Supporting Actor*

In Freebase, that information is modeled via a so-called mediator object, which has id *ns:m.0r4b38v* in the following example. For the sake of readability, we replaced the ids of the other objects with human-readable names and shortened the relation names.

Christoph Waltz	*award*	*ns:m.0r4b38v*
ns:m.0r4b38v	*name*	*Oscar for Best Supporting Actor*
ns:m.0r4b38v	*year*	*2012*
ns:m.0r4b38v	*movie*	*Django Unchained*
ns:m.0r4b38v	*role*	*Dr. King Schulz*

The approach taken by YAGO is simpler, the approach taken by Freebase allows to capture more information.

2.2 Search Paradigms

We distinguish between three major search paradigms: keyword search, structured search, and natural language search. Each of them is explained in one of the following subsections. As a first approximation, think of the name of the paradigm to describe the type of the query that is being asked.

keyword search: just type a few keywords

structured search: a query in a language like SQL or SPARQL

natural language: a complete question, as humans typically pose it

Before we describe each of these paradigms, let us comment on some of the finer points of this categorization:

Kind of result: We also considered a (further) categorization by the kind of result but this turned out to be impractical. The kind of result usually follows from the type of query and the type of data that is being searched. Sometimes there are variations, but that usually does not affect the fundamental approach. This is briefly explained in each of the following subsections, and in more detail in the various subsections of Section 4 on *Approaches and Systems for Semantic Search*.

Search on combined data: When dealing with combined data (in the sense of Section 2.1.3), there are two prevailing search paradigms. One is basic keyword search, optionally extended with a specification of the desired type of result entities (for example, *astronauts who walked on the moon* with a restriction to entities of type *person*). This kind of search is discussed in Section 4.5. The other is structured search that is extended with a keyword search component (there are many variants for the semantics of such an extension). This kind of search is discussed in Section 4.6.

Keyword vs. natural language: Keyword search and natural language search are less clearly delineated than it may seem. For example, consider the simple query *birth date neil armstrong*. A state-of-the-art system for keyword search on text will return (on a typical corpus, say the Web) a prominent document that contains the query words in prominent position (say, the Wikipedia article of Neil Armstrong). This document will probably also contain the desired piece of information (his birth date). We classify such a system under keyword search. A state-of-the-art question answering system might understand the very same query as an abbreviated natural language query (*what is the birth date of neil armstrong*), much like a human would do when seeing the four keywords above. It would then return the corresponding answer in a human-friendly form. We classify such a system under natural language search.

An example for the opposite case would be *how to tell gzip to leave the original file*. This is clearly a natural language query. But, at the time of this writing, there happens to be a web page with exactly that title and the complete answer as content. Any state-of-the-art system for keyword search will easily find this web page, without any semantic analysis of the query whatsoever.

The bottom line is that the distinction between keyword search and natural language search is best made not by the apparent form of the query, but by the basic technique used to process the query. From this point of view, it is then usually clear to which of the two categories a given system belongs.

2.2.1 Keyword Search

Query	Keywords (typically few)
Example 1	*space flight*
Example 2	*apollo astronauts*
Result	Documents or entities (or both) that are "relevant" to the information need
Example 1	*Documents on the topic of space flight*
Example 2	*Astronauts from the Apollo missions*
Strength	Easy and quick for the user to query
Weakness	Often hard to guess the precise information need, that is, what it means to be "relevant"

This is still the most ubiquitous search paradigm, and the one we are most familiar with. All of the major web search engines use it. The user types a few keywords and gets a list of matching items in return. When the data is text, matching items are (snippets of) documents matching the keywords. When the data is a knowledge base, matching items are entities from the knowledge base. With combined data, the result is a combination of these, usually grouped by entity.

The query processing is based on matching the components of the query to parts of the data. In the simplest case, the keywords are matched literally. In more sophisticated approaches, also variants of the keywords are considered as well as variants or expansions of the whole query. This is explained in more detail in Section 4.1 on *Keyword Search on Text*. With respect to our classification, all such techniques still count as keyword search.

In contrast, natural language search tries to "understand" the query. This often means that the query is first translated to a logical form. Then that logical form is matched to the data being searched.

Recall from the discussion at the beginning of this section that even the most basic form of keyword search (which looks for literal matches

of the query words) can answer a complex natural language query, when there is a document with exactly or almost that question in, say, the title. We still consider that keyword search in this survey.

2.2.2 Structured Search

Query *Example*	Structured query languages like SPARQL *SELECT ?p WHERE {* *?p has-profession Scientist .* *?p birth-date 1970 }*
Result *Example*	Items from the knowledge base matching the query; the order is arbitrary or explicitly specified (using an ORDER BY clause in SPARQL) *Scientists born in 1970*
Strength	Maximally precise "semantics" = it is well-defined, which items are relevant to the query
Weakness	Cumbersome to construct queries; hard to impossible to guess the correct entity and relation names for large knowledge bases

Structured query languages are the method of choice when the data is inherently structured, as described in Section 2.1.2 on *Knowledge Bases*. Then even complex information needs can be formulated without ambiguity. The price is a complex query language that is not suited for ordinary users. For large knowledge bases, finding the right entity and relation names becomes extremely hard, even for expert users. Interactive query suggestions (see Section 2.3) can alleviate the problem, but not take it away.

The example query in the box above is formulated in SPARQL [2008], the standard query language for knowledge bases represented via triples. SPARQL is a recursive acronym for: SPARQL Protocol and RDF Query Language. It is very much an adaption of SQL [1986], the standard query language for databases. SQL is an acronym for:

Structured Query Language. The translation from SPARQL to SQL is discussed in Section 4.2 on *Structured Search in Knowledge Bases*.

On combined data (as discussed in Section 2.1.3), structured search requires an extension of SPARQL by a text search component. A simple realization of this is discussed in Section 4.2.1, a semantically deeper realization is discussed in Section 4.6.1.

2.2.3 Natural Language Search

Query *Example*	Natural language queries, often starting with one of the 5W1H: Who, What, Where, When, Why, How *Who won the oscar for best actor in 2015 ?*
Result *Example*	The correct answer, in human-friendly form *Eddie Redmayne ... and maybe some disambiguating information, like a picture, his nationality, and his birth date*
Strength	Most natural for humans, suitable for speech input
Weakness	Ambiguity in natural language; queries can be very complex; queries can require complex reasoning

This is the most natural form of communication for humans. In the simplest case, queries ask for an entity from a single relationship, like in the example above. More complex queries may ask for the combination of several relationships, for example:

what is the gdp of countries with a literacy rate of under 50%

A query may also ask several questions at once, for example:

what is the population and area of germany

Natural language search may also accept and correctly process keyword queries, for example: *oscar best actor 2015* . As explained at the beginning of Section 2.2, the yardstick is not the apparent form of the query but the technique used to process it.

Questions that require the synthesis of new information or complex reasoning are out of scope for this survey. For example:[5]

HAL, despite your enormous intellect, are you ever frustrated by your dependence on people to carry out your actions?

2.3 Other Aspects

Our basic classification focuses on the main aspects of a search paradigm: the kind of queries that are supported and the basic technique used to process these queries. In actual systems, a variety of other aspects can play an important role, too. We here name three particularly important and widely used aspects:

Interactivity: The search engine may provide autocompletion, query suggestions, and other means to aid the query construction. This is particularly important for semantic search. The query language may be more complex, and thus unaided query construction may be hard, even for an expert. Precise formulation of the names of entities or relations can be key to get meaningful results.

Interactivity can also help the system to get more information from the user (on the query intent), which can help result quality. For example, the query suggestions of Google steer users towards queries that the search engine can answer well.

Faceted Search: Along with the results, the search engine may provide various kinds of categories to narrow, broaden, or otherwise modify the search. Again, this is particularly important for semantic search. For complex queries and when the result is not simply a single entity or web page, one iteration may simply not be enough to get to the desired results.

Result presentation: When searching for entities, it is often useful to group the results by entity and accompany them with additional information, like an appropriate picture. When the entity was extracted from text, it can be useful to show a containing snippet of appropriate

[5] Question to the HAL 9000 computer in the movie "2001: A Space Odyssey".

size. When the entity comes from a knowledge base, it can be useful to show some related information. When searching for entities that have a geographical location (like cities or events), it is often useful to depict those locations on a map. The map may be interactive in a number of ways. For example, hits can be represented by markers and clicking on a marker provides more detailed information or zooms in. When searching for events, they could be shown on a timeline.

Note that these extensions make sense for any data type and any search paradigm discussed in the previous sections. For example, if the results are cities, it makes sense to show them on a map, no matter how the query was formulated, and no matter whether the information came from a knowledge base or from the Semantic Web.

Several of the systems described in our main Section 4 on *Approaches and Systems for Semantic Search* implement one or several of these extensions. We provide details when we describe the respective systems. Apart from that, we do not delve deeper into these aspects in this survey. We do acknowledge though that proper user interfaces and result presentation are essential for the success of semantic search. In fact, at the time of this writing, all the major search engines already have basic features for each of the aspects discussed above.

3

Basic NLP Tasks in Semantic Search

Semantic search is about search with meaning. In text, this meaning is expressed in natural language. Even a knowledge base, where much of the meaning is implicit in the structure, has elements of natural language, for example, in the (sometimes rather long) relation names or in the object literals (which, in principle, can contain arbitrary text).

The following subsections discuss four basic techniques to capture aspects of the meaning of natural language text: POS tagging and chunking, entity recognition and disambiguation, sentence parsing, and word vectors. These four techniques should be in the toolbox of every researcher in the semantic search field.

Many of the systems described in our main Section 4 use one or more of these techniques. However, even NLP-agnostic approaches can achieve remarkable results for certain query classes. This is particularly true for keyword search on text, as discussed in Section 4.1. Still, there is no doubt that for queries above a certain complexity, natural language understanding is essential. This is discussed further in Section 6 on *The Future of Semantic Search*.

3.1 Part-of-Speech Tagging and Chunking

In *Part-of-Speech (POS) tagging*, the task is to assign to each word from a sentence a tag from a pre-defined set that describes the word's grammatical role in the sentence.

Definition 3.1. Given a sentence $s = (w_1, \dots w_n)$ and a set of available POS tags T, POS tagging outputs a sequence $t_1, t_2, \dots t_n$ that assigns each word w_i a corresponding tag $t_i \in T$.

Some typical POS tags are: NN (noun), VB (verb), adjective (JJ), RB (adverb). Here is a POS-tagged example sentence using all of these:

Semantic/JJ search/NN is/VB just/RB great/JJ.

Depending on the application, POS tags of different granularity can be considered. For example, the tags may distinguish between singular (NN) and plural (NNS) nouns. Or between a regular adverb (RB), a comparative (RBR), and a superlative (RBS).

A closely related problem is that of *chunking*, sometimes also referred to as *shallow parsing*. The task of chunking is to identify and tag the basic constituents of a sentence, based on the POS-tagged words.

Definition 3.2. Given a sentence $s = (w_1, \dots w_n)$ and a set of available chunk tags C, chunking outputs word sequences identified by triples (s_i, e_i, c_i) where s_i is the start index, e_i the end index and $c_i \in C$ the chunk type of chunk i. The chunks don't overlap and don't have to cover all of the words.

Some typical chunking tags are: NP (noun phrase), VB (verb phrase), ADJP (adjective phrase). A possible chunking of the example sentence above, using all these tags, is:

NP(Semantic/JJ search/NN) VB(is/VB) ADJP(just/RB great/JJ).

Chunking is a natural first step for both entity recognition and sentence parsing, which are discussed in the two subsections following this one.

3.1.1 Benchmarks and State of the Art

A classical benchmark is provided as part of the Penn Treebank [Marcus, Santorini, and Marcinkiewicz, 1993]. We describe it in more detail in Section 3.3 on *Sentence Parsing* below.

Both POS tagging and chunking can be solved fast and with high accuracy. For example, the well-known and widely-used Stanford POS tagger [Toutanova et al., 2003] achieves an accuracy of 97% on the Penn Treebank-3 [1999] dataset (2,499 stories from the Wall Street Journal). This is close to the accuracy achieved by human experts (which is also not perfect). Tagging speed was reported to be around 15,000 words per second, on a typical server in 2008.[1] In an experiment on a current server with Intel Xeon E5-1650 (3.50GHz) CPUs, the Stanford POS tagger was able to tag around 55,000 words per second.

An accuracy of 97% sounds impressive, but when looking at whole sentences this means that only 56% of the sentences are tagged without any error. But a fully correct tagging is important for sentence parsing; see the subsection below. Manning [2011] explores options to achieve a per-word accuracy of close to 100%.

3.1.2 Application to Short Text and Queries

For very short text, e.g., queries, the methods described here are less successful. For example, in the query *pink songs*, the word "pink" certainly refers to the pop artist and not the color. However, a typical POS tagger is not used to the telegraphic form of queries and would thus incorrectly tag "pink" as an adjective.

Hua et al. [2015] present an approach to solve variants of POS tagging and chunking for short texts. Unlike for regular text, the chunking is done before tagging. Throughout the whole process, decisions are made based on semantics, in particular, the coherence between candidate chunks and tags. This distinguishes the approach from those for larger texts where the structure of grammatically well-formed sentences plays a central role and probabilities of chains of tags determine the outcome. In a subsequent step, the approach also solves a variant

[1] Reported on `http://nlp.stanford.edu/software/pos-tagger-faq.shtml`.

of the named-entity recognition and disambiguation problem that is described in the next subsection.

Short text understanding is very useful for question answering since it provides a semantic interpretation of the query. Thus, systems from Section 4.8 and Section 4.9 often include components that address this problem or variants tailored towards their particular use case.

3.2 Named-Entity Recognition and Disambiguation

Assume we are given a collection of text documents and a knowledge base. In the simplest case, the knowledge base is just a list of entities with their common name or names. Additional knowledge on these entities may be helpful for the task defined next.

The task of *Named-Entity Recognition (NER)* is to recognize which word sequences from the text documents might refer to an entity from the knowledge base. For example, in the following sentence, all word sequences referring to an entity which has its own page in the English Wikipedia (as of the time of this writing), are underlined:

Buzz Aldrin joined Armstrong and became the second human to set foot on the Moon.[2]

Definition 3.3. Given some text and a set of entity types T, NER outputs word sequences which mention a named entity. The mentions are identified by triples (s_i, e_i, t_i) where s_i is the start index, e_i the end index and $t_i \in T$ the entity's type. The entity mentions don't overlap and aren't nested.

Note that usually no knowledge base is required for NER. The task is only to identify possible entity mentions (typically proper nouns) which refer to an entity from a few classes. Most typically, these are: *person*, *location*, and *organization*. For example, the Stanford NER tagger [Finkel, Grenager, and Manning, 2005] is of this kind. If entities are linked to a knowledge base in a subsequent step, the knowledge base can be a valuable resource for NER already, e.g., in the form of a gazetteer.

[2] It is often arguable what exactly constitutes a *named entity*. For example, in some cases it may be desirable to tag *human* and *foot* as well.

The task of *Named-Entity Disambiguation (NED)* follows up on NER. The task of NER is just to identify the word sequences, that is, in the example above, underline them (and assign them a course type). The task of NED is to decide for each identified sequence to exactly which entity from the knowledge base it refers.

Definition 3.4. Given a knowledge base with entities E, some text and a set of possible entity mentions (s_i, e_i, t_i) from NER, the task of NED is to assign for each entity mention an entity from $E \cup \emptyset$. If no entity from E is mentioned, $\emptyset$ should be assigned.

For example, in the sentence above, the word *Moon* could refer to any of the many moons in our solar system, or to the generic *Moon*, or to one of the many people named *Moon*. However, it is clear from the context of the sentence that the one moon from planet Earth is meant. Likewise, the word *Armstrong* could refer to many different people: Lance Armstrong, Louis Armstrong, Neil Armstrong etc. Again, the context makes it clear that Neil Armstrong is meant. The task of NED is to establish these "links" (indeed, NED is sometimes also referred to as *Named-Entity Linking*).

Also note that, in the example above, the word *Buzz* on its own could refer to a number of different entities: there are many people with that name, there is a film with that name, there is a series of video games with that name. It is part of the NER problem to find out that the entity reference in this sentence consists of the word sequence *Buzz Aldrin* and not just of the word *Buzz*.

For semantic search systems, we usually require NER and NED together. In the overview below, we therefore only consider the state of the art of research that considers both problems together.

3.2.1 Co-reference and Anaphora Resolution

It is a frequent phenomenon in natural language texts that an entity is referred to not by its name but by a placeholder word. For example, consider the following sentences:

The stalks of rhubarb are edible. Its roots are medicinally used. The leaves of the plant are toxic.

Co-reference and anaphora resolution means to identify all mentions that refer to the same entity. The first underlined word (*rhubarb*) is an entity reference in the sense above. The second underlined word (*its*) is a pronoun, which in this case refers to the rhubarb from the sentence before. The third underlined word (*the plant*) is a noun phrase, which in this context does not refer to a plant in general, but again to the rhubarb from the sentence before.

The three references together are called *co-references* because they refer to the same entity. The last two references are called *anaphora*, because they refer to an entity mentioned earlier in the text. Note that anaphora are not needed for expressiveness, but for the sake of brevity (pronouns are short) or variety (to avoid repeating the same name again and again).

There are many variations of this task depending on the context. Co-references can be within or across documents. Only references between noun-phrases can be considered or also between events described by whole sentences.

For some of the semantic search systems described in Section 4 on *Approaches and Systems for Semantic Search*, it is important that as many entity references are recognized and disambiguated as possible (high recall). In that case, anaphora resolution is just as important as NER and NED. However, in papers or benchmarks solely about NER or NED, anaphora resolution is usually not included as part of the problem statement.

A recent survey on anaphora resolution is provided in the book by Mitkov [2014]. Supervised approaches for co-reference resolution of noun-phrases are surveyed by Ng [2010].

3.2.2 Benchmarks and State of the Art

At the time of this writing, there were two active benchmarks: TAC and ERD. We briefly describe the setting and best results for each.

TAC: The *Text Analysis Conference (TAC)* is co-organized by the National Institute of Standards (NIST) and the Linguistic Data Consortium (LDC). The first TAC was held in 2008, and has since replaced the *Automatic Content Evaluation (ACE)* series, which had similar

goals and was last held in 2008. From 2009 until the time of this writing, TAC contained a track called *Entity Linking* [Ji, Nothman, and Hachey, 2014], with a new benchmark every year. [3] In the benchmarks before 2014, the offsets of the word sequences that have to be disambiguated are given as part of the problem. That is, the NER part, according to our definition above, is already solved. For the NED part, an additional challenge is added in that some of the word sequences do not refer to any entity from the given knowledge base. It is part of the problem, to identify these as new entities and group them accordingly if there are several references to the same new entity. In 2014, a new end-to-end English entity discovery and linking task was introduced. This task requires to automatically extract entity mentions, link them to a knowledge base, and cluster mentions of entities not in the knowledge base.

The knowledge base used in all of the tasks was a collection of entities (800K of them in 2013) of type person, organization, or location from a dump of the English Wikipedia from October 2008. The corresponding entries from the Wikipedia infoboxes were also provided. The systems were evaluated on a mix of documents from news and posts to blogs, newsgroups, and discussion fora.

The best system for the 2012 and 2013 benchmarks (and the 2014 variant that provided the perfect mention as an input) is the MS_MLI system by Cucerzan [2012]. It achieved a variant of the F-measure of 70%.[4] These systems are adoptions of the approach described in [Cucerzan, 2007]. The best system for the 2014 benchmark that also performs entity discovery is the system by Monahan et al., 2014.

ERD: The *Entity Recognition and Disambiguation Challenge (ERD)* was co-located with SIGIR 2014. An overview is given in [Carmel et al., 2014]. As the name of the challenge says, the benchmark comprises both

[3] Other tracks of TAC are discussed in subsection 4.3.5 of the Section on *Structured Data Extraction from Text*.

[4] The variant is referred to as b-cubed+ in the TAC benchmarks. It groups together all co-references in the same document into one cluster, and applies the F-measure to these clusters, not to the individual references. This avoids giving undue weight to frequent references in a document (which, by the way typical algorithms work, will either all be correct or all be wrong).

NER and NED, whereas TAC, before 2014, just asked for NED. Also different from TAC, it was not required to recognize and disambiguate entities that were not in the knowledge base.

The knowledge base for ERD was a dump of Freebase (see Table 2.2) from September 2009, restricted to entities with a corresponding page in the English Wikipedia at that same time. There were two tracks, with a different text collection each. For the long track, parts of the ClueWeb09 and ClueWeb12 collections (see Table 2.1) were used. For the short track, web search queries from various past TREC tracks were used. For both tracks, small test sets were provided for learning.

The best system in the long track was again the MS_MLI NEMO system [Cucerzan, 2014], with an F-measure of 76%. That paper also discusses (in its Section 1) the difference to the TAC benchmarks. The best system in the short track was SMAPH by Cornolti et al. [2014], with an F-measure of 69%.

3.2.3 Scale

The first large-scale NER+NED was performed by the SemTag project from Dill et al. [2003]. They recognized and disambiguated 434 million entity occurrences in 264 million web documents. Precision was estimated (from a sample) to be 82%. A relatively small knowledge base (TAP) was used, which explains the small number of recognized occurrences per document and the relatively high precision.

The largest-scale NER+NED at the time of this writing was performed by Google Research [Orr et al., 2013]. They recognized and disambiguated 11 billion entities on 800 million documents from the ClueWeb09 and ClueWeb12 collections (see Table 2.1). The knowledge base used was Freebase, and the NER+NED was reportedly "optimized for precision over recall". Precision and recall were estimated (from a sample) with 80-85% and 70-85%, respectively.

Note that the corpus from Orr et al. [2013] is only about 3 times larger than the one used in the 10 year older study of Dill et al. [2003]. However, the number of entity occurrences is about 25 times larger. Also note that the web crawl of the Common Crawl project from August 2014 contained around 2.8 billion web pages (200 TB), which is only

about 3 times larger than both of the ClueWeb datasets together. In 2015, Google revealed that it knows over 30 thousand trillion different URLs on the Web. However, only a fraction of these point to textual content that is actually useful for search. Also, many URLs point to similar content. As of 2015, the number of distinct web pages indexed by Google is estimated to be around 50 billion.[5]

The bottom line is that web-scale NER+NED with a large general-purpose knowledge base is feasible with good (but still far from perfect) precision and recall.

3.3 Sentence Parsing

The goal of sentence parsing is to identify the grammatical structure of a sentence. There are two kinds of representations that are widely used: the *constituent parse* and the *dependency parse*. Both parses can be viewed as a tree. In a constituent parse, the sentence is first split, e.g., into a *subject noun phrase* (NP) and a *predicate verb phrase* (VP), which are then recursively split into smaller components until the level of words or chunks is reached.

Definition 3.5. A constituent parse of a sentence consists of a tree with the individual words as its leaves. Labels of internal nodes represent grammatical categories of the word sequences corresponding to their subtree, e.g., *noun phrase* (NP), *verb phrase* (VP), *subordinate clause* (SBAR), or *independent clause* (S). These can be nested recursively. The root of the tree is usually labeled *S*.

In a dependency parse, each word in the sentence depends on exactly one other word in the sentence, its head; the root of the tree points to the main verb of the sentence.

Definition 3.6. A dependency parse of a sentence is a tree[6] with individual words as nodes. Nodes are connected via directed edges from the

[5] Source: `http://www.worldwidewebsize.com`, who extrapolate from the number of hits for a selection of queries.

[6] This is often also referred to as a dependency *graph*, but for all practical purposes it can be considered a tree.

governor (*head*) to the *dependent*. Each node has exactly one governor. Edges can be labeled with the grammatical relationship between the words. The main verb of the sentence has an artificial $ROOT$ node as its head.

For example, the sentence from above has the following constituent and dependency parse, respectively. For the constituent parse, the tree structure is shown via nested parentheses labeled with constituent type. The dependency parse only shows unlabeled arcs and no $ROOT$ node.

S(NP((Semantic) (search)) VP(VB(is) ADJP((just) (great)))).

Semantic $\leftarrow$ *search* $\leftarrow$ *is* $\rightarrow$ *great* $\rightarrow$ *just* .

From a linguistic perspective, the two types of grammars are almost equivalent [Gaifman, 1965]. Indeed, many widely used parsers (including the Stanford parser referred to below) produce one type of parse from which they can easily derive the other type of parse.

3.3.1 Semantic Role Labeling

Although this section is about sentence parsing, let us also briefly discuss *Semantic Role Labeling (SRL)* at this point. SRL goes beyond sentence parsing in that it also assigns "roles" to the semantic arguments of a predicate or verb of a sentence. For example, consider the following two sentences:

John gives the book to Mary.

John is given the book by Mary.

Both have the same (structure of the) constituent parse tree. But the role of John with respect to the verb *give* is different: in the first sentence, he is the one who gives (his role is *giver*), in the second sentence, he is the one who is given (his role is *recipient*).

Semantic Role Labeling looks very relevant for semantic search. For example, for the query *who was given the book*, the correct answer is different for each of the two sentences above. However, most semantic search systems nowadays work with (surprisingly) shallow linguistic technology. Many do not even use sentence parsing, and none of the systems described in this survey uses SRL.

There are two apparent reasons. One reason is that semantic search is still in its infancy, with major challenges still to overcome even for relatively basic search scenarios that do not involve any natural language processing or only a relatively shallow one. The other reason is that natural language processing has not yet reached the necessary level of sophistication in order to be unreservedly useful for improving search results. For a look further ahead see Section 6 on *The Future of Semantic Search*.

3.3.2 Benchmarks and State of the Art

Datasets: A benchmark for sentence parsing (that is, a text annotated by the correct parse for each sentence) is referred to as a *Treebank*. The most widely used Treebank is the Penn Treebank [Marcus, Santorini, and Marcinkiewicz, 1993]. It comprises 2,499 articles from the Wall Street Journal from 1989 (about 44K sentences with about 1M words), annotated for syntactic structure by human experts. The articles are split into 25 sections with the convention to use sections 2-21 as a training set and section 23 as test set. Remaining sections can be used as development set. The Penn Treebank also contains the POS-tagged Brown corpus (carefully selected English text from 15 genres, about 1M words, from 1961). There are also two follow-up benchmarks, called Penn Treebank-2 [1995] and Penn Treebank-3 [1999].

More recent English Treebanks are *Ontonotes 5.0* by Hovy et al. [2006], which also contains articles from the Wall Street Journal, and the *Google Web Treebank* by Petrov and McDonald [2012], which consists of annotated sentences from the Web. However, most English parsers are still trained and evaluated on the Penn Treebank.

CoNLL 2007 Shared Task: This task [Nivre et al., 2007] featured two tracks on dependency parsing: one *multilingual* (with an English sub-track), and one called *domain adaptation* (learn from one domain, test on another). Two standard metrics emerged from that task: the *Labeled Attachment Score (LAS)* and the *Unlabeled Attachment Score (UAS)*. Both are percentages, with 100% being a perfect result. UAS measures the degree to which the structure is correct (head and arcs). LAS also takes into account the correctness of the dependency labels.

The best system in the English part of the multilingual track achieved 89.6% LAS and 90.6% UAS. The best system in the domain adaptation track achieved 81% LAS and 83.4% UAS.

SANCL 2012 Shared Task: SANCL [Petrov and McDonald, 2012] is the name of the Workshop on Syntactic Analysis of non-canonical Language. Parsers were trained on the Penn Treebank (sections 2-21, about 30K sentences), but evaluated on the Google Web Treebank. Both dependency and constituent parsers were evaluated. The Stanford parser achieved 80.7% precision and 79.6% recall in constituent mode, and 83.1% LAS and 87.2% UAS in dependency mode. The best constituent parser achieved 84.3% precision and 82.8% recall.

Socher et al. [2013] describe recent improvements to the Stanford parser, and compare it to other state-of-the-art parsers. F-measures between 85% and 92% are achieved.

It is noteworthy that good parsers achieve about equal figures for precision and recall, which is why often only the F-measure is reported. Already the early Charniak parser [Charniak, 2000] achieved both precision and recall of about 90% (on a relatively simple benchmark).

Note that with close to perfect chunking alone (which is a relatively simple task) one already gets around 50% recall (all the minimal constituents are correct) and close to perfect precision, that is, around 75% F-measure. But such a "parsing" would be quite useless for semantic search or information extraction, where it is important to detect which items of a sentence "belong together" semantically. Bastings and Sima'an [2014] therefore introduce an alternative measure, called FREVAL. This measure weighs each component by its height in the parse tree (a leaf has height 1, the root has the largest height). Mistakes in the larger components then incur a higher penalty in the score. Using this measure, they report only 35% to 55% F-measure for current state-of-the-art parsers. Indeed, these figures better reflect our own experience of the limited use of sentence parsing for semantic search, than the close to 90% achieved in the standard measures.

A further problem is that parsers perform worse on long sentences than on short sentences. For example, Klein and Manning [2002] report a drop in F-measure from about 90% for sentences with 10 words to

about 75% for sentences with 40 words. Unfortunately, much of the information in a text is often contained in long (and often complex) sentences. This is exacerbated by the fact that available parser models are usually trained on newswire text but applied to web-like text, which is more colloquial and sometimes ungrammatical (see the results of SANCL 2012 shared task above).

3.3.3 Performance and Scale

In terms of F-measure, the Charniak constituent parser achieves the state-of-the-art result at 92% and claims about 1 second per sentence [McClosky, Charniak, and Johnson, 2006; Charniak, 2000]. The recursive neural network (constituent) parser from Socher et al. [2013] needs about 0.8 seconds per sentence, and achieves 90% F-measure on the Penn Treebank. Recently, parsers have improved parsing times considerably while maintaining or improving state-of-the-art quality. The greedily implemented shift-reduce based constituent parser that is part of the Stanford CoreNLP toolkit [Manning et al., 2014] achieves comparable 88.6% F-measure but is about 30 times as fast (27 ms per sentence). A recent neural network based dependency parser [Chen and Manning, 2014] can process about 650 sentences per second (1.5 ms per sentence) and produce state-of-the-art results (89.6% LAS and 91.8% UAS on the Penn Treebank with Stanford dependencies). spaCy[7] is the currently fastest greedy shift-reduce based parser, which can process about 13K sentences per second (0.08 ms per sentence) with state-of-the-art performance (91.8% UAS on the Penn Treebank). A recent comparison of parsers is given by Choi, Tetreault, and Stent [2015].

3.4 Word Vectors

Word vectors or *word embeddings* represent each word as a real-valued vector, typically in a space of dimension much lower than the size of the vocabulary. The main goal is that semantically similar words should have similar vectors (e.g., with respect to cosine similarity). Also, word vectors often have linear properties, since they are usually obtained by

[7] `https://spacy.io/`

(implicit or explicit) matrix factorization. For example, the methods described below will produce similar vectors for *queen* and *king*, because they are both monarchs, as well as similar vectors for *queen - woman + man* and *king*. Word vectors are also popular as a robust representation of words used as input to machine learning algorithms.

Research on word vectors has a long history in natural language processing dating back to a famous statement by John Rupert Firth in 1957: *You shall know a word by the company it keeps.* Indeed, words that occur in similar contexts are likely to be similar in meaning. This implies that word vectors can be learned in an unsupervised fashion from a huge text corpus, without additional knowledge input. In the following, we discuss the main techniques, extensions to text passages, and the most popular benchmarks.

Applying word vectors has recently gained interest. In this survey many of the approaches in Section 4.8 on *Question Answering on Knowledge Bases* use it as part of the input to machine learning algorithms in order to provide a notion of (semantic) synonyms. It is also part of the future work planned for many recent systems, for example, for Joshi, Sawant, and Chakrabarti [2014] in Section 4.9 on *Question Answering on Combined Data.*

3.4.1 Main Techniques

The straightforward approach is to build a word-word co-occurrence matrix [Lund and Burgess, 1996], where each entry counts how often the two words co-occur in a pre-defined context (in the simplest case: within a certain distance of each other). A row (or column) of the matrix can then be considered as a (huge but sparse) word vector. From there, a low-dimensional dense embedding can be obtained via matrix factorization techniques. For example, using principal component analysis (PCA, typically computed from an eigenvector decomposition) or non-negative matrix factorization (NMF, typically computed with a variant of the EM algorithm) [Lee and Seung, 2000]. There are many variations of this basic approach; for example, the co-occurrence matrix can be row-normalized, column-normalized, or each entry can be replaced by its positive pointwise mutual information.

Explicit semantic analysis (ESA) [Gabrilovich and Markovitch, 2007] represents a word as a vector of weighted Wikipedia concepts. The weight of a concept for a word is the tf-idf score of the word in the concept's Wikipedia article. The resulting word vectors are often sparse, because each concept article contains only a small subset of all possible words. By construction, longer text passages can be represented by the sum of the word vectors of the contained words. The resulting vector is then supposed to be a good representation of what the text "is about". Like PCA and NMF, ESA can be combined with standard ranking techniques (like BM25) to improve retrieval quality in keyword search on text.

Word2vec [Mikolov et al., 2013a; Mikolov et al., 2013b] computes (dense) word vectors using a neural network with a single hidden layer. The basic idea is to use the neural network for the following task: given a current word w_i, predict the words w_{i+c} occurring in its context (a window around w_i, e.g., positions $-2, -1, +1, +2$). The network is trained on an arbitrary given text corpus, with the goal of maximizing the product of these probabilities. Once trained, the word vectors can be derived from the weights of the intermediate layer. Interestingly, Levy and Goldberg [2014] could show that word2vec implicitly performs a matrix factorization of the word-context matrix. The major advantage over the explicit matrix factorization techniques from above is in space consumption and training speed; see the next but one paragraph.

Glove [Pennington, Socher, and Manning, 2014] is a log-bilinear regression model that, intuitively, is trained to predict word co-occurrence counts. The model effectively performs a factorization of the log co-occurrence count matrix [Levy, Goldberg, and Dagan, 2015]. Experiments show that it performs similarly to word2vec; see the next paragraph. It is also fast to train, but requires a co-occurrence count matrix as input.

Levy, Goldberg, and Dagan [2015] perform an extensive comparison of the many approaches for word vectors, including word2vec, Glove and co-occurrence based methods. They also give valuable advice on how to choose hyperparameters for each model. In their experiments, none of the techniques consistently outperforms others. Experiments

also show that hyperparameters have a huge impact on the performance of each model, which makes a direct comparison difficult. However, they report significant performance differences on a corpus with 1.5 billion tokens, in particular: half a day versus many days for the training phases of word2vec vs. Glove, and an unfeasibly large memory consumption for the explicit factorization methods.

3.4.2 Extensions to Text Passages

The straightforward approach to obtain a low-dimensional vector of the kind above for an arbitrary text passage is to sum up or average over the vectors of the contained words. This works well in some applications, but can only be an approximation because it completely ignores word order.

Le and Mikolov [2014] have extended word2vec to compute vectors for paragraphs and documents. The vector is learned for the given passage as a whole, and not just statically composed from individual word vectors. On a sentiment analysis task, the approach beats simple composition methods (as described in the previous paragraph) as well as classical supervised methods (which do not leverage external text).

Two related problems are *paraphrasing* and *textual entailment*, where the task is to determine for two given pieces of text, whether they mean the same thing or whether the first entails the second, respectively. For example, does *John go to school every day* entail *John is a student*? Learning-based methods for paraphrasing and textual entailment are discussed in Section 8.1 of the survey by Li and Xu [2014].

3.4.3 Benchmarks

There are two popular problems that allow an intrinsic evaluation of word vectors. For each problem, several benchmark datasets are available.

Word similarity: This is a ranking task. Word pairs must be ranked by how similar the two words are. For example, the words of the pair (*error*, *mistake*) are more similar than (*adventure*, *flood*). Benchmarks contain word pairs with human-judged graded similarity scores, often

retrieved via crowdsourcing. The final quality is assessed by computing rank correlation using Spearman's ρ. Some relevant benchmarks are:

WordSim353 [Finkelstein et al., 2002]: 353 word pairs
SimLex-999 [Hill, Reichart, and Korhonen, 2015]: 999 word pairs
MEN [Bruni, Tran, and Baroni, 2014]: 3000 word pairs
Rare words [Luong, Socher, and Manning, 2013]: 2034 word pairs
Crowdsourcing benchmark [Radinsky et al., 2011]: 287 word pairs

For a recent performance comparison we refer to the experiments done by Levy, Goldberg, and Dagan [2015]. Typical rank correlations are between .4 and .8, depending on the dataset and the model.

Word analogy: Analogy questions are of the form "*saw* is to *sees* as *returned* to *?*" and the task is to fill in the missing word (*returns*). More formally, the task is: given words a, a^* and b find the word b^* such that the statement "a is to a^* as b is to b^*" holds. One variant of this task addresses syntactic similarities, as in the example above. The other variant focuses on semantic similarities as in "*Paris* is to *France* as *Tokyo* is to *?*" (*Japan*). To solve this task, simple vector arithmetic is used. Most prominently:

$$\arg \max_{b^* \in V \setminus \{a, a^*, b\}} \cos(b^*, a^* - a + b)$$

where V is the vocabulary. Levy, Goldberg, and Dagan [2015] improved on that function, essentially by taking the logarithm. The resulting function is called *3CosMul*:

$$\arg \max_{b^* \in V \setminus \{a, a^*, b\}} \frac{\cos(b^*, a^*) \cdot \cos(b^*, b)}{\cos(b^*, a) + \epsilon}$$

Notable benchmarks are from Microsoft Research [Mikolov, Yih, and Zweig, 2013], which consists of 8,000 questions focusing on syntactic similarities, and Google [Mikolov et al., 2013b], which consists of 19,544 questions for syntactic as well as semantic similarities. The evaluation measure is the percentage of words b^* that were correctly predicted. Again, we refer to the experiments from Levy, Goldberg, and Dagan [2015] for a recent comparison. Current models answer about 55% to 69% of these questions correctly.

4

Approaches and Systems for Semantic Search

This is the core section of this survey. Here we describe the multitude of approaches to and systems for semantic search on text and knowledge bases. We follow the classification by data type and search paradigm from Section 2, depicted in Figure 1.1. For each of the nine resulting subgroups, there is a subsection in the following. Each of these subsections has the same structure:

Profile ... a short characterization of this line of research
Techniques ... what are the basic techniques used
Systems ... a concise description of milestone systems or software
Benchmarks ... existing benchmarks and the best results on them

We roughly ordered the sections historically, that is, those scenarios come first, which have been historically researched first (and the most). Later sections correspond to more and more complex scenarios, with the last one (Section 4.9 on *Question Answering on Combined Data*) being the hardest, with still relatively little research to date. Also, approaches and systems of the later section often build on research from the more fundamental scenarios from the earlier sections. For example, almost any approach that deals with textual data uses standard data structures and techniques from classical keyword search on text.

4.1 Keyword Search in Text

Data	Text documents, as described in Section 2.1.1
Search	Keyword search, as described in Section 2.2.1 This is classical full-text search: the query is a sequence of (typically few) keywords, and the result is a list of (excerpts from) documents relevant to the query *Methods aimed at a particular entity or list of entities are addressed in Section 4.5*
Approach	Find all documents that match the words from the query or variants/expansions of the query; rank the results by a combination of relevance signals (like prominent occurrences of the query words in the document or occurrences in proximity); learn the optimal combination of these relevance signals from past relevance data
Strength	Easy to use; works well when document relevance correlates well with basic relevance signals
Limitation	Is bound to fail for queries which require a match based on a deeper understanding (of the query or the matching document or both), or which requires the combination of information from different sources

This is the kind of search we are all most familiar with from the large web search engines: you type a few keywords and you get a list of documents that match the keywords from your query, or variations of them.

A comprehensive treatment of this line of research would be a survey on its own. We instead provide a very brief overview of the most important aspects (Section 4.1.1), widely used software which implements the state of the art (Section 4.1.2), and an overview over the

most important benchmarks in the field and a critical discussion of the (lack of major) quality improvements over the last two decades (Section 4.1.3).

4.1.1 Basic Techniques

With respect to search quality, there are two main aspects: the *matching* between a keyword query and a document, and the *ranking* of the (typically very many) matching documents. We do not cover performance issues for keyword search on text in this survey. However, Section 5.2 discusses various extensions of the inverted index (the standard indexing data structure for keyword search on text) to more semantic approaches.

Basic techniques in matching are: lemmatization or stemming (*houses* → *house* or *hous*), synonyms (*search* ↔ *retrieval*), error correction (*algoritm* ↔ *algorithm*), relevance feedback (given some relevant documents, enhance the query to find more relevant documents), proximity (of some or all of the query words) and concept models (matching the *topic* of a document, instead of or in addition to its words). A recent survey on these techniques, cast into a common framework called *learning to match*, is provided by Li and Xu [2014].

Basic techniques in ranking are either query-dependent ranking functions, like BM25 (yielding a score for each occurrence of a word in a document) and language models (a word distribution per document), or query-independent popularity scores, like PageRank (yielding a single score per document). Hundreds of refinements and signals have been explored, with limited success; see the critical discussion in the benchmark section below. The most significant advancement of the last decade was the advent of *learning to rank (LTR)*: this enables leverage of a large number of potentially useful signals by learning the weights of an optimal combination from past relevance data. See [Liu, 2009] for a survey with a focus on applicable machine learning techniques. We discuss applications of LTR to other forms of semantic search in Section 5.1 on *Ranking*.

4.1.2 Popular State-Of-The-Art Software

Researchers have developed countless systems for keyword search on text. A list is beyond the scope of this article, and bound to be very incomplete anyway. Instead, we focus on open-source software and prototypes that are widely used by the research community. Each of the systems below provides basic functionality like: incremental index updates (adding new documents without having to rebuild the whole index), fielded indices (to store arbitrary additional information along with each document), distributed processing (split large text collections into multiple parts, which are then indexed and queried in parallel), standard query operators (like: conjunction, disjunction, proximity), and multi-threading (processing several queries concurrently). Also, each of the systems below is used as basis for at least one system for more complex semantic search, which are described in one of the following sections.

There are several studies comparing theses systems. For a quality comparison of some of these, see [Armstrong et al., 2009a]. For a performance comparison, see [Trotman et al., 2012].

Apache's Lucene[1] is the most widely used open-source software for basic keyword search. It is written in Java and designed to be highly scalable and highly extensible. It is the most used software in commercial applications. Lucene provides built-in support for some of the basic matching and ranking techniques described in Section 4.1.1 above: stemming, synonyms, error correction, proximity, BM25, language models.

Indri[2] is written in C++. It is a general-purpose search engine, but particularly used for language-model retrieval. Terrier[3] is written in Java, and provides similar functionality as Indri.

MG4J [Boldi and Vigna, 2005] is written in Java. It makes use of *quasi-succinct* indexes, which are particularly space-efficient and enable particularly fast query processing also for complex query operators. MG4J supports fielded BM25, which is used by various of the

[1] `http://lucene.apache.org`
[2] `http://www.lemurproject.org/indri`
[3] `http://terrier.org`

approaches described in Section 4.5 on *Keyword Search on Combined Data*.

4.1.3 Benchmarks

The classical source for benchmarks for keyword search on unstructured text is the annual Text Retrieval Conference (TREC) series [Voorhees and Harman, 2005], which began in 1992.[4] TREC is divided into various so-called *tracks*, where each track is about a particular kind of retrieval task. Each track usually runs over a period of several years, with a different benchmark each year. Each benchmark consists of a document collection, a set of queries, and relevance judgments for each query.[5]

Keyword search on text documents was considered in the following tracks: *Ad-hoc* (1992 - 1999, keyword search on the TIPSTER[6] collection), *Robust* (2003 - 2005, hard queries from the ad-hoc track), *Terabyte* (2004 - 2006, much larger document collection than in previous tracks), and *Web* (1999 - 2004 and 2009 - 2014, web documents).

Armstrong et al. [2009a] and Armstrong et al. [2009b] conducted an extensive comparative study of the progress of ad-hoc search over the years. Systems were compared in two ways: (1) by direct comparison of different results from different papers on the same (TREC ad-hoc) benchmarks, and (2) by a comparison across benchmarks using a technique called *score standardization*. Their surprising conclusion from both studies is that results for ad-hoc search have not improved significantly since 1998 or even earlier. New techniques were indeed introduced, but the evaluations were almost always against weak baselines, instead of against the best previous state-of-the-art system.

Viewed from a different perspective, this study merely confirms a typical experience of information retrieval researchers regarding keyword search. The shortcomings are clear, and promising new ideas

[4] When researching proceedings, it helps to know that the first 9 TREC conferences, from 1992 to 2000, are referenced by number: TREC-1, ..., TREC-9. Starting from 2001, they are referenced by year: TREC-2001, TREC 2002, ...

[5] For the later (very large) collections, only partial relevance judgments (for the top documents from each participating system) were available. This is called *pooling*.

[6] The TIPSTER collection comprises news articles, government announcements, and technical abstracts.

spring to mind relatively quickly. But a comprehensive and honest evaluation of any single idea over a large variety of queries is often sobering: the results for some queries indeed improve (usually because relevant documents are found, which were not found before), while the results for other queries deteriorate (usually because of lower precision). Often, the two opposing effects more or less balance out, and it is mostly a matter of careful parameter tuning to get a slight improvement out of this.

A real improvement was brought along by the learning to rank approach, discussed briefly in Section 4.1.1 above. With learning to rank, a large number of potentially useful signals can be combined, and the best "parameter tuning" can be learned automatically from past relevance data. Indeed, the winners of the last three TREC Web Tracks are all based on this approach.

In absolute terms, results remained relatively weak however, with typical nDCG@20 values of around 30%. This makes it all the more reasonable to go beyond this simple form of keyword search and aim at deeper forms of understanding, which is exactly what the approaches described in the following sections do.

4.2 Structured Search in Knowledge Bases

Data	A knowledge base, as described in Section 2.1.2
Search	Structured search, as described in Section 2.2.2 The query is from a language like SPARQL; the result is a list of matching items from the knowledge base; the order is arbitrary or explicitly specified
Approach	Store the knowledge base in a standard RDBMS and rewrite queries to SQL; or use a dedicated index data structure and query engine
Strength	Expert searches with a precise semantics; the canonical back end for any service that involves non-trivial queries to a knowledge base
Limitation	Query formulation is cumbersome, especially for complex queries; finding the right entity and relation names becomes very hard on large knowledge bases; the amount of information contained in knowledge bases is small compared to the amount of knowledge contained in text

Structured search in knowledge bases is not so much a technique for semantic search on its own, but rather a basic building block for all approaches that work with one or more knowledge bases.

4.2.1 Basic Techniques

There are two main approaches to storing a knowledge base: in a standard relational database management system (RDBMS), or in a system dedicated to storing knowledge as collections of triples and hence often called *triple store*. Both approaches are widely used. The design and implementation of a typical triple store is described in Section 4.2.2 below.

When the knowledge base is stored in an RDBMS and the query language is SPARQL, queries can be translated to equivalent SQL queries.

A complete translation scheme is described in [Elliott et al., 2009].[7] When the data is stored in an RDBMS using a non-trivial schema (that is, not just one big table of triples), a mapping is needed to specify how to make triples out of this data. For this mapping, R2RML [2012] has emerged as a standard. Given such a mapping, generating a SQL query that can be executed as efficiently as possible becomes a non-trivial problem [Unbehauen, Stadler, and Auer, 2013].

Traditional RDBMSs store their data row oriented, that is, the items from one row are contiguous in memory. This is advantageous when retrieving complete rows via direct access (e.g., via their key). When storing a knowledge base in an RDBMS, column orientation is the layout of choice. This is because typical SPARQL queries require scans of very long runs of entries for one attribute. For example, to find all people born in a given city, we need to determine all triples with that city as their object. Also, these columns are typically highly compressible. For the example just given, there will be long runs of triples with the same city (if sorted by object). A simple run-length encoding then saves both space and query time. A recent survey on column-oriented databases aka column stores (with focus on efficiency) is provided by Abadi et al. [2013].

The list of systems and benchmarks in Sections 4.2.2 and 4.2.3 below focuses on systems that explicitly support SPARQL. There are two main aspects when comparing these systems: their performance and which features they support.

Performance

It appears that dedicated triples stores have an advantage over RDBMS-based systems. Dedicated triple stores can use index data structures that are tailored to sets of triples (in particular, exploiting the high repetitiveness and hence compressibility involved, see above). Similarly, they can use query optimizers that exploit the structure of

[7] The SPARQL features ASK, CONSTRUCT, and DESCRIBE are treated specially, since they can only be approximated in SQL. They are not essential for the expressiveness of SPARQL, however.

typical SPARQL queries.[8] It turns out, however, that RDBMS-based approaches can still be superior, especially for complex queries, because of their more mature query-optimizer implementations. This is briefly discussed in the benchmark subsection below. Query planning and optimization are a research topic of their own, and we refer the interested reader to [Schmidt, Meier, and Lausen, 2010].

Features

All the widely used systems below support the full SPARQL standard. Research prototypes often focus on SELECT queries, details below. Other features, which some but not all of the systems provide, are:

Reasoning: support for reasoning, e.g., using OWL or RDFS; this is the topic of Section 5.4 on *Inference*.

Web API: query or modify the database via HTTP.

Exchangeable back end: plug in different back ends; in particular, allow the choice between a dedicated triple store and an RDBMS.

Full-text search: support for keyword search in objects which are string literals; here is an example using the syntax from Virtuoso's keyword search extension (the prefix *bif* stands for built-in function and prefixes for the other relations are omitted):

```
SELECT ?p WHERE {
  ?p has-profession Astronaut .
  ?p has-description ?d .
  ?d bif:contains "walked AND moon" }
```

Note that already standard SPARQL enables regular-expression matching of entity names via the *FILTER regex(...)* operation. In principle, regular expressions can simulate keyword queries, but not very practically so. For example, a string literal matches the two keywords *w1* and *w2* if it matches one of the regular expressions *w1.*w2* or *w2.*w1* .

[8] From a more general perspective, such special-purpose databases are often called *NoSQL* (acronym for "non (relational) SQL", sometimes also interpreted as "not only SQL"). Another example of a NoSQL database is Google's *BigTable*, which supports database-like queries on extremely large amounts of data that may be stored distributed over thousands of machines.

Note that entity names are also string literals. This simple kind of search is hence also useful when the exact name of the entity is not known, or for long names. For example, the entity name *Barack Hussein Obama* would be found with the keyword query "barack AND obama".

4.2.2 Systems

The three most widely used systems at the time of this writing are (in chronological order of the year the system was introduced): Virtuoso[9], Jena[10], and Sesame [Broekstra, Kampman, and Harmelen, 2002].

All three provide all of the features listed above. Virtuoso is written in C, Jena and Sesame are written in Java. Virtuoso is different in that it is also a full-featured RDBMS; in particular, it can run with its own RDBMS as back end. For a performance comparison, see the benchmarks below.

Traditional database companies, like Oracle or MySQL, have also started to provide support for triple stores and SPARQL queries. However, at the time of this writing, they still lack the breadth of features of systems like Virtuoso, Jena, or Sesame.

Details of the implementation of a dedicated triple store and SPARQL engine are described in [Neumann and Weikum, 2009; Neumann and Weikum, 2010], for a system call RDF-3X. The software is open source. RDF-3X supports SELECT queries with the most important modifiers and patterns.[11] RDF-3X builds an index for each of the six possible permutations of a triple (SPO, SOP, OPS, OSP, POS, PSO, where S = subject, P = predicate, O = object). This enables fast retrieval of the matching subset for each part of a SPARQL query. Join orders are optimized for typical SPARQL queries, including star-shaped (all triples have the same variable as their subject) and paths (the object of one triple is the subject of the next). Query plans are ranked using standard database techniques, like estimating the cost

[9] `http://virtuoso.openlinksw.com`

[10] `https://jena.apache.org`

[11] Supported patterns: OPTIONAL and FILTER. Supported modifiers: ORDER BY, DISTINCT, REDUCED, LIMIT, and OFFSET. Not supported: ASK, DESCRIBE, and CONSTRUCT queries.

via histogram counts. The authors provide a performance evaluation, where RDF-3X is faster than two column-store RDBMs (MonetDB and PostgreSQL) on a variety of datasets (including BTC'09 and UniProt from Tables 2.3 and 2.2). This is inconsistent with the results from the Berlin SPARQL benchmark, discussed below, where an RDBMs (Virtuoso) wins when the data is very large.

Bast et al. [2014a] provide a system for the incremental construction of tree-like SPARQL queries. The system provides context-sensitive suggestions for entity and relation names after each keystroke. The suggestions are ranked such that the most promising suggestions appear first; this ranking is discussed in more detail in Section 5.1.4. As of this writing, an online demo for Freebase (see Table 2.2) is available: `http://freebase-easy.cs.uni-freiburg.de`. The demo also addresses the challenge of providing unique and human-readable entity names.[12]

SIREn [Delbru, Campinas, and Tummarello, 2012] uses an inverted index (Lucene) to support star-shaped SPARQL queries (with one entity at the center), where predicate and relation names can be matched via keyword queries. We describe the index in more detail in Section 5.2.1 on *Using an Inverted Index for Knowledge Base Data*.

4.2.3 Benchmarks

The Berlin SPARQL Benchmark [Bizer and Schultz, 2009] is modeled after a real use case: a consumer looking for a product on an e-commerce website. 12 generic queries are chosen to model the SPARQL queries sent to the back end during such a session. The queries are parameterized, e.g., by the type of product that the consumer is looking for initially. The benchmark demands that the queries are asked in sequence, with multiple sequences being asked concurrently, again as in a real setting. The dataset is modeled after a real set of products (with various features and textual descriptions) and is synthetically generated, with an arbitrary, given size.

[12] The entity names from Freebase are not unique, and the identifiers are alphanumeric strings. In contrast, for example, Wikipedia has human-readable unique identifiers for each of its entities.

Bizer and Schultz [2009] compare a large variety of systems: explicit triple stores (including: Jena, Sesame, and Virtuoso with its own triple store back end), and SPARQL-to-SQL rewriters using an RDBMS (including: MySQL and Virtuoso with its own RDBMS back end). Dataset sizes used were 1M, 25M, and 100M triples. No single system came out as the clear winner. However, for the largest datasets (100M), the best RDBMS-based approach (Virtuoso) was about 10 times faster on average than the best dedicated triple store. The authors attribute this to the more mature query optimizers of established RDBMS systems. It is noted that the SPARQL-to-SQL rewriting takes up to half the time.

The DBpedia SPARQL Benchmark [Morsey et al., 2011] is a generic benchmark that aims at deriving *realistic* SPARQL queries from an arbitrary given query log for an arbitrary given knowledge base. In particular, 25 query templates are derived for the DBpedia dataset (see Table 2.2). Their evaluation confirms the performance differences from previous benchmarks, notably Bizer and Schultz [2009], except that the performance differences are even larger with realistic data and queries.

4.3 Structured Data Extraction from Text

Data	Text documents, as described in Section 2.1.1 This includes web documents with markup that helps to identify structure in the data
Search	The main purpose of the systems described in this section is to extract structured information from text; the search is then an add-on or left to systems as discussed in Section 4.2 on *Structured Search in Knowledge Bases*
Approach	Extract structured data from text; store in a knowledge base or reconcile with an existing one; an alternative for very simply structured queries is to translate them to suitable keyword queries
Strength	Make the vast amounts of structured data contained in text documents accessible for structured search
Limitation	Extraction with high precision and recall is hard; reconciling extracted information in a single knowledge base is hard; some information is hard to express in structured form

A large part of the world's information is provided in the form of natural language text, created for humans. Large amounts of this information could be naturally stored (and queried) in structured form.

4.3.1 Basic Techniques

We distinguish three kinds of approaches to access structured data contained in text documents: relationship extraction from natural language text, extraction of tables or infoboxes, and knowledge base construction. In a sense, the approaches build on each other, which is why we describe them in this order.

For each of the three approaches, we describe the state-of-the-art systems and performance in Sections 4.3.2 - 4.3.4. For a holistic overview of the whole field of information extraction from text we refer to the excellent survey from Sarawagi [2008].

Relationship Extraction from Natural Language Text

Relationship extraction aims at extracting subject-predicate-object tuples from a given collection of natural language text. Consider the following sentence from Wikipedia:

Aldrin was born January 20, 1930, in Mountainside Hospital, which straddles both Glen Ridge and Montclair

In basic relationship extraction, the searched relation is part of the input. For example, extract all triples for the *place of birth* relation from a given text. For the sentence above such a triple would be:

Buzz Aldrin place of birth Glen Ridge

The subject and object may or may not be linked (one also says: *grounded*) to a matching entity from a given knowledge base (in the example they are: we use the names from an – imaginary in this case – knowledge base, not from the original sentence). Since the relation was given, the predicate is easily grounded. Depending on the verb, there may be multiple objects (in the example, there is just one).

Banko et al. [2007] introduced *Open Information Extraction* (OIE), where the goal is to extract as many tuples as possible (for any relation) from the given text. For the example sentence above, a typical OIE system would extract:

Aldrin was born Glen Ridge

Unlike for the triple above, the subject and, especially, the predicate are not grounded, but are simply expressed using words from the sentence.

Specialized Extraction

Web documents often contain additional structure in the form of markup for some of the contents. Two notable such sub-structures

are *tables* and Wikipedia *infoboxes*. Tables are interesting because a lot of structured information contained in text is formatted as tables. Balakrishnan et al. [2015] report that they have indexed over a hundred million HTML tables that contain interesting structured data.[13] Infoboxes are interesting because Wikipedia covers a lot of general-purpose knowledge with high quality. In Section 4.3.3 below, we discuss several systems developed for these sub-structures.

There is also vast literature on domain-specific extraction, in particular, for the life sciences. For example, extract all pairs of proteins (subject and object) that interact in a certain way (predicate) from a large collection of pertinent publications. The main challenge for such systems is domain-specific knowledge (e.g., the many variants how protein names are expressed in text), which is beyond the scope of this survey.

Knowledge Base Construction

Basic extraction processes, as described in the previous two subsections, yield a (typically very large) collection of elements of structured data, often triples. To obtain a knowledge base, as described in Section 2.1.2, two challenging steps are still missing: entity resolution and knowledge fusion, which we briefly explain here.

For *entity resolution*, sometimes also called entity de-duplication, strings referring to the same entity must be mapped to a unique identifier for that entity. For example, the extraction process might yield the two triples:

Buzz Aldrin	*born in*	*Glen Ridge*
Aldrin	*born in*	*Montclair*

Here, the two subjects are different strings but refer to the same entity. Depending on the extraction process, this might also happen for the predicates.

For *knowledge fusion*, different triples might contain conflicting or complementary information, which needs to be resolved or unified. For the two triples above, both provide correct information in a sense (the

[13] Note that HTML tables are often used in web pages merely for formatting.

hospital where Aldrin was born straddles both Glen Ridge and Montclair). A system might also choose to discard one triple (because, in this case, place of birth is a functional relation, that is, for each subject there can be only one "true" object).

An excellent overview of the creation of state-of-the-art knowledge bases is given in the tutorial by Bordes and Gabrilovich [2015].

4.3.2 Systems for Relationship Extraction from Text

Early approaches to relationship extraction make use of hand-crafted rules or patterns. A classical example is the pattern *NP such as NP*, which if matched in a text likely points to a *hyponomy* relation between the two noun phrases [Hearst, 1992].

The next generation of systems was based on supervised classification using linguistic features such as phrase chunks and dependency paths [Zhou et al., 2005; Fundel, Küffner, and Zimmer, 2007] or tree kernels [Zelenko, Aone, and Richardella, 2003]. Zhou et al. [2005] report 55.5% F-measure (63.1% precision and 49.5% recall) over a set of 43 relations on a corpus of the NIST Automatic Content Extraction (ACE) program.[14] Again, we refer to the survey by Sarawagi [2008] for a good overview.

A common problem for supervised approaches is that labeled training data is required for each relation to be extracted. Therefore, recent approaches make use of *distant supervision* [Mintz et al., 2009] to derive (noisy) training data. The idea is to find training examples for each relation, by finding sentences in which entities, that are known to be in the given relation, co-occur. This is possible with large-scale public domain knowledge bases like Freebase, covering many relations and entities, and a large text corpus, where mentioned entities have been identified. Note that distant supervision is a technique that can be applied for other tasks as well. In general, whenever noisy training data can be derived using an authoritative source, one can speak of distant supervision.

Of course, the assumption that co-occurring entities are in the given relation does not always hold. For example, the entities *Neil Armstrong*

[14] `https://www.ldc.upenn.edu/collaborations/past-projects/ace`

and *Wapakoneta* can co-occur in a sentence because it states that Armstrong was born in Wapakoneta or because he took flying lessons there. Hence, recent approaches focus on better learning from this kind of noisy data [Riedel, Yao, and McCallum, 2010; Hoffmann et al., 2011; Surdeanu et al., 2012].

There is no standard annual benchmark for evaluation, and results differ based on the considered relations and used corpus. The much compared-to work by Hoffmann et al. [2011] reports around 60% F-measure (72.4% precision and 51.9% recall) across all extracted relations.

The first approaches to Open Information Extraction (OIE) mainly used patterns over shallow NLP, e.g., part-of-speech tags. Given a set of seed entities that are in a specified relation, TextRunner learns surface patterns from a text corpus and the initial bootstrap set is enriched with additional entities [Yates et al., 2007]. Later systems combined manually crafted rules with classifiers learned via this bootstrapping process, e.g., ReVerb [Fader, Soderland, and Etzioni, 2011; Etzioni et al., 2011].

More recent systems tend to utilize deeper NLP (dependency or constituent parses), e.g., OLLIE [Mausam et al., 2012] learns patterns on dependency parses. The currently best approaches use manually crafted rules over deep NLP, notably ClausIE [Corro and Gemulla, 2013] and CSD-IE [Bast and Haussmann, 2013] (extended by Bast and Haussmann [2014] to make triples more informative). Both systems report around 70% of correct extractions with around twice as many correct extractions as OLLIE [Mausam et al., 2012].

Triples from OIE systems can be used for semantic search in a variety of ways. In [Fader, Zettlemoyer, and Etzioni, 2013], the triples are searched directly, with parts of the query being matched to parts of the triples, making extensive use of paraphrases. This approach only works when the elements from the result sets correspond to individual triples. A demo of this kind of search is provided under `http://openie.allenai.org`. In [Bast et al., 2014b], triples are used to establish semantic context (which entitites and words "belong together") in semi-structured search on combined data; the system is

described in Section 4.6.2. In principle, OIE triples could also be used for knowledge base construction. However, all of the systems described in Section 4.3.1 below work with a fixed set of relations. This takes away the burden of the problem of predicate name resolution (which is hard, see Section 4.8.1). Additionally, the schema of the knowledge base provides a filter on which triples are actually useful. For example, OIE systems also extract triples like

John *cheered for* *his team*

that are usually not desirable to include in a knowledge base.

4.3.3 Systems for Specialized Extraction

WebKB [Craven et al., 1998] was one of the first systems to extract triples from hyperlinked documents, namely the website of a computer science department. In their approach, web pages stand for entities (for example, the homepage of a person stands for that person) and links between web pages indicate relations (for example, a link between a person's homepage and the department homepage is a strong indicator that that person works in that department). The correspondence between web pages and entities is learned in a supervised fashion using a Naive Bayes classifier with standard word features. Relations are also learned using FOIL (rule-based, supervised learning) with link paths (for example, a link from a person to a department) and anchor text (for example, the word *department* in the anchor text) as features. In their evaluation, 450 instances of 6 classes are classified with 73% precision and 291 instances of 3 relations are extracted with 81% precision.

EXALG [Arasu and Garcia-Molina, 2003] is a system for gathering knowledge from websites that fill templates with structured data. The goal is to deduce the template without any human input, and then use the deduced template to extract data. Mapping the extracted data to an existing ontology is not part of the system. Technically, the system works in two stages. In the first stage, it collects tokens that occur (exactly) equally often and thus indicate a template (e.g., a label like *Name:*). In the second stage, the data values are extracted. These are expected to be found between the re-occurring tokens from stage one.

Limaye, Sarawagi, and Chakrabarti [2010] present a system that extracts structured information from tables contained in web documents. In a preprocessing step, for each table, its cells, columns, and column pairs are mapped to entities, types, and relations from the YAGO knowledge base. For example, consider a table with two columns: names of persons and their birth place. The cells are mapped to particular persons and places, respectively, the columns are mapped to the types *person* and *location*, and the column pair is mapped to the relation *born in*. The mappings are learned using features such as: the similarity between the entity name and the text in the cell, the similarity between a relation name and a column header, and whether entity pairs from a labeled relation are already in this relation according to YAGO. WebTables [Cafarella et al., 2008] is a system for finding web tables in the first place. For example, given the keyword query *city population*, find tables on the Web containing information about cities and their population.[15]

4.3.4 Systems for Knowledge Base Construction

YAGO [2007] is a knowledge base originally obtained from Wikipedia's infoboxes and from linking Wikipedia's rich category information to the WordNet [Miller, 1992] taxonomy using basic NLP techniques. For example, the Wikipedia category *German Computer Scientists* can be (easily) linked to the WordNet category *Computer Scientist*, and from the WordNet taxonomy one can then infer that an entity with that category is also a *Scientist* and a *Person*. More recent versions of YAGO also contain statements from matching patterns in text, as well as extensive spatial and temporal information [Hoffart et al., 2013].

DBpedia [Auer et al., 2007] is a community effort to extract structured information from Wikipedia. The most important part are templates that extract structured data from Wikipedia infoboxes. However, there are also other extractors, including some that harvest information from full text using NLP techniques [Lehmann et al., 2015]. For example, there is an extractor that infers the gender of a person from the usage of pronouns in the person's article.

[15] WebTables is used in several Google products; see [Balakrishnan et al., 2015].

The Never-Ending Language Learner (NELL) [Carlson et al., 2010; Mitchell et al., 2015] is a system that constructs a knowledge base from the Web in a staged fashion, where previously learned knowledge enables further learning. NELL has been running 24 hours/day since January 2010, and so far has acquired a knowledge base with over 80 million confidence-weighted statements. It started with a small knowledge base that defines a basic ontology (that is, a set of types and predicates of interest) and a handful of seed examples. In each cycle, the current knowledge base is used to train several components, which are then used to update the knowledge base. These components include: relationship extraction (see Section 4.3.2), removing mutually exclusive statements (see Section 4.3.1), and inference modules that generate new statements (if two people have the same parents, they should also be in a sibling relationship).

Google's Knowledge Vault [Dong et al., 2014] is a web-scale probabilistic knowledge base that combines extractions from web content with knowledge derived from existing repositories. Knowledge Vault contains three major components. First, triple extractors that utilize distant supervision using basic NLP features derived from POS tagging, NER+NED, dependency parsing, and co-reference resolution (see Section 3). Second, graph-based priors that predict possibly missing triples (with a probability) based on what is already stored in the knowledge base. For example, one can infer a missing instance of a *sibling* relation, if two persons have the same *parent*. Missing *parent* triples can also be hinted at by a *sibling* triple, but with less confidence as the other way round. These predictions are made without manually specified rules. The final component is knowledge fusion that computes the probability of a triple being true, based on agreement between different extractors and priors. According to [Bordes and Gabrilovich, 2015], Knowledge Vault contained 302M high-confidence facts in 2015.

DeepDive [Zhang, 2015; Wu et al., 2015] provides the basic building blocks for knowledge base construction systems. An initial knowledge base and a text corpus are required. Users of DeepDive have to provide extractors, training examples, and rules. Extractors can be off-the-shelf tools or tailor-made and extract entity occurrences from the text (as

offsets) and whatever else might be a useful feature: POS tags, dependency parses, etc. Training examples are typically obtained using *distant supervision* (explained in Section 4.3.2), but can also be provided manually. Rules can state something like "if a person smokes, the person is likely to have cancer". DeepDive then learns weights for those rules and performs inference without the developer having to worry about the algorithmic intricacies. Therefore, it creates a probabilistic model and jointly learns: (1) optimal weights for the user-defined rules, and (2) probabilities for candidate triples to be added to the knowledge base. In the example above, smoking makes cancer more probable, but that does not mean every smoker necessarily has cancer. The weight for that rule is learned from existing data and, together with evidence from the text (typical patterns or formulations), determines the confidence of new statements that might be added to the knowledge base.

Angeli et al. [2014] present a system based on DeepDive. This system was the best performing system at the 2014 TAC-KBP slot filling task [Surdeanu and Ji, 2014], which is described below. Distant supervision is performed with Freebase as a source of training data. To improve upon this, a manual feedback round is added to find features that are good indicators of the relation or not.

4.3.5 Benchmarks

For the basic task of relationship extraction, there is no widely agreed-upon benchmark. Rather, each of the systems described in Section 4.3.2 comes with its own benchmark (as briefly summarized above). The likely reason is the many variants of the extraction task: which relations to extract (fixed subset or open), the subjective judgment which triples are actually entailed by the text (and hence counted as correct), whether to extract triples or n-tuples, optimize for precision or for recall, etc.

Since 2009, the TAC conference series has a Knowledge Base Population (KBP) track. Overview papers from 2010 to 2014 are available via the conference's website: [Ji et al., 2010; Ji, Grishman, and Dang, 2011; Mayfield, Artiles, and Dang, 2012; Surdeanu, 2013; Surdeanu and Ji, 2014]. Over the years, KBP has always featured two tasks that

are crucial to knowledge base construction: *Entity Linking* (see Section 3.2), and a so-called *Slot Filling* task, where missing facts about entities are retrieved and thus these *slots* are filled in a knowledge base. Since 2012, KBP also includes a *Cold Start* task, where a knowledge base is constructed from scratch and then evaluated as a whole.

The slot filling task is most relevant to this section. First of all, it evaluates the main aspect of knowledge base construction: to retrieve facts from a text corpus. Second, it is an example for searching with structured queries on text itself. The goal of the task is, given an entity (e.g., a particular person) together with the names of a number of relations (e.g., countries of residence), compute the missing objects (e.g., the countries of residence of the given person). All of them are attributes of either persons or organizations. Each query contains the name of the entity, its type (person or organization), and a link to one occurrence in the corpus of the task.

The text corpus consists of documents from multiple sources, with newswire text and web documents (1 million documents each) making up the biggest part. The knowledge base includes nodes for entities based on a dump of the English Wikipedia from October 2008. Results are evaluated against manually judged extractions based on pooling. Annotations from previous years are provided as additional training data to facilitate the use of the reference knowledge base.

The slot filling task is consistently lively with 15, 15, 11, 19, and 18 participants over the years. The best performing system in 2014 is the system by Angeli et al. [2014] described above. The system achieves 37% F-measure with 55% precision and 28% recall.

The cold start task introduced in 2012 has become its own track and replaced the classic KBP track in 2015 [Mayfield and Grishman, 2015]. Differently from the other tasks, no knowledge base is given as input. Instead, it is built from scratch using a given document collection and a predefined schema. This collection consists of 50K English documents from newswire text and discussion forum posts. According to the schema, systems have to recognize person, organization, and geopolitical entities (entity discovery task) , their relations (slot filling task) in the text corpus, and populate a knowledge base.

Also in 2012, the TREC conference series introduced a Knowledge Base Acceleration (KBA) track [Frank et al., 2012; Frank et al., 2013; Frank et al., 2014]. The streaming slot filling task is similar to the slot filling task of KBP, except that the data is given as a stream (with time stamps for each document), and the knowledge about entities evolves over time. As the stream of documents progresses, the entities change and evolve, so KBA systems must detect when vital, new information appears that would motivate an update to the knowledge base. The data comes from the Stream Corpus (see Table 2.1). Systems are evaluated by similarity between their slot fills and those found by humans (using cosine similarity between word vectors when taking all slot fills as bags of words). The best system [Qi et al., 2014] achieves a similarity of 61%. It makes use of training data from the non-streaming TAC KBP task (described above) to learn patterns on dependency parses of sentences (see Section 3.3).

4.4 Keyword Search on Knowledge Bases

Data	A knowledge base, as described in Section 2.1.2
Search	Keyword search, as described in Section 2.2.1 The query is a sequence of (typically few) keywords; the result is a SPARQL query or a ranked list of matching items from the knowledge base
Approach	Match keywords to entities from the knowledge base; generate candidates for SPARQL queries from these matching entities; rank candidate queries using graph, lexical, and IR measures; some overlap with the techniques from Section 4.8
Strength	Easier access to structured data for simple queries
Limitation	Is bound to fail for complex search intents that cannot be adequately (unambiguously) expressed by a keyword query

The main strength of a knowledge base is that even complex queries can be asked with precise semantics. The main drawback is that it is challenging to formulate these queries. For arbitrarily complex search requests, a complex query language is inevitable. However, for relatively simple search requests, a simpler kind of search is feasible. Keyword search has the strong benefit of being an established search paradigm that users are already accustomed to.

There is a small overlap with Section 4.8 on *Question Answering on Knowledge Bases*. According to our discussion at the beginning of Section 2.2, we distinguish systems by technique and not by the apparent form of the query. The core of the approaches in this section is to match keywords to entities and then find small subgraphs in the knowledge base connecting these entities. The approaches in Section 4.8 go further, for example, by also trying to match relation names, or by considering grammatical structure.

4.4.1 Basic Techniques

Systems for keyword search on knowledge bases, or more generally on relational databases, view the data as a graph. For knowledge bases, the graph structure is already given, for relational databases, it is induced, e.g., by foreign key relations. Keywords are then mapped to nodes in this graph. Typically, an inverted index over the (words of the) entity, class, or relation names is used. This allows to efficiently match keywords to nodes of the graph during run-time. Using standard techniques as described in Section 4.1, word variants and synonyms can be matched as well; for example, matching the keyword *cmu* to a knowledge base entity *Carnegie Mellon University*.

A problem is that keyword queries might mention relations differently from how they are represented in a knowledge base or might not mention them at all. For example, the keyword query *films by francis ford coppola* doesn't explicitly mention a *directed* relation. Therefore, systems try to connect the elements that were identified via the keywords, to form a connected (sub-)graph. This can be done by exploring the neighborhood of identified elements and finding the smallest (spanning) tree connecting all elements. Often, this is an instance of the Steiner tree problem, which is NP-complete. Hence, a lot of work tries to find efficient and good approximations. From the matched graph, a structured query can be derived, for example, by replacing identified classes with result variables.

Because words from the query can match several components of the graph, the translation results in several candidate queries which need to be ranked. Techniques for ranking these make use of two main factors: the relevance and the structure of the matching (sub-)graph. For relevance, ranking functions from information retrieval (see Section 4.1 on *Keyword Search on Text*) can be adapted to this setting, e.g., by assuming that each matched subgraph corresponds to a virtual document. In addition, the popularity of matching nodes (for example, derived via PageRank) and the quality of the keyword mappings (e.g., via the Levenshtein distance) can be considered. The structure of the matching graphs is incorporated, for example, by ranking smaller graphs higher. The intuition behind this is that simpler queries are more likely to be

correct. Similarly, the number of joins of the corresponding query can be considered (as a proxy for query complexity).

To improve usability, some systems also include user feedback in the translation process. This is done, for example, by suggesting keyword completions that lead to results, or by allowing the user to select the correct interpretation for each keyword (when several are possible).

Below, we first introduce systems designed for keyword queries on general relational databases, followed by systems specifically designed for knowledge bases.

4.4.2 Systems for Keyword Search on Relational Databases

Keyword search on relational databases is an actively researched field on its own. The survey by Yu, Qin, and Chang [2010] gives a good overview. Coffman and Weaver [2010] and Coffman and Weaver [2014] perform a qualitative evaluation of many state-of-the-art systems on a benchmark they introduce for that purpose (see below).

DBXplorer [Agrawal, Chaudhuri, and Das, 2002], DISCOVER [Hristidis and Papakonstantinou, 2002] and BANKS [Bhalotia et al., 2002] were the first prominent systems for keyword search on relational databases. DBXplorer and DISCOVER use the number of joins to rank answers, while BANKS tries to find the smallest matching subgraph. Subsequent work refines and combines the techniques mentioned above to improve results.

Tastier [Li et al., 2009] includes the user in answering keyword queries. In addition to translating to a SQL query it provides context-sensitive auto-completion of keyword queries, similar to what is described in [Bast and Weber, 2006]. This is achieved via specialized data structures (mainly a trie over words in the database) that allow computing completions of keywords that lead to results.

GraphLM [Mass and Sagiv, 2012] applies language models for ranking. Keyword queries are matched to subgraphs which correspond to a possible answer. Nodes in each subgraph have text associated with them via different fields: title (e.g., names), content (all attributes and their values) and structural (only attribute names). This allows learning a language model for each subgraph and field, which can then be

used to compute, e.g., $p(q|a_{title})$, the probability that a query q is generated by the title field of subgraph a. Ranking also incorporates node and edge weights. Intuitively, nodes with high in-degrees and unique edges are more important. The system outperforms all of the previous systems on the benchmark by Coffman and Weaver [2010] that is described in Section 4.4.2 below.

4.4.3 Systems for Keyword Search on Knowledge Bases

SemSearch [Lei, Uren, and Motta, 2006] was one of the first systems for keyword queries on knowledge bases. It accepts keyword queries with some additional structure, e.g., there is syntactic sugar for including *types* in queries and operators AND and OR are supported. An inverted index is used that maps keywords to classes, instances and properties of the knowledge base. The matching elements are combined in all possible ways using several query templates to obtain a structured query in SeRQL (a predecessor of SPARQL).

Tran et al. [2007] suggest a similar method to translate keyword queries to SPARQL queries. Keywords are mapped to entities (via their URIs and labels) of the knowledge base via an inverted index (implemented with Lucene). Starting at the matched entities, the knowledge base is explored in order to find subgraphs connecting the matched elements. The matching subgraphs are ranked by the lengths of their paths (with the intuition that smaller lengths correspond to better paths) and translated into a SPARQL query.

SPARK [Zhou et al., 2007] uses more sophisticated techniques, e.g., synonyms from WordNet and string metrics, for mapping keywords to knowledge base elements. The matched elements in the knowledge base are then connected by finding minimum spanning trees from which SPARQL queries are generated. To select the most likely SPARQL query, a probabilistic ranking model that incorporates the quality of the mapping and the structure of the query is proposed.

Zenz et al. [2009] follow an interactive and incremental approach to translate a keyword query into a SPARQL query. For each keyword provided by the user, a choice of a possible interpretation (with respect to the final SPARQL query) is presented. When the user selects an

interpretation for one keyword, the number of possible interpretations of the remaining keywords is reduced. This allows to incrementally construct complex SPARQL queries from keyword queries.

Hermes [Tran, Wang, and Haase, 2009] can search on multiple, possibly interlinked, knowledge bases.[16] In a preprocessing step, the knowledge bases are partly unified using maps between the various elements of the knowledge bases and their respective ontologies. Hermes also precomputes a map from potential search terms to elements of the knowledge bases. Keyword queries can then be mapped to candidate subgraphs in the resulting meta knowledge base. The candidates are ranked, preferring shorter paths containing important elements which match the keywords well. The system is evaluated on a combination of seven knowledge bases (including DBpedia, Freebase, and GeoNames, see Table 2.2) with a total of around 1.1B triples.

Pound et al. [2012] focus on keyword queries from the logs of the Yahoo web search engine. Keywords of a query are first tagged as entity, type, or relation mentions. The mentions are then arranged by mapping them to one of ten structured query templates. Both steps are learned via manually annotated queries from a query log using straightforward machine learning. In a final step, standard techniques as described above are used to map the mentions to entities and relations of a knowledge base. The system is evaluated using 156 manually annotated queries from the Yahoo query log and YAGO as a knowledge base.

4.4.4 Benchmarks

Coffman and Weaver [2010] introduce a benchmark on three datasets: selected data from Mondial[17] (geographical knowledge), IMDB, and Wikipedia, respectively. For each dataset, 50 queries were manually selected and binary relevance judgments for results are provided by identifying all possible correct answers. Evaluation metrics are those

[16] As such, it seems that Hermes should be described in Section 4.5 on *Keyword Search in Combined Data*. However, due to the unification process, the system is technically more similar to the systems in this section.

[17] http://www.dbis.informatik.uni-goettingen.de/Mondial

typical for information retrieval: precision at 1, mean reciprocal rank, and mean average precision. The authors evaluate nine recent state-of-the-art system on this benchmark. Many previous claims cannot be corroborated, which shows the shortcomings of previous evaluations. The evaluation also shows no clear winner, but that most systems score comparably on average, with different systems performing best on different datasets. In a follow-up evaluation, the GraphLM system [Mass and Sagiv, 2012] discussed above produced the consistently best results.

Balog and Neumayer [2013] assembled queries from a variety of previous benchmarks by mapping relevant entities to DBpedia. Some of these benchmarks were originally designed for semantic web data (like BTC; see Table 2.3), but the best systems mostly return results from the (comparably tiny) DBpedia part only. The new benchmark includes keyword queries (e.g., from the TREC Entity Tracks; see Section 4.5.3) as well as natural language queries (e.g., from QALD-2; see Section 4.8.5). Evaluation metrics are MAP (mean average precision) and precision at 10. Several baselines have been evaluated on the benchmark but adoption is slow. The current best performing system is from Zhiltsov, Kotov, and Nikolaev [2015], which achieves a MAP of 23%, an absolute improvement of 4% over one of the baselines.

In theory, benchmarks for question answering on knowledge bases (discussed in Section 4.8.5) could also be used by transforming natural language queries into keywords. In fact, some of those benchmarks also provide keyword versions of the questions. Obviously, this will fail when questions are more complex than what a keyword query can reasonably express.

4.5 Keyword Search on Combined Data

Data	Combined data, as described in Section 2.1.3 Specifically here, text with entity annotations or semantic web data
Search	Keyword search, as described in Section 2.2.1 Results are ranked lists of entities, maybe augmented with text snippets matching the query; optionally restricted to entities of a given type
Approach	For each entity, create a virtual text document from (all or a selection of) text associated with it; search these documents using techniques from Section 4.1; alternatively, first search given text using techniques from Section 4.1, then extract entities from the results and rank them
Strength	Easy-to-use entity search on combined data; works well when the data provides sufficiently strong relevance signals for the keyword, just as in keyword search on text
Limitation	Similar precision problems as for keyword search on text; see the box at the beginning of Section 4.1

Many keyword queries actually ask for an entity or a list of entities instead of a list of documents. In a study by Pound, Mika, and Zaragoza [2010] on a large query log from a commercial web-search engine, 40% of queries are for a particular entity (e.g., *neil armstrong*), 12% are for a particular lists of entities (e.g., *astronauts who walked on the moon*), and 5% are asking for a particular attribute of a particular entity (e.g., *birth date neil armstrong*).

Section 4.4 discusses one option for such queries: keyword search on knowledge bases. In this section, we consider combined data, which for keyword search is typically either semantic web data (multiple knowledge bases with different naming schemes and extensive use of string

literals; see Section 2.1.3) or text with entity annotations (this is the simplest form of text linked to a knowledge base; also see Section 2.1.3).

4.5.1 Basic Techniques

There are two prevalent approaches: search in virtual documents (one per entity) and standard keyword search on text followed by an entity extraction and ranking step.

In the *virtual document* approach, all or some of the data related to a particular entity (relation names, object names, string literals) is collected in a single virtual document for that entity. This makes particular sense for semantic web data, where the extreme heterogeneity of the data makes a structured search hard. Also, in some applications, there is a document per entity in the first place. A notable example is Wikipedia, which is used in all of the INEX benchmarks, discussed in Section 4.5.3 below. Given one document per entity (virtual or real), the result corpus can be searched using techniques from Section 4.1. The ranking of this kind of documents is discussed in detail in Section 5.1.1. Efficient indexing is discussed in Section 5.2.1. All of the systems described in Section 4.5.2 below are based on the virtual document approach, and they are all for semantic web data.

In the *search and extract* approach, the first step is keyword search on text. Many systems use one of the off-the-shelf systems from Section 4.1.2 for this task, or a web search engine like Google. In a second step, entities are extracted from the results (either from the full documents or only from the result snippets). This is trivial for collections like FACC (see Table 2.3), where entity annotations are part of the data. In a third step, entities are ranked. This is where the intelligence of systems using this approach lies. We hence describe them in Section 5.1.2 from our section on *Ranking*.

In both of these approaches, entity resolution (that is, different names or URIs for the same entity) is a challenge. The Semantic Web allows users to provide explicit links between such entities, notably via relations such as *owl:sameAs* or *dbpedia:redirect/disambiguate*. Not surprisingly, making use of such links can considerably improve result quality [Tonon, Demartini, and Cudré-Mauroux, 2012]. Section 5.1.1

describes a method that uses language models, which are normally used for ranking, for automatically establishing *owl:sameAs* links in semantic web data. Section 5.1.3 is about ranking interlinked entities (obtained from a semantic web search) in general, where *owl:sameAs* links also influence scores.

A special case of keyword search on combined data is *expertise retrieval*, where the goal is to retrieve a list of experts on a given topic. For example, find experts on *ontology merging* from a collection of W3C documents. The experts are persons, and it is either part of the problem to identify their mentions in the text (this is an instance of NER+NED; see Section 3.2) or these annotation are already provided. Note that the underlying knowledge base is then a simplistic one: just the entities (persons) and their names. The typical approaches are via virtual documents or via search and extract, as discussed above. A recent survey is provided by Balog et al. [2012].

4.5.2 Systems (all for Semantic Web Data)

Guha, McCool, and Miller [2003] describe an early prototype for search on the Semantic Web. At that time, hardly any semantic web data was available yet. The data was therefore artificially created via scraping[18] from a small selection of websites. Their main use case is single-entity search, that is, part or all of the query denotes an entity. Aspects discussed are disambiguation of entity names in the query (user interaction is suggested as a solution), disambiguation of entity names in matching documents (this is essentially the NER+NED problem from Section 3.2), and which of the usually many triples about the entity to show (various simple heuristics are discussed). The final result about the matching entity is shown in an infobox on the right, similar to how the large web search engines do it nowadays (except that those infoboxes do not come from the Semantic Web, but rather from a single, well-curated knowledge base).

Swoogle [Ding et al., 2004] was one of the fist engines to provide keyword search on the Semantic Web. Swoogle indexes n-grams to leverage

[18] Scraping refers to extracting structured data from ordinary websites, often via simple web-site specific scripts.

the information hidden in the often long URIs of entity and relation names. Also, an n-gram index enables approximate search. The index is augmented by metadata, so that search results can be restricted by certain criteria (e.g., to results in a particular language). The system also comprises a crawler and custom ranking function. As of this writing, there was still a demo available at `http://swoogle.umbc.edu`.

Falcons [Cheng, Ge, and Qu, 2008] provides a similar functionality as Swoogle, with the following additional features. The search can be restricted to entities of a given certain type (e.g., to type *conference* when the query[19] is *beijing 2008*). The search can also be restricted to a particular knowledge base (e.g., to only DBpedia). In the (default) entity-centric view, matching triples are grouped by entity, and for each entity a selection of the matching triples are displayed. Different URIs from the same entity are not merged. As of this writing, there was still a demo available at `http://ws.nju.edu.cn/falcons`.

Sindice [Oren et al., 2008] offers similar functionality on a distributed very large scale by using Hadoop and MapReduce. It also inspects schemata to identify properties that uniquely identify an entity, e.g., *foaf:personalHomepage*, which allows retrieval based on the property and its value. The system is not designed to be an end-user application but to serve other applications that want to locate information sources via an API. Unfortunately, as of this writing, the service was no longer available.

Glimmer [Blanco, Mika, and Vigna, 2011] constructs a virtual document for each entity using fielded BM25F. The particular index is described in Section 5.2.1. This allows customizing the contribution weight of contents from certain data sources and relations. Both quality and performance are evaluated on the WDC dataset (see Table 2.3) with queries from the SemSearch Challenge 2010 (see Section 4.5.3 below). Queries are keywords, possibly annotated by fields or relations they should match. As of this writing, a live demo is available under `http://glimmer.research.yahoo.com`.

[19] The WWW'08 conference, where this paper was presented, took place in Beijing.

As of this writing, there is no single system that searches the totality of semantic web data with a coverage and result quality even remotely comparable to that of the large commercial web search engines. This is largely due to the fact that, although the data is large in size, the amount of information contained is tiny compared to the regular web. It is also noteworthy that approaches with good results, like Glimmer above, boost high-quality contents like DBpedia. Indeed, as of this writing, all major commercial systems rely on internal well-curated knowledge bases; see Section 4.8 on *Question Answering on Knowledge Bases*.

4.5.3 Benchmarks

There are three notable series of benchmarks for keyword search on combined data, in particular, semantic web data: the TREC Entity Track (2009 - 2011), the SemSearch Challenge (2010 and 2011), and the INEX series of benchmarks (2006 - 2014). The QALD (Question Answering on Linked Data) benchmarks are described in Sections 4.8.5 (Question Answering on Knowledge Bases) and 4.9.4 (Question Answering on Combined Data).

We remark that participation in these competitions was low (generally below 10 participating groups, sometimes only a couple of participants). However, the datasets and queries continue to be used in research papers related to semantic search.

TREC Entity Track (2009 - 2011): An overview of each of the three tracks is provided in [Balog et al., 2009; Balog, Serdyukov, and Vries, 2010; Balog, Serdyukov, and Vries, 2011]. A typical query is:

airlines that currently use boeing 747 planes

The central entity of the query (*boeing 747*) and the type of the target entities (*airlines*) was explicitly given as part of the query. There were two kinds of datasets: text (ClueWeb, see Table 2.1) and semantic web data (BTC'10, see Table 2.3).[20]

[20] In the TREC Entity Track 2009, only ClueWeb'09 was used. In the TREC Entity Track 2011, the Sindice dataset was used instead of BTC'10. However, the Sindice dataset is no longer available, which is why we do not list it in Table 2.3.

The best systems that worked with the BTC'10 dataset used the virtual document approach described in Section 4.5.1 above. That is, although the queries appear more as natural language queries (see the example above), the processing is clearly keyword search style. According to our discussion at the beginning of Section 2, we make the distinction between these two kinds of search by technique. This also explains why we describe this benchmark in this section and not in Section 4.9 on *Question Answering on Combined Data*.

The best system that worked with the ClueWeb'09 dataset used the extract and search approach described in Section 4.5.1 above. It is mainly about ranking, and hence described in Section 5.1.2 (right at the beginning). Interestingly, the system chose to ignore the official dataset and instead used Google Search for the initial retrieval step.

The best results for the main task (related entity finding, like for the query above) were an nDCG@R of 31%, 37%, and 25% in 2009, 2010, and 2011, respectively. The best result for the related task of entity list completion (where some result entities are given) was a mean average precision of 26% in 2010.

SemSearch Challenge (2010 and 2011): An overview over each of these two challenges is provided in [Halpin et al., 2010] and [Blanco et al., 2011]. In 2010, queries were keyword queries asking for a single entity (for example, *university of north dakota*). In 2011, there were two tasks: keyword queries for a single entity (like in 2010, but new queries) and keyword queries for a list of entities (for example, *astronauts who landed on the moon*). Both challenges used the BTC'09 dataset (see Table 2.3).

The best approaches again construct virtual documents and use a fielded index and corresponding ranking function. The winner in 2010 achieved a precision at 10 of 49% and a mean average precision of 19%. In 2011, the best result for the single-entity task was a precision at 10 of 26% and a mean average precision of 23%. The best result for the entity-list task was a precision at 10 of 35% and a mean average precision of 28%.

INEX (2006 - 2014): INEX (Initiative for the Evaluation of XML Retrieval) has featured several ad-hoc search tasks. The dataset was

Wikipedia with an increasing amount of annotations, all represented in XML; see Section 2.1.3 for an example. From 2006 - 2008, annotations were obtained from Wikipedia markup (in particular: infoboxes, links, and lists). In 2009, cross-references to entities from YAGO (see Table 2.2) were added to the Wikipedia links, as well as for each page as a whole. In 2012, additional cross-references to DBpedia (see Table 2.2) were added. The resulting dataset is Wikipedia LOD (see Table 2.3).

The goal of the early ad-hoc tasks (2006 - 2010) was similar to that of the TREC ad-hoc tasks described in Section 4.1.3. Queries were also similar, for example, *olive oil health benefit* (from 2013) or *guitar classical bach* (from 2012). One notable difference was the focus on the retrieval of (XML) elements rather than whole documents, and the incorporation of the proper focus of these elements (not too large and not too small) in the quality measure. See [Gövert et al., 2006] for an overview paper on this aspect of XML retrieval.

The INEX Entity Ranking Track (2007 - 2009) is similar to the TREC Entity Track from above: given a keyword query (describing a topic) and a category, find entities from that category relevant for that topic. For example, find entities from the category *art museums and galleries* that are relevant for *impressionist art in the netherlands* (from 2007).

The INEX Linked-Data Track (2012 and 2013) explicitly encouraged the use of the external knowledge bases (YAGO and DBpedia) to which the Wikipedia content was linked. However, few participants made use of that information and the results were inconclusive.

Our take on the usability of XML-style retrieval for semantic search is as follows. XML shines for deeply nested structures, with a mix between structured and unstructured elements. Indeed, query languages like XPath and XQuery are designed for precise retrieval involving complex paths in these structures. However, datasets actively used in semantic search at the time of this writing have a flat structure (triples or simple links from the text to entities from the knowledge base; see Tables 2.2 and 2.3). The core challenge lies in the enormous size and ambiguity of the data (queries, text, and entity and relation names), which is nothing where XML can specifically help.

4.6 Semi-Structured Search on Combined Data

Data	Combined data, as described in Section 2.1.3 Specifically here, text linked to a knowledge base
Search	Structured search, as described in Section 2.2.2, extended with a keyword search component Results are ranked lists of entities, maybe augmented with matching text snippets or matching information from the knowledge base
Approach	Store data in an inverted index or extensions of it; use separate indexes for the text and the knowledge base or use tailor-made combined indexes; provide special-purpose user interfaces adapted for the particular kind of search
Strength	Combines the advantages of text (widely available) and knowledge bases (precise semantics); good for expert search and as a back end for question answering
Limitation	Queries with a complex structured part have the same usability problems as described at the beginning of Section 4.2

Since combined data contains both structured and unstructured elements, it is natural that queries also contain a mix of structured and unstructured elements. Simple text search extensions of SPARQL are discussed already in Section 4.2. This section considers more sophisticated extensions.

4.6.1 Basic Techniques

Text linked to a knowledge base allows searches for co-occurrences of arbitrary keywords with arbitrary subsets of entities, as specified by a structured query on the knowledge base. A simple example would be to search for all *politicians* (all entities in the knowledge base with that

profession) that co-occur with the keywords *audience pope*. This could be expressed as an extended keyword query, where some keywords are concepts from the knowledge base, for example:

type:politician audience pope

This kind of search can easily be supported by an inverted index, with an artificial index item (like the *type:politician*) added for each mention of a politician in the text. Alternatively, XML search engines supporting languages like XPath or even XQuery could be used. However, this would be cracking a nut with a sledgehammer; see the discussion at the end of Section 4.5.3, after the description of the INEX benchmarks.

As a more complex example, consider the example query from the introduction *female computer scientists who work on semantic search*. This is naturally expressed as a structured query (that expresses the knowledge base part) extended with a keyword search component (that expresses the co-occurrence with the given keywords). In the syntax of the Broccoli system, discussed below, this can be written as:

SELECT ?p WHERE {
?p has-profession Computer_Scientist .
?p has-gender Female .
?p occurs-with "semantic search" }

For this more general class of queries the simple annotation trick fails, at least for a knowledge base of significant size. We then cannot annotate each entity in the text with all the information that is available about it in the knowledge base. The ways to index such data, as well as their strengths and limitations, are discussed in detail in Section 5.2.2 on *Semi-Structured Search Based on an Inverted Index*.

It is important to understand the difference between a relation like *occurs-with* and a simple text search extension like *bif:contains* discussed in Section 4.2.1. Consider the query above with the last triple replaced by

?p has-description ?d . ?d bif:contains "semantic AND search"

That query requires each matching entity to stand in a *has-description* relation to a string literal containing the desired keywords. This is

unlikely to be fulfilled by a typical knowledge base. In contrast, the original query from above only requires that a matching entity co-occurs with the given keywords somewhere in the text corpus. This is realistic for a sufficiently large text corpus.

4.6.2 Systems

KIM [Popov et al., 2004] was one of the first systems to provide semi-structured search on text linked to a knowledge base, as described above. Results are documents that mention entities from the structured part of the query as well as the specified keywords. The text is indexed with Lucene (see Section 4.1.2), including for each entity an inverted index of the occurrences of that entity in the text. The knowledge base is indexed with Sesame (see Section 4.2.2). The results from the two indexes are combined by computing the union of the inverted lists of the entities matching the structured part of the query. This runs into efficiency problems when the structured part matches very many entities (for example, a structured query for just *person*).

Ester [Bast et al., 2007] provides similar functionality as KIM, but achieves scalability with a special-purpose combined index, adapted from [Bast and Weber, 2006]. The index also provides fast query suggestions after each keystroke, for words from the text as well as for elements from the knowledge base. The system was evaluated on a variant of the Wikipedia LOD dataset (see Table 2.3).

Broccoli [Bast et al., 2012; Bast et al., 2014b] provides extended keyword search as well as extended structured search; an example query for the latter is given above. The structured part of the query is restricted to tree-like SPARQL queries. Co-occurrence of entities from the text with entities from the knowledge base can be restricted to the semantic contexts from [Bast and Haussmann, 2013], as explained in Section 4.3.1 on *Relationship Extraction from Natural Language Text*. Interactive query suggestions are provided, and an elaborate user interface is provided. Results can be grouped by entity, with matching text snippets. A tailor-made index for the efficient support of these features is provided, which is explained in Section 5.2.2. The system is evaluated on a variant of the Wikipedia LOD dataset (see Table 2.3) with

queries adapted from the TREC Entity Track 2011 and the SemSearch Challenge 2011, as explained in Section 4.6.3 below. As of this writing, a live demo is available: `http://broccoli.cs.uni-freiburg.de`.

Mìmir [Tablan et al., 2015] is an extension of KIM. Compared to KIM, simple queries are implemented more efficiently (for example, a search for *cities* that occur with certain keywords), and full SPARQL is supported for the structured part of the query (though not particularly efficiently when combined with keyword search). For the text corpus, MG4J is used (see Section 4.1.2). The ranking function is customizable, in particular, BM25 is supported (see Section 4.1.1). Results are matching documents, grouping by entities is not supported. The software is open source.

4.6.3 Benchmarks

There are no widely used benchmarks that are explicitly designed for semi-structured search on combined data.

However, the benchmarks from Section 4.5.3 (TREC Entity Track, SemSearch Challenge, and INEX) can be easily adapted for this scenario. Namely, most of the queries of these benchmarks have a part pertaining to information best found in a knowledge base and a part pertaining to information best found in text. For example, for the query *astronauts who landed on the moon* (SemSearch Challenge 2011, entity-list task), the information who is an astronaut is best taken from a knowledge base, whereas the information who landed on the moon is best found in text. The semi-structured representation for this query is similar to the example given in Section 4.6.1 above.

The Broccoli system, discussed in Section 4.6.2 above, has adapted the queries from the TREC Entity Track 2009 (main task: related entity finding) and the SemSearch Challenge 2011 (entity-list task) in this manner. On a variant of the Wikipedia LOD dataset (Table 2.3), an nDCG of 48% and 55%, respectively, is achieved.

The queries for the QALD (Question Answering on Linked Data) benchmarks, which are described in Sections 4.8.5 and 4.9.4, can be adapted in a similar way. QALD-5 features a track with explicitly semi-structured queries; see Section 4.9.4.

4.7 Question Answering on Text

Data	Text documents, as described in Section 2.1.1
Search	Natural language queries, as described in Section 2.2.3; the results are passages or statements from the text that answer the question
Approach	Derive suitable keyword queries and the answer type from the question; extract answer candidates from the (many) result snippets and rank them; optionally use reasoning and an external general-purpose knowledge base
Strength	The most natural kind of queries on the most abundant kind of data
Limitation	Questions that require combination of facts not found in the text; or questions with complex structure

Question answering on text became popular in the early 1990s, when large amounts of natural language texts started to become available online. With the advent of the world wide web, the field blossomed. According to the scope of this survey, as explained in Section 1.2, we here focus on so-called *extractive* question answering, where the desired answers can be found in the text and no synthesis of new information is required. Indeed, most research on question answering on text is of exactly this kind.

4.7.1 Basic Techniques

Prager [2006] gives an excellent survey of the development of the field until 2006. The survey by Kolomiyets and Moens [2011] focuses on techniques (and less on complete systems) and surveys some more recent research like co-reference resolution and semantic role labeling (as discussed in Section 3.3).

In 2007, the popular series of *TREC Question Answering* benchmarks (described below) ended. In all issues, a single system, Lymba's PowerAnswer and its predecessors, beat the competing systems by a large margin. We briefly describe that system below. At the time of this writing, we still consider it the state of the art in (extractive) question answering on text.

The field has since moved away from only text as a data source. Just around the time of the last TREC QA benchmark, large general-purpose knowledge bases like YAGO, DBpedia, and Freebase (see Table 2.1.2) started to gain momentum and comprehensiveness. This spawned extensive research activity on question answering on such knowledge bases, which we describe in Section 4.8 on *Question Answering on Knowledge Bases*. Note that a question like:

what is the average gdp of countries with a literacy rate below 50%

is relatively easy to answer from a knowledge base, but very hard to answer from text alone (unless the text contains that piece of information explicitly, which is unlikely).

At about the same time, IBM started its work on *Watson*, aimed at competing against human experts in the Jeopardy! game. Watson draws on multiple data sources, including text as well as the knowledge bases just mentioned. Therefore, we describe that work in Section 4.9 on *Question Answering on Combined Data*.[21]

Search engines like WolframAlpha or Google also accept natural language queries, but as of this writing, the answers do not come from text, but rather from an internal (well-curated) knowledge base; see Subsection 4.8.4 of the section on *Question Answering on Knowledge Bases*.

4.7.2 The START System

START [Katz, 1997; Katz, Borchardt, and Felshin, 2006] was the first web-based question answering system. It is one of the few systems with

[21] John Prager, the author of the above-mentioned survey, was a member of the team working on Watson.

a reliable online demo[22], which has been up and running continuously since 1993 to this day. It answers natural language queries by first extracting structured information (basically: nested subject-predicate-object triples) from sentences and storing them in a knowledge base. Compared to full-fledged knowledge base construction, as described in Section 4.3 on *Structured Data Extraction from Text*, the constructed knowledge base does not have a single consistent schema and is fuzzy. The system answers questions by transforming them into the same triple-like representation and matching them against the knowledge base. Matched facts are then translated back to a natural language sentence that is presented to the user.

4.7.3 The PowerAnswer System

We briefly describe Lymba's PowerAnswer [Moldovan, Clark, and Bowden, 2007], the undisputed winner of the TREC Question Answering track. The system can still be considered state of the art at the time of this writing. In particular, its basic architecture is typical for a system that does question answering on text.

Depending on the type of question (factoid, list, definition etc.), the system implements different strategies. Each strategy has the following main components:

Answer type extraction: determine the answer type of the query; for example, the answer for *who ...* could be a person or organization, but not a place or date.

Keyword query generation: generate one or more keyword queries, which are then issued to a text search engine, with standard techniques as described in Section 4.1.

Passage retrieval: retrieve passages from the documents matching these keyword queries that could possibly be an answer to the question.

Answer extraction: extract potential answers from the retrieved passages; rank those answers by a score that reflects the "relevance" to and the degree of "semantic match" with the question.

[22] `http://start.csail.mit.edu/index.php`

The process involves subsystems solving many of the natural language processing problems discussed in Section 3. In particular, answer extraction often makes use of POS tagging, chunking, and named-entity recognition and disambiguation. Particular kinds of entities relevant for the kind of questions asked in the TREC benchmarks are *events*, *dates*, and *times*.

In the TREC benchmarks, the answer is eventually to come from the reference text corpus (typically AQUAINT, as described below). However, PowerAnswer also issued keyword queries against external sources like amazon.com, imdb.com, and Wikipedia to find candidate answers. These were then used, in turn, to find correct answers in the TREC collection.

Candidate answers are ranked using pre-trained language models and scoring functions, using state-of-the-art techniques known from keyword search on text as described in Section 4.1. PowerAnswer also makes use of COGEX, a logic prover with basic reasoning capabilities, to re-rank candidate answers. COGEX is similar to the inference engines we describe in Section 5.4 on *Inference and Reasoning* (where we restrict ourselves to publicly available systems). It generates a logic form of the question and candidate answer and performs a proof by contradiction. As part of the reasoning it also makes use of real world knowledge (e.g., that Sumatra is a part of Asia) and natural language statements (e.g., that the verb *invent* is a hyponym of *create*). The proofs (if they succeed) output a confidence score, which depends on the rules and axioms that were applied. The score is used as part of ranking the candidate answers.

4.7.4 Benchmarks

The TREC Questions Answering Track ran from 1999 - 2007, with 9 issues altogether. There is a comprehensive overview article for each year, describing the individual tasks as well as the participating systems and their results [Voorhees, 1999; Voorhees, 2000; Voorhees, 2001; Voorhees, 2002; Voorhees, 2003; Voorhees, 2004; Voorhees and Dang, 2005; Dang, Lin, and Kelly, 2006; Dang, Kelly, and Lin, 2007]. Participation was strong, with at least 20 participating groups, peaking

in 2001 with 36 groups. Lymba's PowerAnswer, described above, and its predecessors participated and dominated the competition in each year. For example, in 2007, PowerAnswer scored an accuracy of 70.6% on factoid questions and of 47.9% on list questions with a runner-up accuracy of 49.4% and 32.4%, respectively.

All tracks made use of the AQUAINT or AQUAINT2 text corpus. The last two tracks also made use of the BLOG'06 corpus. These datasets are described in Table 2.1.

The TREC Entity Tracks (2009 - 2011) featured entity-centric search on ClueWeb (see Table 2.1) as one of their tasks. Overview papers and example queries are provided in Section 4.5 on *Keyword Search on Combined Data*, since other tasks from these tracks used semantic web data. The systems working on ClueWeb used similar techniques as described for Lymba's PowerAnswer above. However, the Entity Track tasks additionally required that systems return an authoritative URL for each result entity, and not just its name. This made the task considerably harder.

In 2015, a new TREC *LiveQA* track was initiated, with the goal to "revive and expand the [Question Answering track described above, but] focusing on live questions from real users". However, many of the questions asked there can hardly be considered *extractive*. One of the three examples questions from the track's call for participation reads:

Is the ability to play an epic guitar solo attractive in a woman? Or do you see it as something aggressive and a turn off?

Apart from being sexist, such questions usually require synthesis of new information and are hence out of scope for this survey.

4.8 Question Answering on Knowledge Bases

Data	A knowledge base, as described in Section 2.1.2
Search	Natural language queries, as described in Section 2.2.3; the result is a SPARQL query or a ranked list of matching items from the knowledge base
Approach	Generate candidates for SPARQL queries by analyzing the structure of the question and mapping parts of the question to entities and relations from the knowledge base; rank query candidates and execute the top query to retrieve the answer
Strength	User-friendly access to the growing amount of data that is available in knowledge bases
Limitation	Very hard for complex queries, especially when the knowledge base is large; only a fraction of the world's information is stored in knowledge bases

Leveraging the rapidly growing amount of information in knowledge bases via natural language queries is a relatively young field. There is some overlap with Section 4.4 on *Keyword Search on Knowledge Bases*, which is discussed at the beginning of that section. The difference becomes clearer when reading and comparing Subsection 4.4.1 from that section and Subsection 4.8.1 below.

4.8.1 Basic Techniques

The goal of the typical systems from this section is the same as for the typical systems from Section 4.4: *translate* the given question to a (SPARQL) query that expresses what the question is asking for. The basic mechanism is also similar to that described in Section 4.4.1: consider a set of candidate queries (which stand for possible interpretations of the question) and from that set pick the one that represents the given question best. However, the way these candidate sets are generated and

how the best query is selected from that set is more sophisticated, going much more in the direction of what could be called "understanding" the question.

As in Section 4.4, recognizing entities from the knowledge base in the query (the NER+NED problem from Section 3.2) is a crucial component. However, all of the systems in this section also try to recognize *relation names* from the knowledge base in the question. This is harder than recognizing entities, because of the much larger variety in which relation names can be expressed in natural language.

A typical approach for recognizing relation names is via indicator words or synonyms that are learned from a text corpus by distant supervision (explained in Section 4.3.2) or by using datasets obtained via distant supervision, e.g., Patty [2013]. Another approach is to use corpora of paraphrased questions, such as Paralex [2013], to derive common paraphrases.

Natural language questions are often longer and provide more information than keyword queries. For example, compare *in what films did quentin tarantino play* to *quentin tarantino films*. The natural language question is more explicit about the expected type of result (*films*) and more precise about the relation (films in which Quentin Tarantino acted, not films which he directed). At the same time, natural language questions can also be more complex. For example, *who was born in vienna and died in berlin*.

Some of the systems below exploit this additional information by performing a linguistic analysis of the question. This is done with existing taggers and parsers (see Sections 3.1 and 3.3), or by training new special-purpose parsers. The result provides the linguistic or semantic structure of the question, which can be used to generate a template for a SPARQL query. It remains to fill in the entity and relation names. It turns out that a joint optimization of the structure (and hence the query template) and the entity and relation names works better than solving the two problems independently.

Selecting a query from the set of candidate queries is also more complex than for the systems in Section 4.4. The techniques sketched above provide a rich set of features for determining how well a candidate

query matches the given question. A typical approach is to use these features for learning to rank the candidates from given training data. This enables solving even hard questions (in the sense that the correct SPARQL query is hard to find using simple matching techniques) as long as there are enough examples in the training data. For example, answering the question *who is john garcia* with *singer* requires understanding that the *who is* part of the question is asking for the profession of the person that follows.

Section 4.8.5 below describes three widely used benchmarks: Question Answering on Linked Data (QALD), Free917, and WebQuestions. The QALD benchmarks sparked from the semantic web community, while Free917 and WebQuestions were initiated by the computational linguistic community. We first describe systems that were evaluated on QALD, followed by systems evaluated on Free917 and WebQuestions.

4.8.2 Systems Evaluated on QALD

The AutoSPARQL system [Unger et al., 2012] bases its translation on a linguistic analysis of the question. Using a lexicon of manually-designed domain-independent expressions (such as *most* or *more than*) query templates are instantiated from the structure of the question. To derive SPARQL queries, the templates are instantiated with elements from the knowledge base. Queries are then ranked by preferring prominent entities but also by considering string similarities of the knowledge base mapping. The system was evaluated on 39 of the 50 questions of QALD-1, of which it was able to answer 19 perfectly.

DEANNA [Yahya et al., 2012] formulates the task of translating a given question to a SPARQL query as an integer linear program. The program incorporates the identification of concept and relation phrases in the question, mapping these to the knowledge base, and a dependency parse to generate (SPARQL) triple candidates. Aliases from YAGO2s [2011] and relation phrases from ReVerb [Fader, Soderland, and Etzioni, 2011] are used to map to entities and relations from the knowledge base. Additionally, semantic coherence and similarity measures are incorporated. The system was evaluated on QALD-1, where it was able to answer 13 out of 27 questions correctly.

Xser [Xu, Feng, and Zhao, 2014] performs the translation in two separate phases. The first phase identifies relevant phrases (mentioned entities, relations, types) in the question, independently of the knowledge base. The identified phrases are arranged in a DAG to represent the structure of the question. Training data is used to learn a model and parser for this. The second phase maps the identified phrases to entities and relations from the knowledge base. For the experiments on DBpedia, the Wikipedia miner tool[23] is used to find matching entities, and the data from Patty [2013] is used to map to relations. Xser was the best performing system at QALD-4 and QALD-5, beating other systems by a wide margin (more than 30% absolute F-measure). According to the authors (private communication), the system achieves 69% and 39% accuracy on Free917 and WebQuestions, respectively. This is about 10% below the current state of the art on these benchmarks (see below).

4.8.3 Systems Evaluated on Free917 and WebQuestions

Sempre [Berant et al., 2013a] produces a semantic parse of a question by recursively computing logical forms corresponding to subsequences of a question. The generation is guided by identified entities in the question, a mapping of phrases to relations from the knowledge base, and a small set of composition rules. Logical forms are scored with a log-linear model and translated into a corresponding SPARQL query on Freebase. Sempre achieves 62% accuracy on Free917 and 36% accuracy on WebQuestions.

Parasempre [Berant and Liang, 2014] uses a set of fixed query patterns that are matched to each question. Each pattern is then translated back into a canonical natural language realization using a set of rules and the Freebase schema. A log-linear model chooses the realization that best paraphrases the original question. The model utilizes word vector representations and a corpus of paraphrases [Paralex, 2013]. Parasempre achieves 69% accuracy on Free917 and 40% accuracy on WebQuestions.

[23] `https://github.com/dnmilne/wikipediaminer`

Graphparser [Reddy, Lapata, and Steedman, 2014] uses distant supervision to generate learning examples (questions and their answer) from natural language sentences. Intuitively, this is achieved by removing an identified entity from a sentence and reformulating the sentence as a question for that entity. To translate a question, an existing CCG parser (a kind of constituent parser) is used to retrieve a logical form. This logical form is then matched to a graph in which identified entities and relations are mapped to Freebase. Graphparser was evaluated on a subset of Freebase, where it achieves 65% accuracy on Free917 and 41% accuracy on WebQuestions.

Bordes, Chopra, and Weston [2014] take an alternative approach that involves neither named-entity recognition nor sentence parsing, and not even POS tagging. Instead, word vectors (see Section 3.4) of words and of entities and relations from Freebase are learned. This is done by using the given training data, augmented by synthetically generated question answer pairs. The idea is to learn the embeddings in such a way that the embedding of a question is close to the embedding of its answer entity. No intermediate structured query is generated. The system achieves 39% accuracy on WebQuestions.

Aqqu [Bast and Haussmann, 2015] directly constructs a SPARQL query by matching a fixed set of query patterns to the question. The patterns are matched by first identifying candidates for entity mentions in the question. Candidate queries are then generated by matching patterns on the subgraphs of these entities in the knowledge base. This way, only candidates that have an actual representation in the knowledge base are created. The candidates are ranked using a learning to rank approach. Features include the quality of entity matches and besides others, distant supervision and n-gram based approaches of matching the relations of a candidate query to the question. For entity synonyms, CrossWikis [2012] is utilized. The system achieves 76% accuracy on Free917 and 49% accuracy on WebQuestions.

STAGG [Yih et al., 2015], like Aqqu, directly constructs a SPARQL query using the knowledge base. Starting from identified entities it also incrementally constructs query candidates. To control the search space, STAGG only considers a limited number of top candidates, scored by

a learned function, for extension at each step. For scoring the candidates it also uses a learning to rank approach. In contrast to Aqqu, it uses more sophisticated techniques based on deep learning for matching relations of query candidates to the question. It also allows adding constraints to queries (e.g., *first* or *last* for dates) and, in theory, allows arbitrary patterns to be generated. In practice, however, patterns are constrained (very similar to those of Aqqu) in order to keep the search space tractable. The system achieves 53% accuracy on WebQuestions.

4.8.4 Commercial Systems

WolframAlpha can answer questions about general knowledge. As of this writing, no technical publications were available, but their FAQ[24] is quite informative concerning the scope and basic techniques used. On the back end side, Wolfram Alpha uses its own internal knowledge base, which is a carefully curated combination of various high-quality knowledge bases. It also uses real-time data (like weather or market prices), which is curated using heuristics. NLP techniques are used, combined with publicly available data. For example, Wikipedia is used for linguistic disambiguation (such that *the big apple* is a synonym for *NYC*). The implementation uses Mathematica as a programming language.

Facebook Graph Search[25] supports personalized searches on the relations between persons, places, tags, pictures, etc. An example query is *photos of my friends taken at national parks*. Results are based on the relationships between the user and her friends and their interests expressed on Facebook. Graph Search was introduced by Facebook in March 2013. It was reduced to a much restricted version (eliminating most search patterns) in December 2014, mainly due to privacy issues.

Google Search answers an increasing fraction of natural language queries from its internal knowledge base, called *Knowledge Graph*. As of this writing, the Knowledge Graph is based on Freebase (and not on the much larger *Knowledge Vault* described in Section 4.3.4) and there is no published work on how this search works.

[24] `http://www.wolframalpha.com/faqs.html`

[25] `http://en.wikipedia.org/wiki/Facebook_Graph_Search`

4.8.5 Benchmarks

Question Answering over Linked Data (QALD) [Lopez et al., 2011b; Lopez et al., 2012; Cimiano et al., 2013; Unger et al., 2014; Lopez et al., 2013; Unger et al., 2015] is an annual benchmark of manually selected natural language queries with their SPARQL equivalent. The questions are of varying complexity, for example:

Who is the mayor of Berlin?
What is the second highest mountain on Earth?
Give me all people that were born in Vienna and died in Berlin.

The name seems to imply semantic web data, but the datasets are DBpedia and MusicBrainz (see Table 2.2), which we consider as knowledge bases in this survey. For the first version of the benchmark (QALD-1) 50 training questions and 50 test questions were used. Later versions used between 50 and 100 training and test questions. Systems were evaluated by comparing the set of answers returned by a system to the answers in the ground truth (i.e., those returned by the correct SPARQL query) and computing precision and recall for each question. Averages of these on all queries and the resulting F-measure are used to compare systems globally.

The benchmark started in 2011 (QALD-1) with 2 participating groups. Since then participation has constantly increased to 7 groups for QALD-5. Later versions included questions in multiple languages and hybrid questions that require combining search on text as well as knowledge bases. The best system at QALD-4 and QALD-5 was Xser [Xu, Feng, and Zhao, 2014], described above, with an F-measure of 72% and 63%, respectively.

Free917 [Cai and Yates, 2013] is a benchmark consisting of 917 questions and their structured query (SPARQL) equivalent on Freebase. For example, *when was starry night painted* and:

SELECT DISTINCT ?x WHERE {
fb:en.de_sternennacht fb:visual_art.artwork.date_completed ?x }

The goal is, given the question (and knowing the schema of Freebase), to automatically compute the corresponding structured query. Questions and their SPARQL equivalent were constructed manually. All

questions are such that the corresponding entities and relation indeed occur in Freebase; this makes the benchmark simpler than a real-world task with arbitrary questions from real users. 30% of the questions are explicitly marked as test questions and 70% are reserved for learning. As an evaluation metric, accuracy (the percentage of questions answered exactly as in the ground truth) is used. The current best system, Aqqu [Bast and Haussmann, 2015], achieves an accuracy of 76%.

WebQuestions [Berant et al., 2013b] is a benchmark that consists of 5,810 questions and their answers from Freebase (i.e., no corresponding SPARQL query). For example, *what type of music did vivaldi write* and the answer *classical music*. In order to obtain the questions, 100,000 candidates were generated using the Google Suggest API and Amazon Mechanical Turk was used to identify those, which actually have an answer in Freebase. These questions are more realistic (i.e., more colloquial) than those of Free917, which also makes the benchmark considerably harder. 40% of the questions are used as test questions and 60% are reserved for learning. The average F-measure over all test questions is used as evaluation metric. This is computed by comparing the result set of a system to the result set in the ground truth for each question and computing individual F-measures and their average. The current best system, STAGG [Yih et al., 2015], achieves an F-measure of 53%.

4.9 Question Answering on Combined Data

Data	Combined data, as described in Section 2.1.3: text linked to a knowledge base, multiple knowledge bases, or semantic web data
Search	Natural language queries, as described in Section 2.2.3; the result is (close to) the answer one would expect from a human
Approach	A melting pot of all techniques from the previous sections; plus techniques to evaluate the confidence and combine the answers from the various sources; current approaches are still relatively simplistic, however
Strength	The ultimate "semantic search": free-form queries on whatever data there is
Limitation	This line of research is still in its infancy; but it will be the future

In a sense, this is the ultimate "semantic search". Users can formulate queries in natural language, and the system draws on a variety of data sources to answer it: text, knowledge bases, and combinations of the two (including semantic web data).

As of this writing, there is still little research for this scenario. In particular, we know of only one recent benchmark (the QALD-5 hybrid track with only three participants) and few notable systems; see below. This is understandable given what we have seen in the last two subsections: that natural language queries are hard already when restricting to "only" textual data or when restricting to (usually a single) knowledge base.

4.9.1 Basic Techniques

Technically, question answering on combined data is a big melting pot of techniques, in particular, those from the three previous subsections:

Question Answering from Text, *Question Answering from Knowledge Bases*, and *Keyword or Semi-Structured Search on Combined Data*, which in turn draw heavily on techniques from the previous subsections. In Section 4.9.2, we provide a longer description of the popular Watson system. Apart from Watson, there has been little research on this topic so far. In Section 4.9.3 we describe a recent system.

Commercial search engines like Google also provide question answering capabilities on both text and knowledge bases. At the time of this writing, there is no published work on how these subsystems are combined. However, answers appear to come from two different subsystems. If the answer comes from the knowledge base, the result is displayed in an infobox on the right, or as a list of entities on the top of the usual search results. If the answer comes from annotated text, it is displayed with a corresponding snippet, again on top of the usual result list. So far, there is no evidence of a deeper integration of the two kinds of data.

A survey that addresses question answering specifically for the Semantic Web is provided by Lopez et al. [2011a].

4.9.2 Watson

IBM's Watson [Ferrucci et al., 2010; Ferrucci et al., 2013] was developed to compete with human experts in the well-known Jeopardy! game show. In Jeopardy, the question is formulated as an assertion (called "claim") and the answer has to be formulated as a question. The following example clarifies that this is just an entertaining twist of classical question answering; technically the transformation of one to the other is trivial.

Classical: What drug has been shown to relieve the symptoms of ADD with relatively little side effects? Ritalin.

Jeopardy: This drug has been shown to relieve the symptoms of ADD with relatively few side effects. What is Ritalin?

The goal of Watson was to answer roughly 70% of the questions with greater than 80% precision in 3 seconds or less. This would be enough

to beat the best human experts in the game; a goal eventually reached in a much publicized show in 2011.

Watson answers questions using both text and knowledge bases. Among the text data sources are: Wikipedia, several editions of the Bible, and various encyclopedias and dictionaries. These were expanded to contain text extracted from the Web. Overall, the corpus contained 8.6 million documents with a size of 59 GB. Among the knowledge bases are: DBpedia, YAGO, and Freebase (see Table 2.2).

The Watson system consists of a pipeline of steps. Each step is very carefully designed and adjusted to the particular type and distribution of Jeopardy questions. The steps are of varying complexity and make use of state-of-the-art techniques where necessary, but also resort to simple but effective heuristics when sufficient. Here, we outline the main steps and relate them to other techniques in this survey when appropriate. For a comprehensive technical description, we refer to a special issue of the IBM Journal by Pickover [2012] consisting of a series of twelve papers (each about 10 pages) solely about Watson.

Question analysis: First, the *focus* and *lexical answer type* of the question is determined. For example, in

A new play based on this Sir Arthur Conan Doyle canine classic opened on the London stage in 2007.

the focus is *this* and the lexical answer type is *classic*. This is done using manually designed rules, e.g., "use the word *this* as focus and use its head word (*classic*) as a lexical answer type". The rules make use of a linguistic analysis of the question, e.g., a syntactic parse, its logical structure (similar to semantic role labeling) and identified named entities; see Section 3 on *Basic NLP Tasks in Semantic Search*.

Using rules, the question is also categorized into different types, e.g., *puzzle* or *definition* question. These require slightly different approaches later on.

Relations mentioned in the question are identified as well. This is done using human-made rules as well as machine learning. The rules for about 30 relations, with 10 to 20 rules per relation, make use of identified types. For example, from *a David Lean classic* the relation

directorOf can be extracted. The corresponding rule matches if a director (*David Lean*) is used as an adjective of a *film* synonym (*classic*). The machine-learning based approach uses distant supervision (explained in Section 4.3.2) to learn relation mentions for about 7K relations from DBpedia and Wikipedia. The identified relations are simple in the sense that they only connect one entity to a relation. This is in contrast to some of the techniques in Section 4.8 on *Question Answering on Knowledge Bases*, where the goal is a formal representation of the (meaning of the) whole question.

Finally, the question is also decomposed into subquestions, which can be answered independently. For example,

This company with origins dating back to 1876 became the first U.S. company to have 1 million stockholders in 1951.

contains two major hints: that the company has "origins dating back to 1876" and that it was "the first U.S. company to have 1 million stockholders in 1951". The decomposition is done via rules on a syntactic parse of the sentence. Answers from different subquestions are synthesized at the end with a model that is specifically trained to combine the results (lists of entities) of individual subquestions.

Hypothesis generation: After analyzing the question, the system generates candidate answers by searching multiple data sources (text and knowledge bases) independently. The focus in this step is on recall, with the assumption that later steps can weed out incorrect candidates and improve precision. A correct candidate answer not generated in this step will lead to a wrong final answer.

For search in text, standard techniques for keyword search (see Section 4.1) are applied to find documents and passages that contain keywords of the question. Candidate answers are extracted, e.g., from the title of the documents and passages using named-entity recognition. The applied techniques are similar to those of state-of-the-art systems for question answering on text (see Section 4.7).

Knowledge bases are queried for entities that are related to those mentioned in the question. These serve as additional candidate answers. If relations were identified in the question, these are also used for querying the knowledge bases. For each pair of a single entity and relation

in the question, a SPARQL query is derived and executed to retrieve additional candidate answers. In total, this phase typically generates several hundreds of candidate answers.

Soft filtering: The list of candidates obtained from the steps so far is often very long. For performance reasons, the list is filtered using lightweight machine learning, for example, based on how well the lexical answer type matches the candidate. The idea is to weed out candidates that are easy to identify as unlikely answers. Only about 100 candidate answers remain.

Hypothesis and evidence scoring: Now, evidence is collected for each remaining candidate. For example, passages that mention the answer entity along the question keywords are retrieved. Structured data is also used, e.g., in geospatial and temporal reasoning. For example, in

This picturesque Moorish city lies about 60 miles northeast of Casablanca.

the latitude and longitude of *Casablanca* can be retrieved from DBpedia and compared to candidate answers and the identified relation *northeast*.

The retrieved evidence is passed to scorers that determine the degree to which the evidence supports the candidate answer. More than 50 different scorers are used in total. They range from relatively simple string matching (between the question and the retrieved passages), to learning-based reasoning (for example, on spatial or temporal distance). According to the authors, no single algorithm dominates, but it is the ensemble that makes a difference.

Merging and ranking: In a final step, answer candidates are merged and then ranked. Merging is necessary because candidates can have different surface forms but refer to the same entity, for example, *John F. Kennedy* and *J.F.K.* This can happen because entities are retrieved from many different sources (text, DBpedia, Freebase etc.) and no canonical entity representation is enforced before this merging step.

The answer candidates are then ranked, based on the previously computed evidence scores. The question type is also taken into account.

This is important, since, for example, different features are important for factoid questions and puzzle-type questions. Ranking is done via a machine learning framework that takes as input a set of candidate answers with their evidence scores and outputs a confidence score for each candidate that indicates whether it is the final correct answer. Candidates ranked by their confidence scores are the final output of Watson.

4.9.3 Other Systems

Joshi, Sawant, and Chakrabarti [2014] answer entity-oriented (telegraphic) keyword queries on a text linked to a knowledge base (ClueWeb + FACC, see Table 2.3). Telegraphic queries are abbreviated natural language queries, for example, *first american in space*.[26] Given a question, the system computes a score for all possible entities, e_a, as answer. For this, the question is first split into entity, target type, relation, and contextual (everything else) segments. Then, evidence is collected from the knowledge base or text. For example, for the entity identified in the question, e_q, the relation to a possible answer entity, e_a, is retrieved from the knowledge base. A score is computed (using a language model) indicating how well the relation segment of the question expresses this knowledge-base relation. Furthermore, a text index is queried for snippets mentioning both e_q and e_a, scored by an adaptation of BM25. The final ranking incorporates the scores above as well as further evidence, like answer type information and a likelihood for the segmentation (several are possible). Overall, this is similar to the systems described in Section 4.8, but with a full-text component added to the underlying search. The system is evaluated on adapted queries from TREC (Section 4.1.3), INEX (Section 4.5.3), and WebQuestions (Section 4.8.5). On the WebQuestions dataset it achieves an nDCG@10 of 47% compared to Sempre, an approach only utilizing the knowledge base (see Section 4.8), with 45%. On the TREC and INEX questions they achieve an nDCG@10 of 54% versus 25% with Sempre.

[26] See our discussion on the gray zone between keyword queries and natural language queries at the beginning of Section 2.2.

4.9.4 Benchmarks

The QALD series, described in Section 4.8 on *Question Answering on Knowledge Bases*, featured a *hybrid search* task in 2015. The benchmark contains ten hybrid questions, for example:

Who is the front man of the band that wrote Coffee & TV?

The correct answer requires the combination of triples with information from the textual description of the entity (both contained in DBpedia, see Table 2.2). For example, for the example question above, the information who is a front man is contained only in the text. Only five systems participated in this benchmark. The best F-measure of only 26% was achieved by the ISOFT system [Park et al., 2015].

The INEX series, described in Section 4.5.3 on *Keyword Search on Combined Data*, featured a Jeopardy! task in 2012 and 2013. However, participation was low, with only a single group in 2013.

5

Advanced Techniques used for Semantic Search

This section is about four more advanced aspects of semantic search: ranking, indexing, ontology matching and merging, and inference. They are advanced in the sense that powerful semantic search engines can be built with relatively simplistic solutions for these aspects. Indeed, this is the case for several state of the art systems and approaches from Section 4. However, when addressed properly, they can further boost result quality and/or performance.

5.1 Ranking

Many of the systems from our main Section 4 produce a list of entities as their result. In our descriptions so far, we have focused on how the set of result entities is retrieved from the respective data. In this section, we elaborate on the aspect of ranking these entities. We also (but not exclusively) include research on ranking techniques that have not been implemented as part of a full-fledged system.

The following subsections roughly correspond to the representation of the entities that should be ranked: entities associated with virtual documents (typically obtained from keyword search on combined

data; see Section 4.5), entities obtained from text linked to a knowledge base (typically obtained from keyword or semi-structured search on combined data; see Sections 4.5 and 4.6), interlinked entities (from a knowledge base or from semantic web data), and entities obtained from a structured or semi-structured search (as described in Section 4.2 and Section 4.6).

In the following, we assume basic knowledge about standard ranking techniques for document-centric keyword search on text, such as: BM25 scoring, language models, and PageRank.

5.1.1 Ranking of Entities Associated with (Virtual) Documents

A standard approach for entity search in heterogeneous data is to construct, for each entity, a virtual document consisting of (all or a selection of) text associated with the entity in the given data (typically: semantic-web data); see Section 4.5. A ranking can then be obtained by using standard techniques for keyword search in text, like BM25 or language models.

The original structure (provided by triples) does not necessarily have to be discarded. It can be preserved in a fielded index and by a ranking function like BM25F, which is an extension of BM25 by Zaragoza et al. [2004]. In comparison to standard BM25, BM25F computes a field-dependent normalized term frequency tf_f^* which, instead of document length and average document length, uses field length (l_f) and average field length ($avfl$). In addition, each field has its own "b-parameter" B_f.

$$tf_f^* := \frac{tf_f}{1 + B_f(\frac{l_f}{avfl} - 1)}$$

The final term pseudo-frequency, that is used in the BM25 formula, is then obtained as weighted sum (field weight W_f) over the values for each field:

$$tf^* = \sum_f tf_f^* \cdot W_f$$

Originally, this improves ranking keyword queries on text by accounting for document structure (for example, with fields like *title*, *ab-*

stract, and *body*), but this extension also applies to fields that originate from different triple predicates.

The effectiveness of BM25F for ad-hoc entity retrieval on RDF data is demonstrated by Blanco, Mika, and Vigna [2011]. Some predicates and domains are manually classified as important or unimportant (for example, *abstract* and *description* are important properties, *date* and *identifier* are unimportant). Everything not classified is treated neutrally. Important predicates have their own index field, which is then boosted in the ranking function by using a higher field weight W_f. Important domains are boosted in a separate step after the BM25F value is computed. Compared to vanilla BM25, this leads to 27% MAP instead of 18% and 48% nDCG instead of 39% on the benchmark from the SemSearch Challenge 2010 (see Section 4.5.3).

Neumayer, Balog, and Nørvåg [2012] show that creating language models from virtual documents can outperform the fielded approach above. An entity e is ranked by the probability $p(q|e)$ of generating the query q. Different possibilities for computing this model are suggested. An *unstructured* model estimates term probabilities from the virtual document of each entity. A *structured* model groups each entity's predicates (groups are attributes, names, incoming and outgoing relations) and computes a model for each group's virtual document. The final score for an entity is a linear interpolation of these models with manually chosen weights. Experiments on the benchmarks from the SemSearch Challenges 2010 and 2011 show that the unstructured model outperforms previous state of the art but is in turn outperformed by the structured model. The authors also suggest a *hierarchical* language model that is supposed to preserve the structure (i.e. predicates) associated with entities, but the model fails to improve on previous results.

Herzig et al. [2013] also rank entities (virtual documents) using language models (LM). The work addresses two problems: identifying entities that refer to the same real-world entity, and ranking for federated search (where results from multiple sources have to be combined). The LM for an entity consists of multiple standard LMs, one for each of its attributes. A similarity distance between two entities is com-

puted by the Jensen-Shannon divergence (JSD) between the LMs for attributes that both entities have in common. If this distance is below a threshold, two entities are considered the same. Ranking for federated search works as follows. The query is issued to the multiple sources and the ranked results are used to create a new virtual document. This virtual document consists of the contents of the virtual documents of each result entity (weighted by rank). Then, a language model for the query is computed from the virtual document, which serves as a form of pseudo-relevance feedback. All entities are ranked by the similarity (again using JSD) of their language model to that of the query. In a final step, identical entities are merged as determined by the procedure above or by explicit *sameAs* links.

5.1.2 Ranking of Entities from Text Linked to a Knowledge Base

Search on text linked to a knowledge base (see Sections 4.5 and 4.6) provides two sources of signals for ranking result entities: from the matching text, and from the entity's entry in the knowledge base. However, it is not obvious how they should be combined for maximum effectiveness.

Fang et al. [2009] provide a simple but effective ranking approach for keyword queries on text, which won the TREC Entity Track in 2009 (see Section 4.5.3). In the first step, answer candidates are extracted from results (using basic NER techniques as described in Section 3) of a query to Google. This establishes the link between the text and an entity's representation in the knowledge base. In addition, a supporting passage (for an occurrence in text: the sentence; for an occurrence in a table: elements from the same column, column header, and sentence preceding the table) is extracted for each entity. Entities are then ranked by the product of three relevance probabilities: of the containing document to the query, of the containing passage, and of the entity. Document relevance is computed using a standard language model. Passage relevance is computed as a sum of similarity scores (from WordNet) between all pairs of words in the passage and the query. Entity relevance is computed as the frequency of the first query keyword (which usually corresponds to the target type, see the example above) in the entity's list of Wikipedia categories.

Kaptein and Kamps [2013] perform keyword search on Wikipedia (where documents naturally correspond to entities) and additionally make use of target categories that restrict the set of relevant entities. For example, for the query *works by charles rennnie mackintosh*, answers should be restricted to *buildings and structures*. These target categories are either given as part of the query, or can be derived automatically from the result set of an issued keyword query. Instead of simply filtering the result entities by the given or determined category, the authors suggest using language models that are supposed to deal with the hierarchical structure of categories. Two standard language models are precomputed: one for each entity (that is, for its Wikipedia page), and one for each category (that is, for the text from all entities in that category). The final score is a weighted combination of the two language models (how well they model the query) and an additional pseudo-relevance feedback computed via links between the Wikipedia pages of the top results. Weights for the combination are chosen manually.

Schuhmacher, Dietz, and Ponzetto [2015] adapt the learning-to-rank approach to keyword entity search on text, with entities already linked to a knowledge base. For a given query, the entities from the (top) documents matching the keyword query are retrieved and a feature vector is constructed for each query-entity pair. There are two groups of features: text features (for example, the occurrence count of the entity in text) and query features (for example, does the entity, or an entity closely connected in the knowledge base, occur in the query). The training set consists of queries with a given set of relevant entities. A support vector machine is used for learning to rank. It uses a linear kernel that is enhanced with a function to compute the similarity between two entities via their *relatedness* in the given knowledge base. This is supposed to provide a form of pseudo-relevance feedback. The approach is evaluated on an own benchmark that is constructed from the TREC Robust and Web benchmarks (see Section 4.1.3).

5.1.3 Ranking of Interlinked Entities

A knowledge base, or a collection of interlinked knowledge bases as in semantic-web data, can be viewed as a graph with the entities as nodes and the edges as relations; see Section 2.1.2 on *Knowledge Bases*.

Swoogle [Ding et al., 2004] adapts PageRank, the well-known algorithm to compute query-independent importance scores for web pages, to semantic-web data. Links between so-called *semantic web documents* (RDF documents defining one or more entities) are weighted differently depending on their type. Swoogle classifies links into four categories (*imports*, *uses-term*, *extends*, and *asserts*). This is done by manually defining which original RDF properties belong to which category. For each of these types, a weight is assigned manually. The PageRank transition probability from node i to node j then depends on the sum of weights of all links from i to j. This approach is feasible because only a relatively small number of different link types is considered.

ObjectRank [Balmin, Hristidis, and Papakonstantinou, 2004] adapts PageRank to keyword search on databases. The computed scores depend on the query. Intuitively, a random surfer starts at a database object that matches the keyword and then follows links pertaining to foreign keys. Edge weights are, again, based on types and assigned manually. For example, in a bibliographic database, citations are followed with high probability. Like this, the approach allows relevant objects to be found even if they do not directly mention the query keyword.

Agarwal, Chakrabarti, and Aggarwal [2006] combine PageRank with the learning to rank approach. The input is a knowledge graph (think of the semantic web) and a partial preference relation on the set of entities (think of a user more interested in some entities than in others). The goal is to learn edge weights such that the scores computed by the PageRank process (with transition probabilities proportional to these edge weights) reflect the given user preferences. Two scenarios are considered: individual weights for each edge, and one weight per edge type (predicate). For the first scenario, the problem is formulated as a constrained flow optimization problem, where the constraints come from the user preferences. For the second scenario, the problem is solved using gradient descent optimization, where the loss function

captures the user preferences (approximately only, so that it becomes differentiable).

TripleRank [Franz et al., 2009] extends the HITS algorithm to semantic-web data. HITS is a variant of PageRank, which computes hub and authority scores for each node of a sub-graph constructed from the given query. For TripleRank, the subgraph is represented as a 3D tensor where each slice is the (entity-entity) adjacency matrix for one predicate. Standard 3D tensor decomposition then yields n principal factors (corresponding to the singular values in the 2D case) and three 2D matrices with n columns each. One of these matrices can be interpreted as the underlying "topics" of the subgraph (expressed in terms of relations). The entries in the other two matrices can be interpreted as hub and authority scores, respectively, of each entity in the subgraph with respect to the identified topics.

5.1.4 Ranking of Entities Obtained from a Knowledge Base Search

SPARQL queries have precise semantics and, like SQL, the language provides an ORDER BY attribute for an explicit ranking of the result set; see Section 4.2 on *Structured Search in Knowledge Bases*. Still, there are scenarios, where a ranking according to "relevance", as we know it from text search, is desirable.

Elbassuoni et al. [2009] construct language models for SPARQL queries with support for keyword matching in literals. The language model for the query is defined as a probability distribution over triples from the knowledge base that match triples from the query. The language model for a result graph is straightforward: it has probability 1 or 0 for each triple, depending on whether that triple is present in the result graph (with some smoothing). Results are then ranked by their Kullback-Leibler (KL) divergence to the query.

Broccoli [Bast et al., 2012] ranks result entities using a combination of popularity scores for entities and frequency scores obtained from its interactive query suggestions. For example, a simple query for *Scientist* simply ranks all scientists in the indexed knowledge base by their popularity. But the query *Scientist occurs-with information retrieval* ranks scientist according to how frequently they co-occur with the words *in-*

formation retrieval in the given text collection. Suggestions are ranked in a similar way. For example, the suggestions for predicates for *Scientist* are ranked by how many scientists have that particular predicate. This simple ranking provided average precision at 10 and MAP scores of 67-81% and 42-44%, respectively, on two benchmarks (TREC Entity Track 2009 and Wikipedia).

Bast, Buchhold, and Haussmann [2015] present an approach to compute relevance scores for triples from type-like relations. Such a score measures the degree to which an entity "belongs" to a type. For example, one would say that Quentin Tarantino is more of a film director or screenwriter than an actor. Such scores are essential in the ranking of entity queries, e.g., "american actors" or "quentin tarantino professions". To compute the scores, each entity and type is associated with text. The text for entities is derived via linking to their occurrences in Wikipedia. Text for entire types is derived from entities that have only one entry in the particular relation. For the example above, text for the *profession actor* is derived from entities that only have the profession *actor*. Scores are then computed by comparing the text for an entity to that for each type. For this, many different models are considered: standard machine learning, a weighted sum of terms based on their tf-idf values, and a generative model. The best models achieve about 80% accuracy on a benchmark where human judges were able to achieve 90% and sensible baselines scored around 60%.

5.2 Indexing

Most of the work in this survey is concerned with the quality aspect of semantic search. This section is concerned with the efficiency aspect. Note that indirectly, efficiency is also relevant for quality: a method with good result quality with a response time of minutes or worse is impractical in many application scenarios.

Following our classification in Section 4, semantic indexing works differently depending on the particular approach: Keyword search on text (Section 4.1) is handled by an inverted index. The inverted index is a well-researched data structure and important for information retrieval

in general. A discussion of its particularities is beyond the scope of this survey. Special indexes for structured search in knowledge bases are already discussed at length in Section 4.2. Section 4.3 is concerned with structured data extraction from text. Indexing is not an issue here. Systems for keyword search on knowledge bases (Section 4.4) and question answering (Sections 4.7, 4.8, and 4.9) are concerned with finding the right queries and post-processing results. The way data is indexed is adopted from other approaches. This leaves Sections 4.5 on *Keyword Search on Combined Data* and 4.6 on *Semi-Structured Search on Combined Data* where advanced indexing techniques are required.

In this section, we distinguish three basic approaches used by the systems from that section: using an inverted index for knowledge base data, semi-structured search based on an inverted index, and integrating keyword search into a knowledge base.

5.2.1 Using an Inverted Index for Knowledge Base Data

In Section 4.2 on *Structured Search in Knowledge Bases* we discussed indexing techniques for full SPARQL support. However, semantic web applications often have different requirements: (1) the data is extremely heterogeneous, so that queries with anything but the simplest of semantics are pointless; (2) predicate and object names can be very verbose, so that keyword matching (only an optional add-on for a SPARQL engine) is a must; (3) the data volume is large, so that speed is of primary concern. In such a scenario, the inverted index provides simple solutions with high efficiency.

In the simplest realization, a *virtual document* is constructed for each entity, consisting of (all or a subset of) the words from the triples with that entity as subject; see Section 4.5.1. A standard inverted index on these virtual documents then enables keyword queries which return ranked lists of entities. A typical system in this vein is Semplore [Wang et al., 2009].

A more advanced system is described by Blanco, Mika, and Vigna [2011]. They study three variants of a fielded index, implemented using MG4J (see Section 4.1.2). The variants have different trade-offs between query time, query expressibility, and result quality. In the basic

variant, there are different fields for subject, predicate and object of a triple and positional information is used to align items from the same original triple. This allows keyword matches for object and predicate names (e.g., find triples where the predicate matches *author*) at the price of a larger query time compared to vanilla BM25 indexing. In an alternative variant, there is a field for each distinct predicate. This still allows to restrict matches to a certain predicate (e.g., *foaf:author*) but keyword matches for predicates are no longer possible. In a refined variant, predicates are grouped by importance into three classes, with one field per class. This supports only keyword queries (without any structure, like in the basic approach from the previous paragraph), but with query times similar to vanilla BM25 indexing. Result quality is vastly improved due to consideration of those fields in the ranking function: 27% MAP instead of 18% and 48% nDCG instead of 39% on the benchmark from the SemSearch Challenge 2010 (see Section 4.5.3).

SIREn [Delbru, Campinas, and Tummarello, 2012] is built on Lucene and supports keywords queries that correspond to star-shaped SPARQL queries (with one entity at the center), where predicate and relation names can be matched via keyword queries. There are inverted lists for words in predicate names and for words in object names. Each index item contains information about the triple to which it belongs, namely: the id of the subject entity, the id of the predicate, the id of the object (only for words in object names), and the position of the word in the predicate or object. Standard inverted list operations can then be used to answer a query for all entities from triples containing, e.g., *author* in the predicate name, and *john* and *doe* in the object name. As of this writing, the software is available as open source[1].

5.2.2 Semi-Structured Search Based on an Inverted Index

This is the method of choice for semi-structured search on combined data, as described in Section 4.6. Often, an inverted index is combined with techniques or even off-the-shelf software for indexing knowledge bases, such as Virtuoso. However, the extra effort to achieve an efficient combination usually happens on the side of the inverted index.

[1] `https://github.com/rdelbru/SIREn`

In the most basic realization, for each occurrence of an entity from the knowledge base in the text (for example, *... Obama ...*), we add an artificial word to the text (for example, *entity:Barack_Obama*). The inverted list for *entity:Barack_Obama* then contains all occurrences of this entity in the text. Standard inverted list operations enable queries such as *entity:Barack_Obama audience pope* (documents mentioning him in the context of an audience with the pope).

We next discuss two simple options to enhance the query expressiveness for this approach, by not just allowing concrete entities in the query, but semantic concepts ranging from types to arbitrary SPARQL queries.

One option, that is taken by KIM [Popov et al., 2004], is to compute the result for the knowledge base parts using a standard SPARQL engine, and to add this to the keyword query as a disjunction of all the result entities. This is simple but very inefficient when the number of result entities is large. Another option, that is taken by Mìmir [Tablan et al., 2015], is to add further artificial words to the index, which allow direct processing of more complex queries without resorting to the SPARQL engine. For example, if in the example above we also add the artificial word *type:politician* to the index, we could efficiently answer queries such as *type:politician audience pope* (passages mentioning a politician in the context of an audience with the pope). This works for simple queries, but does not allow complex SPARQL queries. In this case, Mìmir falls back to the inefficient KIM approach.

ESTER [Bast et al., 2007] solves this dilemma by adding artificial *entity:...* and selected *type:...* words to the inverted index (just like in the example above) and resorting to *joins* for all the remaining queries. These joins require additional information in the index lists: triples from the knowledge base are inserted into canonical documents for each entity. Join operations on the *entity:...* words are needed to use this information for matches outside of this canonical document. Therefore, items in the inverted lists have to contain a word id in addition to the usual document id, position, and score. However, using standard compression techniques, the index size is comparable to that of an ordinary inverted index, despite this addition.

All the approaches described so far share the major drawback that the result is inherently a list of document passages. For example, the keyword query *type:politician audience pope* yields a list of matching passages, possibly many of them with the same entity. From a usability perspective, the more natural result would be a list of entities (ideally, with the passages as result snippets). Worse than that, if this query appears as a sub-query of a more complex query (e.g., looking for entities of a certain type who co-occur with the result entities), we need the list of entities (and not matching passages) to be able to process that query.

Broccoli [Bast and Buchhold, 2013] solves this problem using a non-trivial extension of the inverted index. The main technical idea is to augment the inverted list for each word by information about the co-occurring entities. For example, for the occurrence of the word *edible* in a document containing *the stalks of rhubarb and broccoli are edible*, the inverted list for *edible* would not only contain one item with the id of that document (plus score and positional information) but also two additional items with the ids for the entities *rhubarb* and *broccoli*. Each entity occurrence hence leads to an extra item in all inverted lists that have an entry for that document. Broccoli avoids a blow-up of the index by indexing semantic contexts instead of whole documents, which at the same time improves search quality (see Section 4.6.2). The knowledge base part is handled by lists of id pairs for each relation, sorted by either side. This is reminiscent of using PSO (predicate-subject-object) and POS permutations, like for the systems from Section 4.2 on *Structured Search on Knowledge Bases*. Together, the extended inverted lists and relation permutations allow that knowledge base facts and textual co-occurrence can be nested arbitrarily in tree-shaped queries while retaining very fast query times.

5.2.3 Integrating Keyword Search into a Knowledge Base

All major SPARQL engines feature integrations of keyword search; see Section 4.2.1. There are two basic variants, depending on the desired semantics of the integration.

To realize something like Virtuoso's *bif:contains* predicate (see Section 4.2), it suffices to pre-compute inverted lists for words in predicate names and objects. Standard inverted list operations then lead to lists of (ids of) predicates or objects, which can be processed further by the SPARQL engine. Compared to the approach described for SIREn above, the expressiveness is much larger (no longer restricted to star queries). The price is a much larger processing time for some queries. For example, the query *author john doe* requires a full scan over all triples using the approach just described. The reason is that both, the predicate and object part can match many items and that these do not correspond to ranges but lists of ids. In the example, many ids may match the keyword *author* and many ids may match *john doe*. While these individual lists of ids are both efficiently retrieved, a subsequent step towards matching triples is problematic.

To realize an index for text linked to a knowledge base, one could add an id for each document (or short passage) to the knowledge base, and add a special relation *occurs-in* (between words or entities and the id of the document they occur in). This covers the expressiveness of Broccoli, but with a much larger processing time. For example, the query *type:politician audience pope* requires a full scan over all triples of the *occurs-in* relation. Furthermore, such a relation becomes huge with larger text corpora because it contains an entry for each word occurrence in the collection. Note that adding a relation *occurs-with* between word and entity occurrences instead, doesn't provide the same semantics. This doesn't allow restricting multiple occurrences to the same document or context.

5.3 Ontology Matching and Merging

Most semantic search systems work with some kind of knowledge base, in particular, all the systems from Sections 4.2, 4.4, 4.5, 4.6, 4.8, and 4.9. Most of these systems assume a *single* knowledge base with a consistent schema/ontology, as defined in Section 2.1.2. However, to cover the data relevant for a given application, often several different knowledge bases need to be considered.

For example, think of an application that requires knowledge on movies as well as on books, and that there is a separate knowledge base for each of the two domains. A problem is that these knowledge bases may contain different representations of the same real-world entity. For example, *Stephen King* is likely to be present as a *novelist* in the book knowledge base and as a *screenwriter* in the movie knowledge base. To make proper use of the data, their ontologies (their classes/concepts, properties, relations) as well as their actual population (instances) should either be linked or merged. This means, for example, identifying links between identical persons, such as:

```
<movies:Stephen_King> <owl:sameAs> <books:Stephen_Edwin_King>
```

This is known as *instance matching*. Identifying links between classes, which is referred to as *ontology matching*, is also important in that context. For example, a script is a kind of written work:

```
<movies:Filmscript> <rdfs:subClassOf> <books:Written_Work>
```

Such links can be used to merge the ontologies into a single ontology in a pre-processing step. This is known as *ontology merging*. Alternatively, systems like Virtuoso, Jena, and Sesame (see Section 4.2.2) can be configured to make use of such links during query time.

These problems have been studied (with minor differences) mainly by the semantic web community and the database community. In a relational database, tables and their columns and data types make up the schema, analogously, to the defined classes, properties, and relations in an ontology. A database row or record is the equivalent of an instance in a knowledge base. Both communities make a distinction between matching schemata and matching actual instances or database records. Approaches to these tasks are very similar in both communities, so we provide corresponding pointers for further reading. In the following, we describe the general ideas used to tackle the problems. We focus on methods, that target automatic solutions. In practice, most systems integrate user feedback on some level.

5.3.1 Ontology Matching

Matching two ontologies means to determine an alignment between them. An alignment is a set of correspondences between uniquely identified elements (e.g., classes and properties) that specifies the kind of relation they are in. For example, whether two classes are equivalent, or one subsumes the other. Shvaiko and Euzenat [2013] provide a good survey on ontology matching. The database community refers to this problem as data matching. We refer to Doan and Halevy [2005] for a good survey on database related approaches.

Approaches: Approaches to ontology matching mainly make use of matching strategies that use terminological and structural data [Shvaiko and Euzenat, 2013]. Terminological data refers to string similarities of, e.g., labels and comments in the ontology. The idea is that highly similar names can indicate equivalence. Relationships between classes (e.g., part-of, is-a) make up structural data. The intuition is that classes in similar positions in the class hierarchy are more likely to be the same. In addition to that, some approaches make use of the actual instances of a knowledge base, or try to perform logical reasoning. The output of the different matchers is combined using pre-defined or learned weights to derive a decision.

Benchmarks: In 2004, the Ontology Alignment Evaluation Initiative (OAEI) started an annual benchmark to evaluate ontology matching systems [Euzenat et al., 2011a]. Each year, a set of benchmarking datasets that include reference alignments is published.[2] Participation was low in the beginning but since 2007 on average 17 groups participate with a peak of 23 groups in 2013. Systems can compete against each other and compare results. On a real world ontology matching task, systems have shown to give results with above 90% F-measure [Grau et al., 2013]. Shvaiko and Euzenat [2013] note that while great progress was made in the first years, progress is slowing down. They formulate a set of eleven major challenges that need to be tackled in the near future, in particular, more efficient matching and matching utilizing background knowledge.

[2] `http://oaei.ontologymatching.org/`

5.3.2 Instance Matching

Instance matching refers to finding instances that represent the same individual or entity. In the Semantic Web, these are linked using *owl:sameAs*. In the database community, this is referred to as record linkage, duplicate record identification or detection, and entity matching (and some more). A lot of research on this problem has been in done in that community . We refer to the surveys by Köpcke and Rahm [2010] and by Elmagarmid, Ipeirotis, and Verykios [2007]. Most of the approaches for instance matching are minor adaptations from those for databases [Castano et al., 2011].

Approaches: Similar to ontology matching, to match two instances, their attribute values are compared. This involves using string similarity (e.g., edit distance and extensions, and common q-grams), phonetic similarity (similar sounding field names are similar, even if they are spelled differently) or numerical similarity (difference) depending on the data type. Then, learning based techniques represent such an instance tuple as a feature vector of similarities and use a binary classifier. If no learning data is available, manually derived weights and thresholds can be used. Extensions of these methods also consider relationships to other instances, apply unsupervised learning techniques, or apply additional rules based on domain knowledge.

Benchmarks: The benchmark by the Ontology Alignment Evaluation Initiative (OAEI) contains several instance matching tasks since 2009. Different knowledge bases are provided for which identical entities need to be identified. The used knowledge bases consist of parts of real-world knowledge bases like DBpedia [2007] or Freebase [2007]. Systems have shown to provide up to 90% F-Measure on identifying identical instances in these [Euzenat et al., 2010; Euzenat et al., 2011b].

5.3.3 Ontology Merging

Instead of using several inter-linked knowledge bases, it may be desirable to merge them into a single coherent knowledge base. Merging these involves merging their schema/ontology (concepts, relations etc.) as well as merging duplicate instances. This requires, for example, re-

solving conflicting names and attribute values. Merging can be done in an *asymmetric* fashion, where one or more ontologies are integrated into a target ontology. In contrast, *symmetric* merging places equal importance on all input ontologies.

Approaches: Most work in the research so far has focused on computing alignments between ontologies. Bruijn et al. [2006] and Shvaiko and Euzenat [2013] describe some approaches that perform merging of ontologies. These usually take computed alignments between the ontologies as input and perform semi-automatic merging. For example, PROMPT [Noy and Musen, 2000] performs merging by iteratively suggesting merge operations based on heuristics to the user, applying the user-selected operation, and computing the next possible merge operations. The fact that many systems are semi-automatic makes it extremely hard to compare their performance and currently no widely accepted benchmark exists.

The problem of merging actual instances has not received much attention by the semantic web community. It has, however, been extensively studied by the database community. Bleiholder and Naumann [2008] give a good overview and present some systems on *data fusion*. Strategies for resolving conflicts, such as different values for the same attribute, are mainly rule based. Some common strategies are, for example, asking the user what to do, using values from a preferred source, or using the newest or average value.

5.4 Inference

Inference (or reasoning) means deriving information that is not directly in the data, but can be inferred from it. For example, from the facts that *Marion Moon* is an ancestor of *Buzz Aldrin* and that *Buzz Aldrin* is an ancestor of *Janice Aldrin*, one can infer that *Marion Moon* is also an ancestor of *Janice Aldrin*.

Surprisingly, only few systems make use of inference as an integral part of their approach to semantic search. One of the few examples is the question answering system by Lymba [Moldovan, Clark, and Bowden, 2007], which uses a reasoning engine in its answering process (see

Section 4.7.3 on *The PowerAnswer System*). This is in line with the survey by Prager [2006], who observes the same for question answering. We suppose that the reason for this is that current systems already struggle solving lower level problems, such as information extraction and translating a query into a formal representation (semantic parsing). These are, however, prerequisites for utilizing inference (let alone benefiting from it). Nonetheless, inference will certainly play a more important role in the future. As a result, here we focus on technical standards and components that enable researchers to perform inference.

A lot of triple stores include an inference engine. In addition to triples, these require as input a set of inference rules, for example, that the facts *A is ancestor of B*, and *B is ancestor C* imply that *A is ancestor of C*. First, we introduce some languages that can be used to express these rules. We then describe some triple stores and engines (also referred to as reasoners) that allow inference over triples.

5.4.1 Languages

We first describe languages that are mainly used to describe the schema of an ontology. These allow expressing constraints for the facts of a knowledge base, for example, that a child cannot be born before its parents. This also allows inference, but, broadly speaking, with a focus on taxonomic problems.

RDF Schema [RDFS, 2008] is an extension of the basic RDF vocabulary. RDFS defines a set of classes and properties expressed in RDF, that provides basic features for describing the schema of an ontology. For example, using the RDFS elements *rdfs:Class* and *rdfs:subClassOf* allows declaring a hierarchy of classes. This allows inferring missing information, such as deriving missing class memberships based on the defined class hierarchy. For example, one might derive that Buzz Aldrin is a person from the fact that he is an astronaut and the definition that astronaut is a sub-class of person.

The Web Ontology Language [OWL, 2004] is a family of languages (OWL Lite, OWL DL, OWL Full) with different levels of expressiveness to describe ontologies. OWL is the successor of DAML+OIL. Like RDFS, OWL is used to describe the schema and semantics of an ontol-

ogy, but with a much larger vocabulary and more options. For example, it allows defining class equivalences and cardinality of predicates. A prominent artifact of OWL is the *owl:sameAs* predicate, which is used to link identical instances. OWL also allows expressing transitive and inverse relationships enabling more complex inference.

The OWL 2 Web Ontology Language [OWL 2, 2012] is the successor of OWL. It extends OWL (and is backwards compatible) by addressing some shortcomings in expressiveness, syntax, and other issues [Grau et al., 2008]. Like OWL, it consists of a family of sub-languages (OWL 2 EL, OWL 2 QL, OWL 2 RL) also called profiles. These trade some of their expressive power for more efficient reasoning and inference. OWL 2 RL and QL are considered appropriate for inference with large volumes of data.

Next, we describe three prominent languages, whose single purpose is the description of inference rules. They allow expressing rules that are either hard or impossible to define in the languages above.

The Rule Markup Language [RuleML, 2001] was defined by the Rule Markup Initiative, a non-profit organization with members of academia and industry. It is XML based and (in contrast to many other languages) allows reasoning over n-ary relations. RuleML has provided input to SWRL as well as RIF (see below).

The Semantic Web Rule Language [SWRL, 2004] uses a subset of OWL DL and RuleML. It extends the syntax of OWL but is more expressive than OWL alone.

The Rule Interchange Format [RIF, 2010] was designed primarily to facilitate rule exchange. It consists of three different dialects: Core, Basic Logic Dialect (BLD), and Production Rule Dialect (PRD). RIF is an XML language and specified to be compatible with OWL and RDF. It also covers most features of SWRL.

5.4.2 Inference Engines

Most triple stores or RDF frameworks include a reasoning component. We introduce a few prominent examples and describe their features.

A widely used benchmark for evaluating OWL reasoners is the Leigh University Benchmark (LUBM) [Guo, Pan, and Heflin, 2005].

It measures, besides other things, performance and soundness of inference capabilities. The University Ontology Benchmark (UOBM) is an extension thereof, focusing on a better evaluation of inference and scalability [Ma et al., 2006]. Because results change frequently with different hardware and software versions, we don't provide them here. Current results are usually available via, e.g., `http://www.w3.org/wiki/RdfStoreBenchmarking` or the provider of the triple store.

We already introduced the triple stores of Virtuoso, Jena, and Sesame in Section 4.2 on *Structured Search in Knowledge Bases*. Virtuoso also includes an inference engine that is able to reason on a subset of the OWL standard (e.g., *owl:sameAs*, *owl:subClassOf*).

The triple store of Jena, TDB, also supports OWL and RDFS ontologies. It comes with a set of inference engines for RDFS and OWL definitions as well as custom rules. An API allows integration of third-party or custom reasoners.

Sesame includes a memory and disk-based RDF store. It provides inference engines for RDFS and custom rules. Additional reasoners can be integrated via an API.

GraphDB[3], formerly OWLIM, is a product by OntoText, also available as a free version. It includes a triple store, an inference engine, and a SPARQL query engine. GraphDB can be plugged into Sesame to provide a storage and inference back end. It can reason over RDFS, (most of) SWRL, and several OWL dialects (including OWL-2 RL).

Pellet [Sirin et al., 2007] is an open source OWL 2 reasoner written in Java. It can be integrated with different Frameworks, for example, Apache Jena. Pellet supports OWL 2 as well as SWRL rules.

OpenCyc[4] is a knowledge base platform developed by Cyc. It provides access to an ontology containing common sense knowledge and includes an inference engine. The inference engine is able to perform general logical deduction. OpenCyc uses its own rule language CycL, which is based on first-order logic.

[3] `http://www.ontotext.com/products/ontotext-graphdb/`

[4] `http://www.cyc.com/platform/opencyc/`

6

The Future of Semantic Search

In this final section, let us briefly review the present state of the art in semantic search, and let us then dare to take a look into the near and not so near future. Naturally, the further we look into the future, the more speculative we are. Time will tell how far we were off.

6.1 The Present

Let us quickly review the best results on the latest benchmarks from the various subsections of our main Section 4: an nDCG@20 of about 30% for keyword search on a large web-scale corpus (Section 4.1); an F1 score of around 40% for filling the slots of a given knowledge base from a given text corpus (Section 4.3); a MAP score of around 25% for keyword search on a large knowledge base (Section 4.4); an nDCGD@R of about 30% for entity keyword search on a large web-scale corpus (Section 4.5); an nDCG of about 50% for semi-structured search on an annotated Wikipedia corpus (Section 4.6); an F1 score of around 50% for natural-language list questions on text (Section 4.7); an F1 score of around 50% for natural language questions on a knowledge base (Section 4.8); and a similar score for the same kind of questions on combined data (Section 4.9).

The results for the basic NLP benchmarks are in a similar league (considering that the tasks are simpler): an F1 score of around 75% for large-scale named-entity recognition and disambiguation (Section 3.2), a weighted F1 score (called Freval) of around 55% for sentence parsing (Section 3.3), and an accuracy of about 70% for word analogy using word vectors (Section 3.4).

These are solid results on a wide array of complex tasks, based on decades of intensive research. But they are far from perfect. In particular, they are far from what humans could achieve with their level of understanding, if they had sufficient time to search or process the data.

Let us look at three state-of-the-art systems for the scenarios from the last three subsections of Section 4: PowerAnswer (Section 4.7), Wolfram Alpha (Section 4.8), and Watson (Section 4.9). It is no coincidence that these systems share the following characteristics: (1) the main components are based on standard techniques that have been known for a long time, (2) the components are very carefully engineered and combined together, and (3) the "intelligence" of the system lies in the selection of these components and their careful engineering and combination.[1] It appears that using the latest invention for a particular sub-problem does not necessarily improve the overall quality for a complex task; very careful engineering is more important. See also the critical discussion at the end of Section 4.1.

The Semantic Web, envisioned fifteen years ago, now exists, but plays a rather marginal role in semantic search so far. It is employed in some very useful basic services, like an e-commerce site telling a search robot about the basic features of its products in a structured way. But the Semantic Web is nowhere near its envisioned potential (of providing explicit semantic information for a representative portion of the Web). Results on benchmarks that use real semantic web data are among the weakest reported in our main Section 4.

Machine learning plays an important role in making the current systems better, mostly by learning the best combination of features

[1] The same can probably be said about a search engine like Google. The details are not published, but telling from the search experience this is very likely so.

(many of which have been used in rule-based or manually tuned systems before) automatically. Important sources of training data are past user interactions (in particular, clickthrough data), crowdsourcing (to produce larger amounts of training data explicitly, using a large human task force), and huge unlabelled text corpora (which can be used for distant supervision). The results achieved with these approaches consistently outperform previous rule-based or manually tuned approaches by a few percent, but also not more. Also note that this is a relatively simple kind of learning, compared to what is probably needed to "break the barrier", as discussed in Section 6.3 below.

Interestingly, modern web search engines like Google provide a much better user experience than what is suggested by the rather mediocre figures summarized above. In fact, web search has improved dramatically over the last fifteen years. We see three major reasons for this. First, the user experience in web search is mainly a matter of high precision, whereas the results above consider recall as well. Second, web search engines have steadily picked up and engineered to perfection the standard techniques over the years (including basic techniques like error correction, but also advanced techniques like learning from clickthrough data, which especially helps popular queries). Third, a rather trivial but major contributing factor is the vastly increased amount of content. The number of web pages indexed by Google has increased from 1 billion in 2000 to an estimated 50 billion in 2015 (selected from over 1 trillion URLs). For many questions that humans have, there is now a website with an answer to that question or a slight variant of it, for example: Stack Overflow (programming) or Quora (general questions about life). Social platforms like Twitter or Reddit provide enormous amounts of informative contents, too.

6.2 The Near Future

Over the next years, semantic search (along the lines of the systems we have described in Section 4) will mature further. The already large amount of text will grow steadily. The amount of data in knowledge bases will grow a lot compared to now.

Knowledge bases will be fed more and more with structured data extracted from the ever-growing amount of text. The basic techniques will be essentially those described in Section 4.3, but elaborated further, applied more intelligently, and on more and more data with faster and faster machines. This extraction will be driven by learning-based methods, based on the basic NLP methods explained in Section 3. Data from user interaction will continue to provide valuable training data. Data from the Semantic Web might provide important training information, too (either directly or via distant supervision).

The combination of information from text and from knowledge bases will become more important. The current state of the art in systems like Watson or Google Search is that the text and the knowledge base are processed in separate subsystems (often with the knowledge base being the junior partner), which are then combined post hoc in a rather simple way. The two data types, and hence also the systems using them, will grow together more and more. Large-scale annotation datasets like FACC (see Table 2.3) and the systems described in Section 4.6 on *Semi-Structured Search on Combined Data* already go in that direction. The meager contents of Section 4.9 on *Question Answering on Combined Data* show that research in this area is only just beginning. We will see much more in this direction, in research as well as in the large commercial systems.

6.3 The Not So Near Future

The development as described so far is bound to hit a barrier. That barrier is an actual *understanding* of the meaning of the information that is being sought. We said in our introduction that semantic search is search with meaning. But somewhat ironically, all the techniques that are in use today (and which we described in this survey) merely simulate an understanding of this meaning, and they simulate it rather primitively.

One might hope that with a more and more refined such "simulation", systems based on such techniques might converge towards something that could be called real understanding. But that is not how progress has turned out in other application areas, notably: speech

recognition (given the raw audio signal, decode the words that were uttered), image classification (given the raw pixels of an image, recognize the objects in it), and game play (beat Lee Sedol, a grandmaster of Go). Past research in all these areas was characterized by approaches that more or less explicitly "simulate" human strategy, and in all these approaches eventually major progress was made by deep neural networks that learned good "strategies" themselves, using only low-level features, a large number of training examples, and an even larger number of self-generated training examples (via distant supervision on huge amounts of unlabelled data or some sort of "self play").

Natural language processing will have to come to a similar point, where machines can compute rich semantic representations of a given text themselves, without an explicit strategy prescribed by humans. It seems that the defining characteristics of such representations are clear already now: (1) they have to be so rich as to capture all the facets of meaning in a human sense (that is, not just POS tags and entities and grammar, but also what the whole text is actually "about"); (2) they have to be hierarchical, with the lower levels of these representations being useful (and learnable) across many diverse tasks, and the higher levels building on the lower ones; (3) they are relatively easy to use, but impossible to understand in a way that a set of rules can be understood; (4) neither the representation nor the particular kind of hierarchy has to be similar to the representation and hierarchy used by human brains for the task of natural language processing.

The defining properties might be clear, but we are nowhere near building such systems yet. Natural language understanding is just so much more multifaceted than the problems above (speech recognition, image classification, game play). In particular, natural language is much more complex and requires a profound knowledge about the world on many levels (from very mundane to very abstract). A representation like word vectors (Section 3.4) seems to go in the right direction, but can at best be one component on the lowest level. The rest is basically still all missing.

Once we come near such self-learned rich semantic representations, the above-mentioned barrier will break and we will be converging to-

wards true semantic search. Eventually, we can then feed this survey into such a system and ask: *Has the goal described in the final section been achieved?* And the answer will not be: *I did not understand 'final section'.* But: *Yes, apparently ;-)*

Acknowledgements

We are very grateful to the three anonymous referees for their inspiring and thorough feedback, which has improved both the contents and the presentation of this survey considerably. And a very warm thank you to our non-anonymous editor, Doug Oard, for his infinite patience and tireless support on all levels.

Appendices

A Datasets

AQUAINT (2002). Linguistic Data Consortium, http://catalog.ldc.upenn.edu/LDC2002T31.

AQUAINT2 (2008). Linguistic Data Consortium, http://catalog.ldc.upenn.edu/LDC2008T25.

Blogs06 (2006). Introduced by [Macdonald and Ounis, 2006], http://ir.dcs.gla.ac.uk/test_collections/blog06info.html.

BTC (2009). Andreas Harth, the Billion Triples Challenge data set, http://km.aifb.kit.edu/projects/btc-2009.

BTC (2010). Andreas Harth, the Billion Triples Challenge data set, http://km.aifb.kit.edu/projects/btc-2010.

BTC (2012). Andreas Harth, the Billion Triples Challenge data set, http://km.aifb.kit.edu/projects/btc-2012.

ClueWeb (2009). Lemur Project, http://lemurproject.org/clueweb09.

ClueWeb (2012). Lemur Projekt, http://lemurproject.org/clueweb12.

ClueWeb09 FACC (2013). Evgeniy Gabrilovich, Michael Ringgaard, and Amarnag Subramanya. FACC1: Freebase annotation of ClueWeb corpora, Version 1 (Release date 2013-06-26, Format version 1, Correction level 0), http://lemurproject.org/clueweb09/FACC1.

ClueWeb12 FACC (2013). Evgeniy Gabrilovich, Michael Ringgaard, and Amarnag Subramanya. FACC1: Freebase annotation of ClueWeb corpora, Version 1 (Release date 2013-06-26, Format version 1, Correction level 0), http://lemurproject.org/clueweb12/FACC1.

CommonCrawl (2007). Retrieved from http://commoncrawl.org, December 2014.

CrossWikis (2012). Introduced by [Spitkovsky and Chang, 2012], http://nlp.stanford.edu/data/crosswikis-data.tar.bz2.

DBpedia (2007). Introduced by [Lehmann, Isele, Jakob, Jentzsch, Kontokostas, Mendes, Hellmann, Morsey, Kleef, Auer, and Bizer, 2015]. Retrieved from http://dbpedia.org, Feburary 2014. Statistics taken from http://wiki.dbpedia.org/about, January 2016.

FAKBA1 (2015). Jeffrey Dalton, John R. Frank, Evgeniy Gabrilovich, Michael Ringgaard, and Amarnag Subramanya. FAKBA1: Freebase annotation of TREC KBA Stream Corpus, Version 1 (Release date 2015-01-26, Format version 1, Correction level 0), http://trec-kba.org/data/fakba1.

Freebase (2007). Introduced by [Bollacker, Evans, Paritosh, Sturge, and Taylor, 2008]. Retrieved from http://www.freebase.com, January 2016. Statistics taken from http://www.freebase.com, January 2016.

GeoNames (2006). Retrieved from http://www.geonames.org, December 2014. Statistics taken from http://www.geonames.org/ontology/documentation.html, January 2016.

MusicBrainz (2003). Retrieved from http://linkedbrainz.org/rdf/dumps/20150326/, March 2015. Statistics taken by counting items in the downloaded dataset.

Paralex (2013). Introduced by [Fader, Zettlemoyer, and Etzioni, 2013], http://openie.cs.washington.edu.

Patty (2013). Introduced by [Nakashole, Weikum, and Suchanek, 2012], http://www.mpi-inf.mpg.de/yago-naga/patty.

Penn Treebank-2 (1995). Introduced by [Marcus, Santorini, and Marcinkiewicz, 1993], https://catalog.ldc.upenn.edu/LDC95T7.

Penn Treebank-3 (1999). Introduced by [Marcus, Santorini, and Marcinkiewicz, 1993], https://catalog.ldc.upenn.edu/LDC99T42.

Stream Corpus (2014). TREC Knowledge Base Acceleration Track, http://trec-kba.org/kba-stream-corpus-2014.shtml.

UniProt (2003). Retrieved from http://www.uniprot.org/, July 2014. Statistics taken from http://sparql.uniprot.org/.well-known/void, January 2016.

WDC (2012). Retrieved from http://webdatacommons.org, November 2013.

Wikidata (2012). Retrieved from http://www.wikidata.org/wiki/Wikidata:Database_download, October 2014. Statistics taken from https://tools.wmflabs.org/wikidata-todo/stats.php, January 2016.

Wikipedia LOD (2012). Introduced by [Wang, Kamps, Camps, Marx, Schuth, Theobald, Gurajada, and Mishra, 2012], http://inex-lod.mpi-inf.mpg.de/2013.

YAGO (2007). Introduced by [Suchanek, Kasneci, and Weikum, 2007]. Retrieved from http://yago-knowledge.org, October 2009. Statistics taken from http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/archive/, January 2016.

YAGO2s (2011). Introduced by [Hoffart, Suchanek, Berberich, Lewis-Kelham, Melo, and Weikum, 2011]. Retrieved from http://yago-knowledge.org, December 2012. Statistics taken from [Hoffart, Suchanek, Berberich, and Weikum, 2013].

B Standards

FOAF (2000). Friend of a Friend, www.foaf-project.org.

OWL (2004). OWL Web Ontology Language, http://www.w3.org/TR/owl-features.

OWL 2 (2012). OWL 2 Web Ontology Language, http://www.w3.org/TR/owl2-overview.

R2RML (2012). RDB to RDF Mapping Language, Suite of W3C Recommendations, http://www.w3.org/TR/r2rml.

RDFS (2008). RDF Schema, http://www.w3.org/TR/rdf-schema.

RIF (2010). Rule Interchange Format, http://www.w3.org/TR/rif-overview.

RuleML (2001). Rule Markup Language, http://ruleml.org/1.0.

Schema.org (2011). Google, Yahoo, Microsoft, Yandex, http://schema.org.

SPARQL (2008). Query Language for RDF, W3C Recommendation, http://www.w3.org/TR/rdf-sparql-query.

SQL (1986). Structured Query Language.

SWRL (2004). A Semantic Web Rule Language, http://www.w3.org/Submission/SWRL.

References

Abadi, D., P. Boncz, S. Harizopoulus, S. Idreos, and S. Madden (2013). The design and implementation of modern column-oriented database systems. In: *Foundations and Trends in Databases* 5.3, pp. 197–280.

Agarwal, A., S. Chakrabarti, and S. Aggarwal (2006). Learning to rank networked entities. In: *KDD*, pp. 14–23.

Agrawal, S., S. Chaudhuri, and G. Das (2002). DBXplorer: enabling keyword search over relational databases. In: *SIGMOD*, p. 627.

Angeli, G., S. Gupta, M. Jose, C. D. Manning, C. Ré, J. Tibshirani, J. Y. Wu, S. Wu, and C. Zhang (2014). Stanford's 2014 slot filling systems. In: *TAC-KBP*.

Arasu, A. and H. Garcia-Molina (2003). Extracting structured data from web pages. In: *SIGMOD*, pp. 337–348.

Armstrong, T. G., A. Moffat, W. Webber, and J. Zobel (2009a). Has adhoc retrieval improved since 1994? In: *SIGIR*, pp. 692–693.

Armstrong, T. G., A. Moffat, W. Webber, and J. Zobel (2009b). Improvements that don't add up: ad-hoc retrieval results since 1998. In: *CIKM*, pp. 601–610.

Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives (2007). DBpedia: a nucleus for a web of open data. In: *ISWC/ASWC*, pp. 722–735.

Balakrishnan, S., A. Halevy, B. Harb, H. Lee, J. Madhavan, A. Rostamizadeh, W. Shen, K. Wilder, F. Wu, and C. Yu (2015). Applying WebTables in practice. In: *CIDR*.

Balmin, A., V. Hristidis, and Y. Papakonstantinou (2004). ObjectRank: authority-based keyword search in databases. In: *VLDB*, pp. 564–575.

Balog, K. and R. Neumayer (2013). A test collection for entity search in DBpedia. In: *SIGIR*, pp. 737–740.

Balog, K., P. Serdyukov, and A. P. de Vries (2010). Overview of the TREC 2010 Entity Track. In: *TREC*.

Balog, K., P. Serdyukov, and A. P. de Vries (2011). Overview of the TREC 2011 Entity Track. In: *TREC*.

Balog, K., A. P. de Vries, P. Serdyukov, P. Thomas, and T. Westerveld (2009). Overview of the TREC 2009 Entity Track. In: *TREC*.

Balog, K., Y. Fang, M. de Rijke, P. Serdyukov, and L. Si (2012). Expertise retrieval. In: *Foundations and Trends in Information Retrieval* 6.2-3, pp. 127–256.

Banko, M., M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni (2007). Open information extraction from the Web. In: *IJCAI*, pp. 2670–2676.

Bast, H. and I. Weber (2006). Type less, find more: fast autocompletion search with a succinct index. In: *SIGIR*, pp. 364–371.

Bast, H., A. Chitea, F. M. Suchanek, and I. Weber (2007). ESTER: efficient search on text, entities, and relations. In: *SIGIR*, pp. 671–678.

Bast, H. and B. Buchhold (2013). An index for efficient semantic full-text search. In: *CIKM*, pp. 369–378.

Bast, H., B. Buchhold, and E. Haussmann (2015). Relevance scores for triples from type-like relations. In: *SIGIR*, pp. 243–252.

Bast, H. and E. Haussmann (2013). Open information extraction via contextual sentence decomposition. In: *ICSC*, pp. 154–159.

Bast, H. and E. Haussmann (2014). More informative open information extraction via simple inference. In: *ECIR*, pp. 585–590.

Bast, H. and E. Haussmann (2015). More accurate question answering on Freebase. In: *CIKM*, pp. 1431–1440.

Bast, H., F. Bäurle, B. Buchhold, and E. Haussmann (2012). Broccoli: semantic full-text search at your fingertips. In: *CoRR* abs/1207.2615.

Bast, H., F. Bäurle, B. Buchhold, and E. Haußmann (2014a). Easy access to the Freebase dataset. In: *WWW*, pp. 95–98.

Bast, H., F. Bäurle, B. Buchhold, and E. Haußmann (2014b). Semantic full-text search with Broccoli. In: *SIGIR*, pp. 1265–1266.

Bastings, J. and K. Sima'an (2014). All fragments count in parser evaluation. In: *LREC*, pp. 78–82.

Berant, J. and P. Liang (2014). Semantic parsing via paraphrasing. In: *ACL*, pp. 1415–1425.

Berant, J., A. Chou, R. Frostig, and P. Liang (2013a). Semantic parsing on freebase from question-answer pairs. In: *EMNLP*, pp. 1533–1544.

Berant, J., A. Chou, R. Frostig, and P. Liang (2013b). The WebQuestions Benchmark. In: Introduced by [Berant, Chou, Frostig, and Liang, 2013a].

Bhalotia, G., A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan (2002). Keyword searching and browsing in databases using BANKS. In: *ICDE*, pp. 431–440.

Bizer, C. and A. Schultz (2009). The Berlin SPARQL Benchmark. In: *IJSWIS* 5.2, pp. 1–24.

Blanco, R., P. Mika, and S. Vigna (2011). Effective and efficient entity search in RDF data. In: *ISWC*, pp. 83–97.

Blanco, R., H. Halpin, D. M. Herzig, P. Mika, J. Pound, H. S. Thompson, and D. T. Tran (2011). Entity search evaluation over structured web data. In: *SIGIR-EOS*. Vol. 2011.

Bleiholder, J. and F. Naumann (2008). Data fusion. In: *ACM Comput. Surv.* 41.1, 1:1–1:41.

Boldi, P. and S. Vigna (2005). MG4J at TREC 2005. In: *TREC*.

Bollacker, K. D., C. Evans, P. Paritosh, T. Sturge, and J. Taylor (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In: *SIGMOD*, pp. 1247–1250.

Bordes, A., S. Chopra, and J. Weston (2014). Question answering with subgraph embeddings. In: *CoRR* abs/1406.3676.

Bordes, A. and E. Gabrilovich (2015). Constructing and mining web-scale knowledge graphs: WWW 2015 tutorial. In: *WWW*, p. 1523.

Broekstra, J., A. Kampman, and F. van Harmelen (2002). Sesame: a generic architecture for storing and querying RDF and RDF schema. In: *ISWC*, pp. 54–68.

Bruijn, J. de, M. Ehrig, C. Feier, F. Martín-Recuerda, F. Scharffe, and M. Weiten (2006). Ontology mediation, merging and aligning. In: *Semantic Web Technologies*, pp. 95–113.

Bruni, E., N. Tran, and M. Baroni (2014). Multimodal Distributional Semantics. In: *JAIR* 49, pp. 1–47.

Cafarella, M., A. Halevy, D. Wang, E. Wu, and Y. Zhang (2008). WebTables: exploring the power of tables on the web. In: *PVLDB* 1.1, pp. 538–549.

Cai, Q. and A. Yates (2013). Large-scale semantic parsing via schema matching and lexicon extension. In: *ACL*, pp. 423–433.

Carlson, A., J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell (2010). Toward an architecture for never-ending language learning. In: *AAAI*, pp. 1306–1313.

Carmel, D., M.-W. Chang, E. Gabrilovich, B.-J. P. Hsu, and K. Wang (2014). ERD'14: entity recognition and disambiguation challenge. In: *SIGIR*, p. 1292.

Castano, S., A. Ferrara, S. Montanelli, and G. Varese (2011). Ontology and instance matching. In: *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution*, pp. 167–195.

Charniak, E. (2000). A maximum-entropy-inspired parser. In: *ANLP*, pp. 132–139.

Chen, D. and C. D. Manning (2014). A fast and accurate dependency parser using neural networks. In: *ACL*, pp. 740–750.

Cheng, G., W. Ge, and Y. Qu (2008). Falcons: searching and browsing entities on the semantic web. In: *WWW*, pp. 1101–1102.

Choi, J. D., J. R. Tetreault, and A. Stent (2015). It depends: dependency parser comparison using a web-based evaluation tool. In: *ACL*, pp. 387–396.

Cimiano, P., V. Lopez, C. Unger, E. Cabrio, A.-C. N. Ngomo, and S. Walter (2013). Multilingual question answering over linked data (QALD-3): lab overview. In: *CLEF*, pp. 321–332.

Coffman, J. and A. C. Weaver (2010). A framework for evaluating database keyword search strategies. In: *CIKM*, pp. 729–738.

Coffman, J. and A. C. Weaver (2014). An empirical performance evaluation of relational keyword search techniques. In: *TKDE* 26.1, pp. 30–42.

Cornolti, M., P. Ferragina, M. Ciaramita, H. Schütze, and S. Rüd (2014). The SMAPH system for query entity recognition and disambiguation. In: *ERD*, pp. 25–30.

Corro, L. D. and R. Gemulla (2013). ClausIE: clause-based open information extraction. In: *WWW*, pp. 355–366.

Craven, M., D. DiPasquo, D. Freitag, A. McCallum, T. M. Mitchell, K. Nigam, and S. Slattery (1998). Learning to extract symbolic knowledge from the world wide web. In: *AAAI*, pp. 509–516.

Cucerzan, S. (2012). The MSR system for entity linking at TAC 2012. In: *TAC*.

Cucerzan, S. (2007). Large-scale named entity disambiguation based on Wikipedia data. In: *EMNLP-CoNLL*, pp. 708–716.

Cucerzan, S. (2014). Name entities made obvious: the participation in the ERD 2014 evaluation. In: *ERD*, pp. 95–100.

Dang, H. T., D. Kelly, and J. J. Lin (2007). Overview of the TREC 2007 Question Answering Track. In: *TREC*.

Dang, H. T., J. J. Lin, and D. Kelly (2006). Overview of the TREC 2006 Question Answering Track. In: *TREC*.

Delbru, R., S. Campinas, and G. Tummarello (2012). Searching web data: An entity retrieval and high-performance indexing model. In: *J. Web Sem.* 10, pp. 33–58.

Dill, S., N. Eiron, D. Gibson, D. Gruhl, R. V. Guha, A. Jhingran, T. Kanungo, K. S. McCurley, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien (2003). A case for automated large-scale semantic annotation. In: *J. Web Sem.* 1.1, pp. 115–132.

Ding, L., T. W. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs (2004). Swoogle: a search and metadata engine for the semantic web. In: *CIKM*, pp. 652–659.

Doan, A. and A. Y. Halevy (2005). Semantic integration research in the database community: a brief survey. In: *AI Magazine*, pp. 83–94.

Dong, X., E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang (2014). Knowledge Vault: a web-scale approach to probabilistic knowledge fusion. In: *KDD*, pp. 601–610.

Elbassuoni, S., M. Ramanath, R. Schenkel, M. Sydow, and G. Weikum (2009). Language-model-based ranking for queries on RDF-graphs. In: *CIKM*, pp. 977–986.

Elliott, B., E. Cheng, C. Thomas-Ogbuji, and Z. M. Özsoyoglu (2009). A complete translation from SPARQL into efficient SQL. In: *IDEAS*, pp. 31–42.

Elmagarmid, A. K., P. G. Ipeirotis, and V. S. Verykios (2007). Duplicate record detection: a survey. In: *TKDE*, pp. 1–16.

Etzioni, O., A. Fader, J. Christensen, S. Soderland, and Mausam (2011). Open information extraction: the second generation. In: *IJCAI*, pp. 3–10.

Euzenat, J., A. Ferrara, C. Meilicke, J. Pane, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Sváb-Zamazal, V. Svátek, and C. dos Santos (2010). Results of the ontology alignment evaluation initiative 2010. In: *OM*, pp. 85–117.

Euzenat, J., C. Meilicke, H. Stuckenschmidt, P. Shvaiko, and C. dos Santos (2011a). Ontology alignment evaluation initiative: six years of experience. In: *J. Data Semantics* 15, pp. 158–192.

Euzenat, J., A. Ferrara, W. R. van Hage, L. Hollink, C. Meilicke, A. Nikolov, D. Ritze, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Sváb-Zamazal, and C. dos Santos (2011b). Results of the ontology alignment evaluation initiative 2011. In: *OM*, pp. 158–192.

Fader, A., S. Soderland, and O. Etzioni (2011). Identifying relations for open information extraction. In: *EMNLP*, pp. 1535–1545.

Fader, A., L. S. Zettlemoyer, and O. Etzioni (2013). Paraphrase-driven learning for open question answering. In: *ACL*, pp. 1608–1618.

Fang, Y., L. Si, Z. Yu, Y. Xian, and Y. Xu (2009). Entity retrieval with hierarchical relevance model, exploiting the structure of tables and learning homepage classifiers. In: *TREC*.

Ferrucci, D. A., E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty (2010). Building Watson: An Overview of the DeepQA Project. In: *AI Magazine* 31.3, pp. 59–79.

Ferrucci, D. A., A. Levas, S. Bagchi, D. Gondek, and E. T. Mueller (2013). Watson: Beyond Jeopardy! In: *Artif. Intell.* 199, pp. 93–105.

Finkel, J. R., T. Grenager, and C. D. Manning (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In: *ACL*, pp. 363–370.

Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin (2002). Placing search in context: the concept revisited. In: *TOIS* 20.1, pp. 116–131.

Frank, J., S. Bauer, M. Kleiman-Weiner, D. Roberts, N. Tripuaneni, C. Zhang, C. Ré, E. Voorhees, and I. Soboroff (2013). Stream filtering for entity profile updates for TREC 2013. In: *TREC-KBA*.

Frank, J., M. Kleiman-Weiner, D. A. Roberts, E. Voorhees, and I. Soboroff (2014). Evaluating stream filtering for entity profile updates in TREC 2012, 2013, and 2014. In: *TREC-KBA*.

Frank, J. R., M. Kleiman-Weiner, D. A. Roberts, F. Niu, C. Zhang, C. Ré, and I. Soboroff (2012). Building an entity-centric stream filtering test collection for TREC 2012. In: *TREC-KBA*.

Franz, T., A. Schultz, S. Sizov, and S. Staab (2009). TripleRank: ranking semantic web data by tensor decomposition. In: *ISWC*, pp. 213–228.

Fundel, K., R. Küffner, and R. Zimmer (2007). RelEx - relation extraction using dependency parse trees. In: *Bioinformatics* 23.3, pp. 365–371.

Gabrilovich, E. and S. Markovitch (2007). Computing semantic relatedness using Wikipedia-based Explicit Semantic Analysis. In: *IJCAI*. Vol. 7, pp. 1606–1611.

Gaifman, H. (1965). Dependency systems and phrase-structure systems. In: *Information and Control* 8.3, pp. 304–337.

Gövert, N., N. Fuhr, M. Lalmas, and G. Kazai (2006). Evaluating the effectiveness of content-oriented XML retrieval methods. In: *Information Retrieval* 9.6, pp. 699–722.

Grau, B. C., I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, and U. Sattler (2008). OWL 2: the next step for OWL. In: *J. Web Sem.* 6.4, pp. 309–322.

Grau, B. C., Z. Dragisic, K. Eckert, J. Euzenat, A. Ferrara, R. Granada, V. Ivanova, E. Jiménez-Ruiz, A. O. Kempf, P. Lambrix, A. Nikolov, H. Paulheim, D. Ritze, F. Scharffe, P. Shvaiko, C. T. dos Santos, and O. Zamazal (2013). Results of the ontology alignment evaluation initiative 2013. In: *OM*, pp. 61–100.

Guha, R. V., R. McCool, and E. Miller (2003). Semantic search. In: *WWW*, pp. 700–709.

Guha, R., D. Brickley, and S. MacBeth (2015). Schema.org: evolution of structured data on the web. In: *ACM Queue* 13.9, p. 10.

Guo, Y., Z. Pan, and J. Heflin (2005). LUBM: a benchmark for OWL knowledge base systems. In: *J. Web Sem.* 3, pp. 158–182.

Halpin, H., D. Herzig, P. Mika, R. Blanco, J. Pound, H. Thompson, and D. T. Tran (2010). Evaluating ad-hoc object retrieval. In: *IWEST*.

Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In: *COLING*, pp. 539–545.

Herzig, D. M., P. Mika, R. Blanco, and T. Tran (2013). Federated entity search using on-the-fly consolidation. In: *ISWC*, pp. 167–183.

Hill, F., R. Reichart, and A. Korhonen (2015). SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation. In: *Computational Linguistics* 41.4, pp. 665–695.

Hoffart, J., F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum (2011). YAGO2: exploring and querying world knowledge in time, space, context, and many languages. In: *WWW*, pp. 229–232.

Hoffart, J., F. M. Suchanek, K. Berberich, and G. Weikum (2013). YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. In: *Artif. Intell.* 194, pp. 28–61.

Hoffmann, R., C. Zhang, X. Ling, L. S. Zettlemoyer, and D. S. Weld (2011). Knowledge-based weak supervision for information extraction of overlapping relations. In: *ACL*, pp. 541–550.

Hovy, E. H., M. P. Marcus, M. Palmer, L. A. Ramshaw, and R. M. Weischedel (2006). OntoNotes: the 90% solution. In: *HLT-NAACL*, pp. 57–60.

Hristidis, V. and Y. Papakonstantinou (2002). DISCOVER: keyword search in relational databases. In: *VLDB*, pp. 670–681.

Hua, W., Z. Wang, H. Wang, K. Zheng, and X. Zhou (2015). Short text understanding through lexical-semantic analysis. In: *ICDE*, pp. 495–506.

Ji, H., R. Grishman, and H. T. Dang (2011). Overview of the TAC 2011 Knowledge Base Population Track. In: *TAC-KBP*.

Ji, H., J. Nothman, and B. Hachey (2014). Overview of TAC-KBP 2014 entity discovery and linking tasks. In: *TAC-KBP*.

Ji, H., R. Grishman, H. T. Dang, K. Griffit, and J. Ellisa (2010). Overview of the TAC 2010 Knowledge Base Population Track. In: *TAC-KBP*.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In: *KDD*, pp. 133–142.

Joshi, M., U. Sawant, and S. Chakrabarti (2014). Knowledge graph and corpus driven segmentation and answer inference for telegraphic entity-seeking queries. In: *EMNLP*, pp. 1104–1114.

Kaptein, R. and J. Kamps (2013). Exploiting the category structure of Wikipedia for entity ranking. In: *Artif. Intell.* 194, pp. 111–129.

Katz, B. (1997). Annotating the world wide web using natural language. In: *RIAO*, pp. 136–159.

Katz, B., G. C. Borchardt, and S. Felshin (2006). Natural language annotations for question answering. In: *FLAIRS*, pp. 303–306.

Klein, D. and C. D. Manning (2002). Fast exact inference with a factored model for natural language parsing. In: *NIPS*, pp. 3–10.

Kolomiyets, O. and M. Moens (2011). A survey on question answering technology from an information retrieval perspective. In: *Inf. Sci.* 181.24, pp. 5412–5434.

Köpcke, H. and E. Rahm (2010). Frameworks for entity matching: a comparison. In: *DKE*, pp. 197–210.

Le, Q. V. and T. Mikolov (2014). Distributed representations of sentences and documents. In: *ICML*, pp. 1188–1196.

Lee, D. D. and H. S. Seung (2000). Algorithms for non-negative matrix factorization. In: *NIPS*, pp. 556–562.

Lehmann, J., R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer (2015). DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. In: *Semantic Web* 6.2, pp. 167–195.

Lei, Y., V. S. Uren, and E. Motta (2006). SemSearch: a search engine for the semantic web. In: *EKAW*, pp. 238–245.

Levy, O. and Y. Goldberg (2014). Neural word embedding as implicit matrix factorization. In: *NIPS*, pp. 2177–2185.

Levy, O., Y. Goldberg, and I. Dagan (2015). Improving Distributional Similarity with Lessons Learned from Word Embeddings. In: *TACL* 3, pp. 211–225.

Li, G., S. Ji, C. Li, and J. Feng (2009). Efficient type-ahead search on relational data: a TASTIER approach. In: *SIGMOD*, pp. 695–706.

Li, H. and J. Xu (2014). Semantic matching in search. In: *Foundations and Trends in Information Retrieval* 7.5, pp. 343–469.

Limaye, G., S. Sarawagi, and S. Chakrabarti (2010). Annotating and Searching Web Tables Using Entities, Types and Relationships. In: *PVLDB* 3.1, pp. 1338–1347.

Liu, T. (2009). Learning to rank for information retrieval. In: *Foundations and Trends in Information Retrieval* 3.3, pp. 225–331.

Lopez, V., V. S. Uren, M. Sabou, and E. Motta (2011a). Is question answering fit for the Semantic Web?: A survey. In: *Semantic Web* 2.2, pp. 125–155.

Lopez, V., C. Unger, P. Cimiano, and E. Motta (2011b). Proceedings of the 1st workshop on question answering over linked data (QALD-1). In: *ESWC*.

Lopez, V., C. Unger, P. Cimiano, and E. Motta (2012). Interacting with linked data. In: *ESWC-ILD*.

Lopez, V., C. Unger, P. Cimiano, and E. Motta (2013). Evaluating question answering over linked data. In: *J. Web Sem.* 21, pp. 3–13.

Lund, K. and C. Burgess (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. In: *Behavior research methods, instruments, & computers* 28.2, pp. 203–208.

Luong, T., R. Socher, and C. D. Manning (2013). Better word representations with recursive neural networks for morphology. In: *CoNLL*, pp. 104–113.

Ma, L., Y. Yang, Z. Qiu, G. T. Xie, Y. Pan, and S. Liu (2006). Towards a complete OWL ontology benchmark. In: *ESWC*, pp. 125–139.

Macdonald, C. and I. Ounis (2006). The TREC Blogs06 collection: Creating and analysing a blog test collection. In: *Department of Computer Science, University of Glasgow Tech Report TR-2006-224* 1, pp. 3–1.

Manning, C. D. (2011). Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In: *CICLING*, pp. 171–189.

Manning, C. D., M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky (2014). The Stanford CoreNLP natural language processing toolkit. In: *ACL*, pp. 55–60.

Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz (1993). Building a large annotated corpus of English: the Penn Treebank. In: *Computational Linguistics* 19.2, pp. 313–330.

Mass, Y. and Y. Sagiv (2012). Language models for keyword search over data graphs. In: *WSDM*, pp. 363–372.

Mausam, M. Schmitz, S. Soderland, R. Bart, and O. Etzioni (2012). Open language learning for information extraction. In: *EMNLP-CoNLL*, pp. 523–534.

Mayfield, J., J. Artiles, and H. T. Dang (2012). Overview of the TAC 2012 Knowledge Base Population Track. In: *TAC-KBP*.

Mayfield, J. and R. Grishman (2015). TAC 2015 Cold Start KBP Track. In: *TAC-KBP*.

McClosky, D., E. Charniak, and M. Johnson (2006). Effective self-training for parsing. In: *HLT-NAACL*.

Meusel, R., P. Petrovski, and C. Bizer (2014). The WebDataCommons Microdata, RDFa and Microformat dataset series. In: *ISWC*, pp. 277–292.

Mikolov, T., W. Yih, and G. Zweig (2013). Linguistic regularities in continuous space word representations. In: *NAACL*, pp. 746–751.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013a). Distributed representations of words and phrases and their compositionality. In: *NIPS*, pp. 3111–3119.

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013b). Efficient estimation of word representations in vector space. In: *CoRR* abs/1301.3781.

Miller, G. A. (1992). WordNet: A Lexical Database for English. In: *Commun. ACM* 38, pp. 39–41.

Mintz, M., S. Bills, R. Snow, and D. Jurafsky (2009). Distant supervision for relation extraction without labeled data. In: *ACL/IJCNLP*, pp. 1003–1011.

Mitchell, T. M., W. W. Cohen, E. R. H. Jr., P. P. Talukdar, J. Betteridge, A. Carlson, B. D. Mishra, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. A. Platanios, A. Ritter, M. Samadi, B. Settles, R. C. Wang, D. T. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling (2015). Never-ending learning. In: *AAAI*, pp. 2302–2310.

Mitkov, R. (2014). *Anaphora resolution*. Routledge.

Moldovan, D. I., C. Clark, and M. Bowden (2007). Lymba's PowerAnswer 4 in TREC 2007. In: *TREC*.

Monahan, S., D. Carpenter, M. Gorelkin, K. Crosby, and M. Brunson (2014). Populating a knowledge base with entities and events. In: *TAC*.

Morsey, M., J. Lehmann, S. Auer, and A.-C. N. Ngomo (2011). DBpedia SPARQL benchmark - performance assessment with real queries on real data. In: *ISWC*, pp. 454–469.

Nakashole, N., G. Weikum, and F. M. Suchanek (2012). PATTY: A taxonomy of relational patterns with semantic types. In: *EMNLP*, pp. 1135–1145.

Neumann, T. and G. Weikum (2009). Scalable join processing on very large RDF graphs. In: *SIGMOD*, pp. 627–640.

Neumann, T. and G. Weikum (2010). The RDF-3X engine for scalable management of RDF data. In: *VLDB J.* 19.1, pp. 91–113.

Neumayer, R., K. Balog, and K. Nørvåg (2012). On the modeling of entities for ad-hoc entity search in the web of data. In: *ECIR*, pp. 133–145.

Ng, V. (2010). Supervised noun phrase coreference research: the first fifteen years. In: *ACL*, pp. 1396–1411.

Nivre, J., J. Hall, S. Kübler, R. T. McDonald, J. Nilsson, S. Riedel, and D. Yuret (2007). The CoNLL 2007 Shared Task on dependency parsing. In: *EMNLP-CoNLL*, pp. 915–932.

Noy, N. F. and M. A. Musen (2000). PROMPT: algorithm and tool for automated ontology merging and alignment. In: *AAAI*, pp. 450–455.

Oren, E., R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello (2008). Sindice.com: a document-oriented lookup index for open linked data. In: *IJMSO* 3.1, pp. 37–52.

Orr, D., A. Subramanya, E. Gabrilovich, and M. Ringgaard (2013). 11 billion clues in 800 million documents: a web research corpus annotated with freebase concepts. In: *Google Research Blog*.

Park, S., S. Kwon, B. Kim, and G. G. Lee (2015). ISOFT at QALD-5: hybrid question answering system over linked data and text data. In: *CLEF*.

Pennington, J., R. Socher, and C. D. Manning (2014). Glove: global vectors for word representation. In: *EMNLP*, pp. 1532–1543.

Petrov, S. and R. McDonald (2012). Overview of the 2012 shared task on parsing the web. In: *SANCL*. Vol. 59.

Pickover, C. A., ed. (2012). *This is Watson* 56.3–4: *IBM Journal of Research and Development*.

Popov, B., A. Kiryakov, D. Ognyanoff, D. Manov, and A. Kirilov (2004). KIM - a semantic platform for information extraction and retrieval. In: *Natural Language Engineering* 10.3-4, pp. 375–392.

Pound, J., P. Mika, and H. Zaragoza (2010). Ad-hoc object retrieval in the web of data. In: *WWW*, pp. 771–780.

Pound, J., A. K. Hudek, I. F. Ilyas, and G. E. Weddell (2012). Interpreting keyword queries over web knowledge bases. In: *CIKM*, pp. 305–314.

Prager, J. M. (2006). Open-domain question-answering. In: *Foundations and Trends in Information Retrieval* 1.2, pp. 91–231.

Qi, Y., Y. Xu, D. Zhang, and W. Xu (2014). BUPT_PRIS at TREC 2014 knowledge base acceleration track. In: *TREC*.

Radinsky, K., E. Agichtein, E. Gabrilovich, and S. Markovitch (2011). A word at a time: computing word relatedness using temporal semantic analysis. In: *WWW*, pp. 337–346.

Reddy, S., M. Lapata, and M. Steedman (2014). Large-scale Semantic Parsing without Question-Answer Pairs. In: *TACL* 2, pp. 377–392.

Riedel, S., L. Yao, and A. McCallum (2010). Modeling relations and their mentions without labeled text. In: *ECML PKDD*, pp. 148–163.

Sarawagi, S. (2008). Information Extraction. In: *Foundations and Trends in Databases* 1.3, pp. 261–377.

Schmidt, M., M. Meier, and G. Lausen (2010). Foundations of SPARQL query optimization. In: *ICDT*, pp. 4–33.

Schuhmacher, M., L. Dietz, and S. P. Ponzetto (2015). Ranking entities for web queries through text and knowledge. In: *CIKM*, pp. 1461–1470.

Shvaiko, P. and J. Euzenat (2013). Ontology matching: state of the art and future challenges. In: *TKDE* 25.1, pp. 158–176.

Silvestri, F. (2010). Mining query logs: turning search usage data into knowledge. In: *Foundations and Trends in Information Retrieval* 4.1-2, pp. 1–174.

Sirin, E., B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz (2007). Pellet: A practical OWL-DL reasoner. In: *J. Web Sem.* 5.2, pp. 51–53.

Socher, R., J. Bauer, C. D. Manning, and A. Y. Ng (2013). Parsing with compositional vector grammars. In: *ACL (1)*, pp. 455–465.

Spitkovsky, V. I. and A. X. Chang (2012). A cross-lingual dictionary for english wikipedia concepts. In: *LREC*, pp. 3168–3175.

Suchanek, F. M., G. Kasneci, and G. Weikum (2007). YAGO: a core of semantic knowledge. In: *WWW*, pp. 697–706.

Surdeanu, M. (2013). Overview of the TAC 2013 Knowledge Base Population evaluation: english slot filling and temporal slot filling. In: *TAC-KBP*.

Surdeanu, M. and H. Ji (2014). Overview of the english slot filling track at the TAC 2014 Knowledge Base Population evaluation. In: *TAC-KBP*.

Surdeanu, M., J. Tibshirani, R. Nallapati, and C. D. Manning (2012). Multi-instance multi-label learning for relation extraction. In: *EMNLP-CoNLL*, pp. 455–465.

Tablan, V., K. Bontcheva, I. Roberts, and H. Cunningham (2015). Mímir: An open-source semantic search framework for interactive information seeking and discovery. In: *J. Web Sem.* 30, pp. 52–68.

Tonon, A., G. Demartini, and P. Cudré-Mauroux (2012). Combining inverted indices and structured search for ad-hoc object retrieval. In: *SIGIR*, pp. 125–134.

Toutanova, K., D. Klein, C. D. Manning, and Y. Singer (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In: *HLT-NAACL*, pp. 173–180.

Tran, T., H. Wang, and P. Haase (2009). Hermes: data web search on a pay-as-you-go integration infrastructure. In: *J. Web Sem.* 7.3, pp. 189–203.

Tran, T., P. Cimiano, S. Rudolph, and R. Studer (2007). Ontology-based interpretation of keywords for semantic search. In: *ISWC/ASWC*, pp. 523–536.

Trotman, A., C. L. A. Clarke, I. Ounis, S. Culpepper, M. Cartright, and S. Geva (2012). Open source information petrieval: a report on the SIGIR 2012 workshop. In: *SIGIR Forum* 46.2, pp. 95–101.

Unbehauen, J., C. Stadler, and S. Auer (2013). Optimizing SPARQL-to-SQL rewriting. In: *IIWAS*, p. 324.

Unger, C., L. Bühmann, J. Lehmann, A. N. Ngomo, D. Gerber, and P. Cimiano (2012). Template-based question answering over RDF data. In: *WWW*, pp. 639–648.

Unger, C., C. Forascu, V. Lopez, A. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter (2014). Question answering over linked data (QALD-4). In: *CLEF*, pp. 1172–1180.

Unger, C., C. Forascu, V. Lopez, A. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter (2015). Question answering over linked data (QALD-5). In: *CLEF*.

Voorhees, E. M. (1999). The TREC-8 Question Answering Track Report. In: *TREC*.

Voorhees, E. M. (2000). Overview of the TREC-9 Question Answering Track. In: *TREC*.

Voorhees, E. M. (2001). Overview of the TREC 2001 Question Answering Track. In: *TREC*.

Voorhees, E. M. (2002). Overview of the TREC 2002 Question Answering Track. In: *TREC*.

Voorhees, E. M. (2003). Overview of the TREC 2003 Question Answering Track. In: *TREC*.

Voorhees, E. M. (2004). Overview of the TREC 2004 Question Answering Track. In: *TREC*.

Voorhees, E. M. and H. T. Dang (2005). Overview of the TREC 2005 Question Answering Track. In: *TREC*.

Voorhees, E. M. and D. K. Harman (2005). *TREC: Experiment and evaluation in information retrieval*. Vol. 63. MIT press Cambridge.

Wang, H., Q. Liu, T. Penin, L. Fu, L. Zhang, T. Tran, Y. Yu, and Y. Pan (2009). Semplore: A scalable IR approach to search the Web of Data. In: *J. Web Sem.* 7.3, pp. 177–188.

Wang, Q., J. Kamps, G. R. Camps, M. Marx, A. Schuth, M. Theobald, S. Gurajada, and A. Mishra (2012). Overview of the INEX 2012 Linked Data Track. In: *CLEF*.

Wu, S., C. Zhang, F. Wang, and C. Ré (2015). Incremental Knowledge Base Construction Using DeepDive. In: *PVLDB* 8.11, pp. 1310–1321.

Xu, K., Y. Feng, and D. Zhao (2014). Answering natural language questions via phrasal semantic parsing. In: *CLEF*, pp. 1260–1274.

Yahya, M., K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum (2012). Natural language questions for the web of data. In: *EMNLP-CoNLL 2012*, pp. 379–390.

Yates, A., M. Banko, M. Broadhead, M. J. Cafarella, O. Etzioni, and S. Soderland (2007). TextRunner: open information extraction on the web. In: *HLT-NAACL*, pp. 25–26.

Yih, W., M. Chang, X. He, and J. Gao (2015). Semantic parsing via staged query graph generation: question answering with knowledge base. In: *ACL*, pp. 1321–1331.

Yu, J. X., L. Qin, and L. Chang (2010). Keyword search in relational databases: a survey. In: *IEEE Data Eng. Bull.* 33.1, pp. 67–78.

Zaragoza, H., N. Craswell, M. J. Taylor, S. Saria, and S. E. Robertson (2004). Microsoft cambridge at TREC 13: web and hard tracks. In: *TREC*.

Zelenko, D., C. Aone, and A. Richardella (2003). Kernel methods for relation extraction. In: *Journal of Machine Learning Research* 3, pp. 1083–1106.

Zenz, G., X. Zhou, E. Minack, W. Siberski, and W. Nejdl (2009). From keywords to semantic queries - Incremental query construction on the semantic web. In: *J. Web Sem.* 7.3, pp. 166–176.

Zhang, C. (2015). DeepDive: A Data Management System for Automatic Knowledge Base Construction. PhD thesis. University of Wisconsin-Madison.

Zhiltsov, N., A. Kotov, and F. Nikolaev (2015). Fielded sequential dependence model for ad-hoc entity retrieval in the web of data. In: *SIGIR*, pp. 253–262.

Zhou, G., J. Su, J. Zhang, and M. Zhang (2005). Exploring various knowledge in relation extraction. In: *ACL*, pp. 427–434.

Zhou, Q., C. Wang, M. Xiong, H. Wang, and Y. Yu (2007). SPARK: adapting keyword query to semantic search. In: *ISWC/ASWC*, pp. 694–707.

Broccoli: Semantic Full-Text Search at your Fingertips

Hannah Bast, Florian Bäurle, Björn Buchhold, Elmar Haussmann
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast,baeurlef,buchholb,haussmann}@informatik.uni-freiburg.de

ABSTRACT

We present Broccoli, a fast and easy-to-use search engine for what we call semantic full-text search. Semantic full-text search combines the capabilities of standard full-text search and ontology search. The search operates on four kinds of objects: ordinary words (e.g., *edible*), classes (e.g., *plants*), instances (e.g., *Broccoli*), and relations (e.g., *occurs-with* or *native-to*). Queries are trees, where nodes are arbitrary bags of these objects, and arcs are relations. The user interface guides the user in incrementally constructing such trees by instant (search-as-you-type) suggestions of words, classes, instances, or relations that lead to good hits. Both standard full-text search and pure ontology search are included as special cases. In this paper, we describe the query language of Broccoli, the main idea behind a new kind of index that enables fast processing of queries from that language as well as fast query suggestion, the natural language processing required, and the user interface. We evaluated query times and result quality on the full version of the English Wikipedia (40 GB XML dump) combined with the YAGO ontology (26 million facts). We have implemented a fully functional prototype based on our ideas and provide a web application to reproduce our quality experiments. Both are accessible via `http://broccoli.informatik.uni-freiburg.de/repro-corr/` .

1. INTRODUCTION

In this paper, we describe a novel implementation of what we call *semantic full-text search*. Semantic full-text search combines traditional *full-text search* with structured search in knowledge databases or *ontology search* as we call it in this paper.

In traditional full-text search you type a (typically short) list of keywords and you get a list of documents containing some or all of these keywords, hopefully ranked by some notion of relevance to your query. For example, typing *broccoli leaves edible* in a web search engine will return lots of web pages with evidence that broccoli leaves are indeed edible.

In ontology search, you are given a knowledge database which you can think of as a store of subject-predicate-object triples. For example, *Broccoli is-a plant* or *Broccoli native-to Europe*. These triples can be thought of to form a graph of entities (the nodes) and relations (the edges), and ontology search allows you to search for subgraphs matching a given pattern. For example, find all plants that are native to Europe.

Many queries of a more "semantic" nature require the combination of both approaches. For example, consider the query *plants with edible leaves and native to Europe*, which will be our running example in this paper. A satisfactory answer for this query requires the combination of two kinds of information. First, a list of plants native to Europe. This is hard for full-text search but a showcase for ontology search, see above. Second, for each plant the information whether its leaves are edible or not. This kind of information can be easily found with a full-text search for each plant, see above. But it is quite unlikely (and unreasonable) to be contained in an ontology, for reasons explained in Section 2.3.

The basic principle of our combined search is to find *contextual* co-occurrences of the words from the full-text part of the query with entities matching the ontology part of the query. Consider the sentence: *The stalks of rhubarb are edible, but its leaves are toxic.* Assume for now that we can recognize entities from the ontology in the full text (we come back to this in Section 3.2). In this case, the two underlined words both refer to *rhubarb*, which our ontology knows is a plant that is native to Europe. Obviously, this sentence should *not* count as evidence that *rhubarb leaves are edible.* We handle this by decomposing each sentence into what we call its *contexts*: the parts of the sentence that "belong" together. In this case *the stalks of rhubarb are edible* and *rhubarb leaves are toxic.* An arc from the query tree now matches if and only if its elements co-occur in one and the same context.

Figures 1 and 2 show screenshots of our search engine in action for our example query. The figures and their captions also explain how the query can be constructed incrementally in an easy way and without requiring knowledge of a particular query language on the part of the user. We encourage the reader to try our online demo that is accessible via `http://broccoli.informatik.uni-freiburg.de/repro-corr/` .

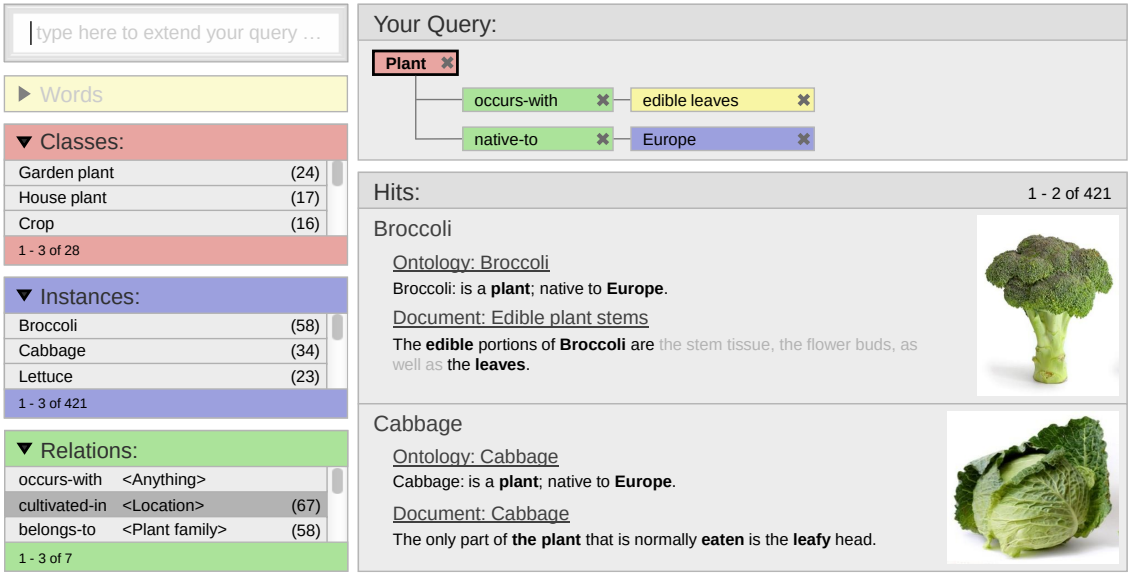

Figure 1: A screenshot of the final result for our example query. The box on the top right visualizes the current query as a tree. There is always one node in focus (shown in bold), in this case, the root of the tree. The large box below shows the hits grouped by instance (of the class from the root node) and ranked by relevance (if Broccoli is among the hits, we always rank it first). Evidence both from the ontology and the full text is provided. For the latter, a whole sentence is shown, with parts outside of the matching context grayed out. With the search field on the top left, the query can be extended further. The four boxes below provide context-sensitive suggestions that depend on the current focus in the query, here: suggestions for sub*classes* of plants, suggestions for *instances* of plants that lead to a hit, suggestions for *relations* to further refine the query. One of the suggestions is always highlighted, in this case the *cultivated-in* relation. It can be directly added to extend the query by pressing Return.

1.1 Our contribution

Broccoli supports a subset of SPARQL[1] (essentially trees with a single free variable at the root) for the ontology part of queries. Moreover, it allows a special *occurs-with* relation that can be used to specify co-occurrence of a class (e.g., *plant*) or instance (e.g., *Broccoli*) with an arbitrary combination of words, instances, and further subqueries. Both traditional full-text search and pure ontology search are subsumed as special cases. This gives a very powerful query language. See Section 4 for details.

For the *occurs-with* relation, we provide a novel kind of pre-processing that decomposes sentences into *contexts* of words that belong together. In particular, this considers enumerations and sub-clauses. Previous approaches have used co-occurrence in a whole paragraph or sentence, or based on word proximity; all of these often give poor results. See Section 3 for details.

We present the key idea behind a novel kind of index that supports fully interactive query times of around 100 milliseconds and less for a collection as large as the full English Wikipedia (40 GB XML dump, 418 million contexts of the kind just described). Previous approaches, including adaptations of the classic inverted index, yield query times on the order of seconds or even minutes for the kind of queries we support on collections of this size. See Section 2.1 for related work, and Section 5 for details.

All the described features have been implemented into a fully functional system with a comfortable user interface. There is a single search field, as in full-text search, and suggestions are made after each keystroke. This allows the user to incrementally construct semantic full-text queries without prior knowledge of a query language. Results are ranked by relevance and grouped by instance, and displayed together with context snippets that provide full evidence for why that particular instance is shown. See Figures 1 and 2 for an example, and Section 6 for details.

We provide experimental results on the result quality for the English Wikipedia combined with the YAGO ontology [20]. For the quality results, we used 46 Queries from the SemSearch List Search Track (e.g., *Apollo astronauts who walked on the Moon*), 15 queries from the TREC 2009 Entity Track benchmarks (e.g., *Airlines that currently use Boeing 747 planes*) and 10 lists from Wikipedia (e.g. *List of participating nations at the Winter Olympic Games*). We allow reproducing our results at `http://broccoli.informatik.uni-freiburg.de/repro-corr/` . See Section 7 for the details of our experiments.

We want to remark that the natural language processing, the index, and the user interface behind Broccoli are com-

[1] `http://www.w3.org/TR/rdf-sparql-query`

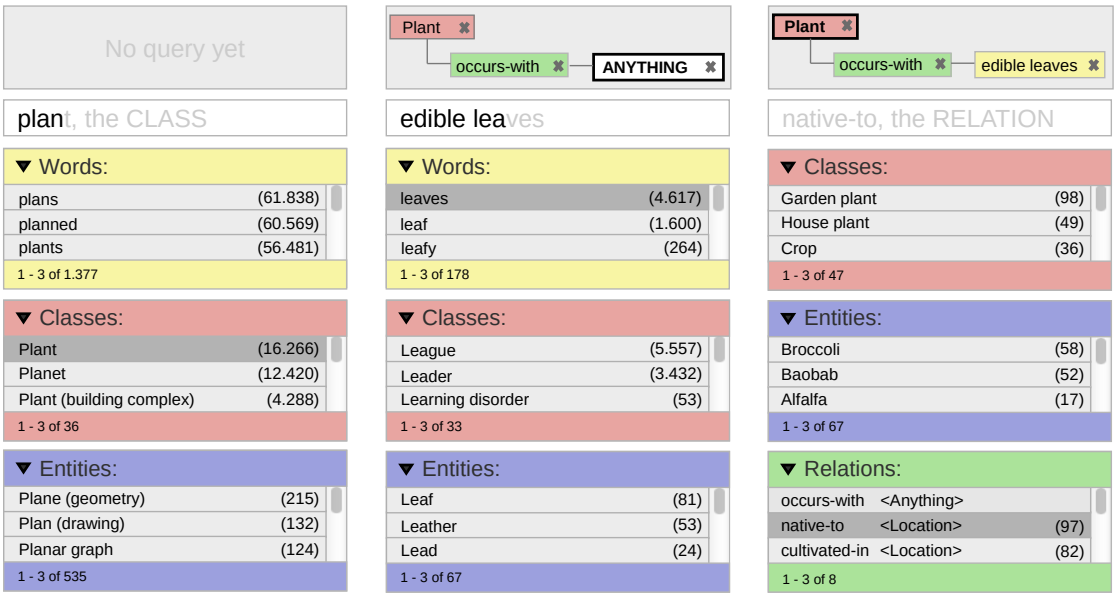

Figure 2: Snapshots of the query, search field, and suggestion boxes for three stations in the construction of our example query. Column 1: At the beginning of the query, after having typed *plan*. Column 2: After the class *plant* has been selected and the *occurs-with* relation has been added and having typed *edible lea*. Column 3: After having selected *edible leaves*. The focus automatically goes back to the root node.

plex problems each on their own. The contribution of this paper is the overall design of the system, the basic ideas for each of the mentioned components, an implementation of a fully functional prototype based on these ideas, and a first performance and quality evaluation providing a proof of concept. Optimization of the various components is the next step in this line of research; see Section 8.

2. RELATED WORK

Putting the work presented in this paper into context is hard for two reasons. First, the literature on semantic search technologies is vast. Second, "semantic" means so many different things to different researchers. We roughly divide work in this broad area into four categories, and discuss each category separately in the following four subsections.

2.1 Combined ontology and full-text search

Ester [7] was the first system to offer efficient combined full-text and ontology search on a collection as large as the English Wikipedia. Broccoli improves upon Ester in three important aspects. First, Ester works with inverted lists for classes and achieves fast query times only on relatively simple queries. Second, Ester does not consider contexts but merely syntactic proximity of words / entities. Third, Ester's simplistic user interface was ok for queries with one relation, but practically unusable for more complex queries.

Various other systems offering combinations of full-text and ontology search have been proposed. Semplore [22] supports a query language similar to ours. However, elements from the ontology are not recognized in their contexts, but there is simply one piece of text associated with each instance (which would correspond to a single large context in our setting). Queries are processed with a standard inverted index , and no particular UI is offered. In Hybrid Search [8], the full text and the ontology are searched separately with standard methods (Lucene and Sesame), and then the results are combined. There is no particular natural language processing. Concept Search [15] adds information about identified noun phrases and hyponyms to the index. Queries are bags of words, which are interpreted semantically. The query processing uses standard methods (Lucene), with very long inverted lists for the semantic index items. GoNTogle [14] combines full text with annotations which are searched separately and then combined, similarly as in [8]. Queries are bags of words. There is no full ontology search and no particular natural language processing. Faceted Wikipedia Search [16] offers a user interface with similarities to ours. However, the query language is restricted, there is nothing comparable to our contexts but only a small abstract per entity like in [22], and query processing is DB-based and very slow, despite the relatively small amount of data. SIREN[2] provides an integration of pure ontology search into Lucene. How to combine the then possible full-text and ontology searches is up to the user of the framework. Finally, systems like [23] try to interpret a given keyword query semantically and translate it into a suitable SPARQL query for pure ontology search.

[2] `http://siren.sindice.com`

2.2 Systems for entity retrieval

Entity retrieval is a line of research which focuses on search requests and corresponding result lists centered around entities (instead of around documents, as in traditional search). Since 2009, there is also a corresponding Entity Track at TREC[3]. The tasks of this track are both simpler and harder than what we aim at in this paper.

They are harder because the overall goal is entity retrieval from *web pages*. The ClueWeb09 collection introduced at TREC 2009 is 25 TB of text. The relative information content is, however, low as is typical for web contents. Moreover, identifying a representative web page for an entity is part of the problem.

To make the tasks feasible at all under these circumstances, the queries are relatively simple. For example, *Airlines that currently use Boeing 747 planes*.[4] Even then the tasks remain very hard, and, for example, *NDCG@R* figures average only around 30% even for the best systems [4].

Broccoli queries can be trees of arbitrary degree and depth. All entities that have a Wikipedia page are supported. And, most importantly, the query process is interactive, providing the user with *instant* feedback of what is in the collection and why a particular result appears. This is key for constructing queries that give results of high quality.

The price we pay is a more extensive pre-processing assuming a certain "cleanliness" of the input collection. Our natural language processing currently requires around 1600 core hours on the 40 GB XML dump of the English Wikipedia. And Wikipedia's rule of linking the first occurrence of an important entity in an article to the respective Wikipedia article helps us for an entity recognition of good quality; see Section 3.2. Bringing Broccoli's functionality to web search is a very reasonable next step, but out of scope for this article.

Another popular form of entity retrieval is known as *ad-hoc object retrieval* [18]. Here, the search is on structured data, as discussed in the next subsection. Queries are given by a sequence of keywords, similar as in full-text search, for example, *doctors in barcelona*. Then query interpretation becomes a non-trivial problem; see Section 2.4.

2.3 Information extraction and ontology search

Systems for ontology search have reached a high level of sophistication. For example, RDF-3X can answer complex SPARQL queries on the Barton dataset (50 million triples) in less than a second on average [17].

As part of the Semantic Web / Linked Open Data [9] effort, more and more data is explicitly available as fact triples. The bulk of useful triple data is still harvested from text documents though. The information extraction techniques employed range from simple parsing of structured information (for example, many of the relations in YAGO or DBpedia [2] come from the Wikipedia info boxes) over pattern matching (e.g., [1]) to complex techniques involving non-trivial natural language processing like in our paper (e.g., [5]). For a relatively recent survey, see [19].

Our work differs from this line of research in two important aspects: (1) the full text remains part of the index that is searched at query time; and (2) our system is fully interactive and keeps the human in the loop in the information extraction process. This has the following advantage:

Ontologies are good for facts like *which plants are native to which regions*, *who was born where on which date*, etc. Such facts are easy to define and can be extracted from existing data sources in large quantity and with reasonable quality. And once in the ontology, they are easily combinable, permitting queries that would not work with full-text search.

But for more complex facts like our *broccoli has edible leaves*, it is the other way round. They are easy to express and search in full text, but tedious to define, include, and maintain in an ontology. Let alone the problem of guessing the right relation names when searching for them.

By keeping the full text, we can leverage the intelligence of the user at query time. The query *Plant occurs-with edible leaves* does not specify the type of the relation between the occurrence of the plant and the occurrence of the words *edible* and *leaves*. Yet a moment's thought reveals that it is quite likely that a context matching these elements gives us what we want. Similarly as in full-text search, there is often no need to be overly precise in order to get what you want. And just like the result snippets in full-text search, Broccoli's result snippets provide instant feedback on whether the listed plant is really one with edible leaves.

Finally, if information extraction is desired nevertheless, Broccoli can be a useful tool for interactively exploring the collection with respect to the desired information, and for formulating appropriate queries.

2.4 Systems for question answering

Question answering (QA) systems provide similar functionality as our semantic full-text search. The crucial difference is that questions can be asked in natural language, which makes the answering part much harder. The system is burdened with the additional and very complex task of "translating", in one way or the other, the given natural language query into a more formal query or queries that can be fed to a search engine and / or a knowledge database.

The perfect QA system would obviate the need for a system like ours here. But research is still far from achieving that goal. All state-of-the-art QA systems, including the big commercial ones, are specialized to quite particular kinds of questions. For example, Wolfram Alpha works perfectly for *Which cities in China have more than 10 million inhabitants*, but does not work if *more* is replaced by *less* or *China* by *Asia*, and does not even understand the question *Which plants have edible leaves*. IBM's Watson was tuned for finding the single most probable entity when given one of the (intentionally obscured) clues from the Jeopardy! game. And both of these systems lack transparency: it is hard to predict whether a question will be understood correctly, it is hard to understand the reasons for a missing or wrong answer, and there is no possibility of interaction or query refinement.

For our semantic full-text search both the query language and the relation between a given query and its result are well-defined and maximally transparent to the user; see the discussion in Section 2.3. The price we pay is query formulation in a non-natural language. The success of full-text search has shown that as long as the language is simple enough, it can work.

[3] `http://ilps.science.uva.nl/trec-entity`

[4] In our framework these are queries with two nodes and one *occurs-with* edge.

3. INPUT DATA AND NATURAL LANGUAGE PRE-PROCESSING

3.1 Input data

Broccoli requires two kinds of inputs, a text collection and an ontology. The text collection consists of documents containing plain text. The ontology consists of typed *relations* with each relation containing an arbitrary set of fact triples. The subjects and objects of the triples are called *instances*. Each instance belongs to one or more *classes*. The classes are organized in a taxonomy; the root class is called *Entity*.

3.2 Entity recognition

The first step is to identify mentions of or referrals to instances from the ontology in the text documents. Consider the following sentence, which will be our running example for this section:

(S) *The usable parts of rhubarb, a plant from the Polygonaceae family, are the medicinally used roots and the edible stalks, however its leaves are toxic.*

Both *rhubarb* and *its* refer to the instance *Rhubarb* from our ontology, which in turn belongs to the classes *Plant* and *Vegetable* (among others).

Our entity recognition on the English Wikipedia is simplistic but reasonably effective. As a rule, first occurrences of entities in Wikipedia documents are linked to their Wikipedia page. When parsing a document, whenever a part or the full name of that entity is mentioned again in the same section of the document (for example, *Einstein* referring to *Albert Einstein*), we recognize it as that entity.

We resolve anaphora in an equally simplistic way. Namely, we assign each occurrence of *he*, *she*, *it*, *her*, *his*, etc. to the last recognized entity of matching gender. We also recognize the pattern *the <class>* as the entity of the document if it belongs to *<class>*, for example, *the plant* in the document of *Broccoli*.

Our results in Section 7.5 suggest that, on Wikipedia, these simple procedures give already a reasonable accuracy.

3.3 Natural language processing

The second step is to decompose document texts into what we call *contexts*, that is, sets of words that "belong" together. The contexts for our example sentence (S) from above are:

(C1) *rhubarb, a plant from the Polygonaceae family*

(C2) *The usable parts of rhubarb are the medicinally used roots*

(C3) *The usable parts of rhubarb are the edible stalks*

(C4) *however rhubarb leaves are toxic*

This will be crucial for the quality of our results, because we do not want to get rhubarb in our answer set when searching for *plants with edible leaves*. Note that we assume here that the entity recognition and anaphora resolution have already been done (underlined words). Also note that we do not care whether our contexts are grammatically correct and form a readable text. This distinguishes our approach from a line of research called *text simplification* [12].

In the following, we will only consider contexts that are part of a single sentence. Indeed, after anaphora resolution, it seems that most simple facts are expressed within one and the same sentence. Our evaluation in Section 7.5 confirms this assumption.

Our context decomposition consists of two parts, each described in the following subsections.

3.3.1 Sentence constituent identification (SCI)

The task of SCI is to identify the basic "building blocks" of a given sentence. For our purposes various kinds of *subclauses* and *enumeration items* will be important, because they usually contain separate facts that have no direct relationship to the other parts of the sentence. For example, in our sentence (S) from above, the relative clause *a plant from the Polygonaceae family* refers to *rhubarb* but has nothing to do with the rest of the sentence. Similarly, the two enumeration items *the medicinally used roots* and *the edible stalks* have nothing to do with each other (except that they both refer to rhubarb); in particular, rhubarb roots are not edible and rhubarb stalks are not medicinally used. Finally the part *however its leaves are toxic* needs to be considered separate from the preceding part of the sentence. As will become clear in the following, we consider these as enumeration items on the top level of the sentence.

Formally, SCI computes a tree with three kinds of nodes: *enumeration (ENUM)*, *sub-clause (SUB)*, and *concatenation (CONC)*. The leaves contain parts of the sentence and a concatenation of the leaves from left to right yields the whole sentence again. See Figure 3 for the SCI tree of the above sentence.

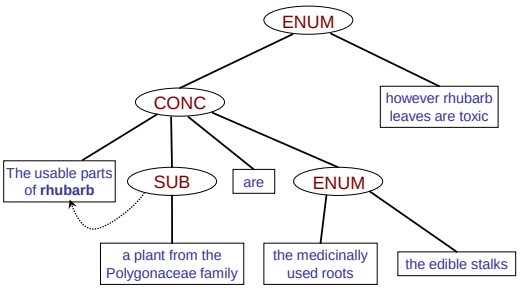

Figure 3: The SCI tree for our example sentence after anaphora resolution. The head of the sub-clause is printed in bold.

We construct our SCI trees based on the output of a state-of-the-art constituent parser. We use SENNA [13], because of its good trade-off between parse time (around 35ms per sentence) and result quality (see Section 7.5).

We transform the parse tree using a relatively small set of hand-crafted rules. Here is a selection of the most important rules; the complete list consists of only 11 rules but is omitted here for the sake of brevity. In the following description when we speak of an *NP* (noun phrase), *VP* (verb phrase), *SBAR* (subordinate clause), or *PP* (prepositional phrase) we refer to nodes in the parse tree with that tag.

(SCI 1) Mark as ENUM each node, for which the children (excluding punctuation and conjunctions) are either all *NP* or all *VP*.

(SCI 2) Mark as SUB each *SBAR*. If it starts with a word from a positive-list (e.g., *which* or *who*) define the first *NP* on the left as the *head* of this SUB; this will be used in (SCR 0) below.

(SCI 3) Mark as SUB each *PP* starting with a preposition from a positive-list (e.g., *before* or *while*), and all *PP*s at the beginning of a sentence. These SUBs have no head.

(SCI 4) Mark as CONC all remaining nodes and contract away each CONC with only text nodes in its subtree (by merging the respective text).

As our quality evaluation in Section 7.5 shows, our rules work reasonably well.

3.3.2 Sentence constituent recombination (SCR)

In SCR we recombine the constituents identified by the SCI to form our *contexts*, which will be the units for our search. Recall that the intuition is to have contexts such that only those words which "belong" together are in the same context. SCR recursively computes the following contexts from a SCI tree or subtree:

(SCR 0) Take out each subtree labeled SUB. If a head was defined for it in (SCI 2), add that head as the leftmost child (but leave it in the SCI tree, too). Then process each such subtree and the remaining part of the original SCI tree (each of which then only has ENUM and CONC nodes left) separately as follows:

(SCR 1) For a leaf, there is exactly one context: the part of the sentence stored in that leaf.

(SCR 2a) For an inner node, first recursively compute the set of contexts for each of its children.

(SCR 2b) If the node is marked ENUM, the set of contexts for this node is computed as the *union* of the sets of contexts of the children.

(SCR 2c) If the node is marked CONC, the set of contexts for this node is computed as the *cross-product* of the sets of contexts of the children.

We remark that once we have the SCI tree, SCR is straightforward, and that the time for both SCI + SCR is negligible compared to the time needed for the full-parse of the sentences.

4. QUERY LANGUAGE

Queries to Broccoli are rooted trees with arcs directed away from the root. The root is either a class or an instance. There are two types of arcs: *ontology arcs* and *occurs-with arcs*. Both have a class or instance as source node.

Ontology arcs are labeled by a relation from the ontology. The two nodes must be classes or instances matching the source and target type of the relation. The class or instance at the target node may be the root of another arbitrary tree.

For occurs-with arcs, the target node can be an arbitrary set of words, prefixes, instances or classes. The instances or classes may themselves be the root of another arbitrary query. Example queries are given in Figures 1 and 2.

To give an example of a more complex query: in Figure 1 we could replace the instance node *Europe* by a class node *Location* and add to it an *occurs-with* arc with the word *equator* in its target node. The intention of this query would be to obtain plants with edible leaves native to regions at or near the equator.

5. INDEX AND QUERY PROCESSING

The index and query processing of Broccoli are described in detail in [6]. In this section, we summarize why standard indexes are not suited for Broccoli and describe the main idea behind our new index.

There are sophisticated systems for both, full-text search and search in ontologies. Since our queries combine both tasks, three ways to answer our queries using those system come to mind: (1) incorporate ontology information into an inverted index; (2) incorporate full-text information into a triple store; (3) use an inverted index for the full-text part of the query, a triple store for the ontology part of the query, and then combine the results somehow.

Neither approach is perfectly suited for our use-case. In a nutshell, approach (1) produces document-centric results and cannot be used to answer complex queries that involve join operations. Approach (2) needs a relation (e.g. *occurs-in-context* featuring both, words and entities) of the size of our entire index to make use of the contexts produced in our contextual sentence decomposition. Efficient queries require a special purpose index over this relation, which already goes in the direction of our approach. Finally, approach (3) will get a list of contexts as a result from the full-text index and has to derive all entities that occur in those contexts. This mapping is not trivial to achieve efficiently, especially since a full mapping from contexts to entities usually does not fit in memory for large collections. Apart from that, we allow queries that demand co-occurrence with some entity from a list that can be the root of another query (e.g. a query for politicians that are friends with an astronaut who walked on the moon). This would require a second mapping in the other direction: from entities to contexts. In summary, the two problems are: Given a list of contexts C, produce a list E of entities that occur in those contexts. Given a list of contexts C and an entity list E, limit C to contexts that include at least one entity from E.

The main idea behind our new index solves these two problems. We use what we call *context lists* instead of standard inverted lists. The context list for a prefix contains one index item per occurrence of a word starting with that prefix, just like the inverted list for that prefix would. But along with that it also contains one index item for each occurrence of an arbitrary entity in the same context as one of these words. For example, consider the context *the usable parts of rhubarb are its edible stalks*, with recognized entities underlined. And let us assume that we have an inverted list for each 4-letter prefix. Then the part of the context list for *edib** pertaining to this context (which has id, say, 14) would be:

edib*:	...	C14	C14	C14	...
	...	#edible	#Rhubarb	#Stalk	...
	...	1	1	1	...
	...	8	5	9	...

The numbers in the first row are context ids. The # in the second row means that not the actual entities (with capital letters) or words are stored, but rather unique ids for them. The third row contains the score for each index item. The fourth row contains the position of the word or entity in the respective context. The context lists are sorted by context id, and, for equal context ids, by word/entity id, with entities coming after the words.

Since entity postings are included in those lists, we can easily solve the two problems introduced above. Actually, our index and query processing support many additional features like excerpt generation, suggestions, prefix search, search for documents instead of entities or ranges over values. For details on those features and a detailed description of the query processing, we again refer the reader to [6].

6. USER INTERFACE

For a convincing proof of concept for our interactive semantic search, we have taken great care to implement a fully functional and intuitive user interface. In particular, there is no need for the user to formulate queries in a language like SPARQL. We claim that any user familiar with full-text search will learn how to use Broccoli in a short time, simply by typing a few queries and following the various query suggestions. The user interface is completely written in Java using the Goole Web Toolkit[5].

The introduction and screenshots (Figures 1 and 2) have already provided a foretaste of the capabilities of our user interface. Here is a list of its most important further features:

(UI 1) Search as you type: New suggestions and results with every keystroke. Very importantly, Broccoli's suggestions for words, classes, instances, and relations are context-sensitive. That is, the displayed suggestions actually lead to hits, and the more / higher-scored hits they lead to, the higher they are ranked.

(UI 2) Pre-select of most likely suggestion: Broccoli knows four kinds of objects: words, classes, instances, and relations. Depending on where you are in the query construction, you get suggestions for several of them. A new user may be overwhelmed to understand the different semantics of the different boxes. For that reason, after every keystroke Broccoli highlights the most meaningful suggestion, which can be selected by simply pressing *Return*.

(UI 3) Visual query representation: At any time, the current query is shown as a tree, with a color code for the various elements that is consistent with the suggestion boxes.

(UI 4) Change of focus / root: A click on any node in the query tree will change the focus of the query suggestions to that node. A double-click on any class or instance node will make that node the root of the tree and re-group and re-rank the results accordingly.

(UI 5) Full history support: The forward and backward buttons of the browser can be used to undo or redo single steps of the query creation process. Furthermore the current URL of the interface can always be used to store its current state or to exchange created queries with others.

(UI 6) Tutorial: Besides some pre-built example queries, the interface also provides a tutorial mode that shows how to create a search query step by step.

7. EXPERIMENTS

7.1 Input data

Our text collection is the text from all documents in the English Wikipedia, obtained via `download.wikimedia.org` in January 2013. Some dimensions of this collection: 40 GB XML dump, 2.4 billion word occurrences (1.6 billion without stop-words), 285 million recognized entity occurrences and 200 million sentences which we decompose into 418 million contexts.

As ontology we use the latest version of YAGO from October 2009. We manually fixed 92 obvious mistakes in the ontology (for example, the *noble prize* was a *laureate* and hence a *person*), and added the relation *Plant native-in Location* for demonstration purposes. Altogether our variant of YAGO contains 2.6 million entities, 19,124 classes, 60 relations, and 26.6 million facts.

7.2 Pre-processing

We use a UIMA[6] pipeline to pre-process the Wikipedia XML. The pipeline includes self-written components to parse the Wikipedia markup, tokenize text, parse sentences using SENNA [13], perform entity-recognition and anaphora resolution (see section 3.2), and decompose the sentences (see section 3.3). We want to note that all these components can easily be exchanged. In principle, this allows Broccoli to work with any given text collection and ontology.

The full parse with SENNA was scaled out asynchronously on a cluster of 8 PCs, each equipped with an AMD FX-8150 8-core processor and 16 GB of main memory. A final non-UIMA component writes the binary index which is kept in three separate files. The file for the context lists has a size of 37 GB. The file for the relation lists has a size of 0.5 GB. And the file for the document excerpts has a size of 276 GB, which could easily be reduced to 85 GB by eliminating the redundant and debug information the file currently contains.

7.3 Computing environment

The code for the index building and query processing is written entirely in C++. The code for the query evaluation is written in Perl, Java, C++ and JavaScript. Our pre-processing components are written in C++ or Java. All performance tests were run on a single core of a Dell PowerEdge server with 2 Intel Xeon 2.6 GHz processors, 96 GB of main memory, and 6x900 GB SAS hard disks configured as Raid-5.

7.4 Query times

For detailed experiments on query times, we refer to the paper describing the index behind Broccoli [6]. In said paper, we have evaluated our system on 8,000 queries of different complexity and 35,000 suggestions. Therefore we here omit a detailed breakdown and limit ourselves to the figures reported in Table 1.

Query set	average	median	90%ile	99%ile
Hit queries	52ms	23ms	139ms	393ms
Suggestion	19ms	6ms	44ms	193ms

Table 1: Statistics of query times over 8,000 queries and 35,000 suggestions.

On our collection, 90% of the queries finish within 140ms, 99% within 400ms. Suggestions are even faster. The breakdown in [6] shows that for a combination of Wikipedia and YAGO, only queries that include text take siginificant time. Purely ontological queries finish within 2ms on average.

[5] `http://code.google.com/webtoolkit`

[6] `http://uima.apache.org/`

		#FP	#FN	Precision	Recall	F1	P@10	R-Prec	MAP	nDCG
SemSearch	sections	44,117	92	0.06	0.78	0.09	0.32	0.42	0.44	0.45
	sentences	1,361	119	0.29	0.75	0.35	0.32	0.50	0.49	0.50
	contexts	**676**	**139**	**0.39**	**0.67**	**0.43**†	**0.25**	**0.52**	**0.45**	**0.48**
Wikipedia lists	sections	28,812	354	0.13	0.84	0.21	0.46	0.38	0.33	0.41
	sentences	1,758	266	0.49	0.79	0.58	0.82	0.65	0.59	0.68
	contexts	**931**	**392**	**0.61**	**0.73**	**0.64***	**0.84**	**0.70**	**0.57**	**0.69**
TREC	sections	6,890	19	0.05	0.82	0.08	0.28	0.29	0.29	0.33
	sentences	392	38	0.39	0.65	0.37	0.58	0.62	0.46	0.52
	contexts	**297**	**36**	**0.45**	**0.67**	**0.46***	**0.58**	**0.62**	**0.46**	**0.55**

Table 2: Sum of false-positives and false-negatives and averages for other measures over all SemSearch, Wikipedia list and TREC queries for Broccoli when running on sections, sentences or contexts. For contexts, the results for the SemSearch and Wikipedia list benchmarks can be reproduced using our web application at `http://broccoli.informatik.uni-freiburg.de/repro-corr/` . $*$, $\dagger$ denotes a p-value < 0.02, < 0.003 for the two-tailed t-test against the sentences baseline.

7.5 Result quality

We performed an extensive quality evaluation using topics and relevance judgments from several standard benchmarking tasks for entity retrieval: the Yahoo SemSearch 2011 List Search Track [21], the TREC 2009 Entity Track [4] and, similarly as in [7], a random selection of ten Wikipedia featured *List of ...* pages. To allow reproducability we provide queries and relevance judgments as well as the possibilty to evaluate (and modify) the queries against a live running system for the SemSearch List Track and the Wikipedia lists at `http://broccoli.informatik.uni-freiburg.de/repro-corr/` . The TREC Entity Track queries were used for an in-depth quality evaluation that does not allow for an easy reproduction. Therefore we do not provide them in our reproducability web application. In the following we first describe each of the tasks in more detail.

The SemSearch 2011 List Search Track consisted of 50 topics asking for lists of entities in natural language, e.g. *Apollo astronauts who walked on the Moon*. The publicly available results were created by pooling the results of participating systems and are partly incomplete. Furthermore, the task used a subset of the Billion Triple Challenge Linked Data as collection, and some of the results referenced the same entity several times, e.g. once in DBPedia and once in OpenCyc. Therefore, we manually created a new ground truth consisting of Wikipedia entities. This is possible because most topics were inspired by Wikipedia lists and can be answered completely by manual investigation. Three of the topics did not contain any result entities in Wikipedia, and we ignored one additional topic because it was too controversial to answer with certainty (*books of the Jewish canon*). This leaves us with 46 topics and a total of 384 corresponding entities in our ground truth[7] . The original relevance judgments only had 42 topics with primary results and 454 corresponding entities, including many duplicates.

The TREC 2009 Entity Track worked with the ClueWeb09 collection and consisted of 20 topics also asking for lists of entities in natural language, e.g. *Airlines that currently use Boeing 747 planes*, but in addition provided the source entity (*Boeing 747*) and the type of the target entity (*organization*). We removed all relevance judgments for pages that were not contained in the English Wikipedia; this approach was taken before in [11] as well. This leaves us with 15 topics and a total of 140 corresponding relevance judgments.

As third benchmark we took a random selection of ten of Wikipedia's over 2,400 manually compiled featured `en.wikipedia.org/wiki/List_of_...` pages[8], e.g. the *List of participating nations at the Winter Olympic Games*. Wikipedia lists are manually compiled by humans, but actually they are answers to semantic queries, and therefore perfectly suited for a system like ours. In addition, the featured Wikipedia lists undergo a review process in the community, based on, besides other attributes, comprehensiveness. For our ground truth, we automatically extracted the list of entities from the Wikipedia list pages. This leaves us with 10 topics and a total of 2,367 corresponding entities in our ground truth[7].

For all of these tasks we manually generated queries in our query language corresponding to the semantics of the topics. We relied on using the interactive query suggestions of our user interface, but did not fine-tune our queries towards the results. An automatic translation from natural language to our query language is part of future work (see section 8). We want to stress that our goal is not a direct comparison to systems that participated in the tasks above. For that, input, collection and relevance judgments would have to be perfectly identical. Instead, we want to show that our system allows to construct intuitive queries that provide high quality results for these tasks.

We first evaluated the impact of our context decomposition from Section 3.3 (*contexts*) on result quality, by comparing it against two simple baselines: taking each sentence as one context (*sentences*) and taking each section as one context (*sections*). Table 2 shows that compared to sentences, our contexts decrease the (large) number of false-positives significantly for all benchmarks. For the TREC benchmark even the number of false-negatives decreases. This is the case because our document parser pre-processes Wikipedia lists by appending each list item to the preceding sentence (before the SCI+SCR phase). These are the only types of contexts that cross sentence boundaries and a rare exception. For the Wikipedia list benchmark we verified that this technique did not cause any results that are in the lists from which we created the ground truth. Since the sentence level

[7] available at `http://broccoli.informatik.uni-freiburg.de/repro-corr/`

[8] `http://en.wikipedia.org/wiki/Wikipedia:Featured_lists`

does not represent a true superset of our contexts we also evaluated on the section level. We can observe a decrease in the number of false-negatives (a lot of them due to random co-occurrence of query words in a section) which does not outweigh the drastic increase of the number of false-positives. Overall, context decomposition results in a significantly increased precision and F-Measure, which confirms the positive impact on the user experience that we have observed.

Considering the ranking related measures in Table 2 we see a varying influence for the context based approach. The number of cases where ranking quality improves, remains unchanged or decreases is roughly balanced. This looks surprising, especially since the increase in F-measure is significant, but the reason is simple. So far our system uses simplistic ranks, determined by mere term frequency. We plan to improve on that in the future; see Section 8. We want to stress the following though. Most semantic queries, including all from the TREC and SemSearch benchmark, have a small set of relevant results. We believe that for such queries the quality of the result set as a whole is more important than the ranking within the result set. Still, for the TREC benchmark, R-precision on contexts is 0.62 and, for the SemSearch benchmark, mean average precision is 0.45. The best run from the TREC 2009 Entity Track when restricted to the English Wikipedia had an R-precision of 0.55 as reported in [11, Table 10]. The best result for the SemSearch List Search Track was a mean average precision of 0.279 [3]. Again, these results cannot be compared directly, but they do provide an indication of the quality and potential of our system.

7.6 Error analysis

To identify areas where our system can be improved we manually investigated the reasons for the false-positives and false-negatives when using contexts. We used the TREC benchmark for this, because it has a reasonable number of queries and relevance judgments that still allow a costly manual inspection of the results. We defined the following error categories. For false-positives: (FP1) a true hit which was *missing* from the ground truth; (FP2) the words in the context have a *different meaning* than what was intended by the query; (FP3) due to an error in the *ontology*; (FP4) a mistake in the *entity recognition*; (FP5) a mistake by the *parser*. (FP6) a mistake in our *context decomposition*. For false-negatives: (FN1) there seems to be *no evidence* for this entity in the Wikipedia based on the query we used. It is possible that the fact is present but expressed differently, e.g., by the use of synonyms of our query words; (FN2) the query elements are *spread* over two or more sentences; (FN3) a mistake in the *ontology*; (FN4) a mistake in the *entity recognition*; (FN5) a mistake by the *parser*; (FN6) a mistake in our *context decomposition*.

#FP	FP1	FP2	FP3	FP4	FP5	FP6
297	55%	11%	5%	12%	16%	1%

#FN	FN1	FN2	FN3	FN4	FN5	FN6
36	22%	6%	26%	21%	16%	8%

Table 3: Breakdown of errors by category.

Table 3 provides the percentage of errors in each of these categories. The high number in FP1 is great news for us: many entities are missing from the ground truth but were found by Broccoli. Errors in FN1 occur when full-text search with our queries on whole Wikipedia documents does not yield hits, independent from our contexts. Tuning queries or adding support for synonyms can decrease this number. FP2 and FN2 comprise the most severe errors. They contain false-positives that still match all query parts in the same context but have a different meaning and false-negatives that are lost because contexts are confined to sentence boundaries. Fortunately, both numbers are quite small.

The errors in categories FP and FN 3-5 depend on implementation details and third-party components. The high number in FN3 is due to errors in our current ontology, YAGO. A closer inspection revealed that, although the facts in YAGO are reasonably accurate, it is vastly incomplete in many areas (e.g., the *acted-in* relation contains only one actor for most movies). Preliminary experiments suggest that switching to Freebase [10] in the future will solve this and improve the results considerably (see section 8). To mitigate the errors caused by entity recognition and anaphora resolution (FP4+FN4), a more sophisticated state-of-the-art approach is easily integrated. Parse errors are harder. Assuming a perfect constituent parse for every single sentence, especially those with flawed grammar, is not realistic. Still, those errors do not expose limits of our approach. We hope to enable SCI+SCR without a full-parse in the future (see Section 8). The low number of errors due to our context decomposition (FP6+FN6) demonstrates that our current approach (Section 3.3) is already pretty good. Fine-tuning the way we decompose sentences might decrease this number even further.

Naturally, an evaluation should not treat entities missing in the ground-truth in the same way as actual errors. Table 4 provides quality measures for our benchmark based on sentences and contexts under three conditions: (*original*) evaluation based on the original TREC ground-truth; (*+missing*) with the entities from FP1 added to the ground truth; (*+correct*) with the errors leading to FP and FN 3,4,5 corrected.

		F1	P@10	R-Prec	MAP
Sentences	original	0.37	0.58	0.62	0.46
	+missing	0.55	0.77	0.76	0.60
Contexts	original	0.46	0.58	0.62	0.46
	+missing	0.65	0.79	0.77	0.62
	+correct	0.86	0.94	0.92	0.85

Table 4: Quality measures on TREC 2009 queries for three different levels of corrections.

The numbers for *+correct* show the high potential of our system and motivate further work correcting the respective errors. As argued in the discussion after Table 3, many corrections are easily applied, while some of them remain hard to correct perfectly.

8. CONCLUSIONS AND FUTURE WORK

We have presented Broccoli, a search engine for the interactive exploration of combined text and ontology data. We have described the index, the natural language processing,

and the user interface behind Broccoli. And we have provided reproducible evidence that Broccoli is indeed fast and gives search results of good quality.

So far, we have implemented all the basic ideas we deemed necessary to provide a convincing proof of concept. Based on this work, there are a lot of interesting directions for future research.

The underlying ontology plays a major role for our system. By switching from YAGO to Freebase we expect a great improvement of the overall quality through a better coverage of relations and thus proposals and results (see Tables 3 and 4 in the previous section). Our current approaches to entity recognition and anaphora resolution work well, but it might be possible to further improve result quality by incorporating more elaborate state-of-the-art approaches. This would also allow the system to be more easily applied to other collections than Wikipedia (our current heuristics rely on its structure, see Section 3.2). Integrating simple inference heuristics could help to reduce the number of errors that are caused by facts that are spread over several sentences. A high-quality sentence decomposition *without* the need for an expensive and error-prone full parse should further increase result quality. While query times are already low, optimized query processing and clever caching strategies have the potential to further improve speed. To investigate how to best approach performance and quality improvements, an evaluation of Broccoli on a larger, web-like collection should provide valuable insights. Automatically transforming natural language queries into our query language could help users that are accustomed to keyword queries in constructing their queries. Finally, a user study of our UI and the whole system is an important next step.

Acknowledgments

This work is partially supported by the DFG priority program Algorithm Engineering (SPP 1307) and by the German National Library of Medicine (ZB MED).

9. REFERENCES

[1] E. Agichtein and L. Gravano. *Snowball*: extracting relations from large plain-text collections. In *ACM DL*, pages 85–94, 2000.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, pages 722–735, 2007.

[3] K. Balog, M. Ciglan, R. Neumayer, W. Wei, and K. Nørvåg. Ntnu at semsearch 2011. In *Proc. of the 4th Intl. Semantic Search Workshop*, 2011.

[4] K. Balog, A. P. de Vries, P. Serdyukov, P. Thomas, and T. Westerveld. Overview of the TREC 2009 Entity Track. In *TREC*, 2009.

[5] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, pages 2670–2676, 2007.

[6] H. Bast and B. Buchhold. An index for efficient semantic full-text search. In *CIKM*, 2013.

[7] H. Bast, A. Chitea, F. M. Suchanek, and I. Weber. Ester: efficient search on text, entities, and relations. In *SIGIR*, pages 671–678, 2007.

[8] R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi, and D. Petrelli. Hybrid search: Effectively combining keywords and semantic searches. In *ESWC*, pages 554–568, 2008.

[9] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[10] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250, 2008.

[11] M. Bron, K. Balog, and M. de Rijke. Ranking related entities: components and analyses. In *CIKM*, pages 1079–1088, 2010.

[12] R. Chandrasekar, C. Doran, and B. Srinivas. Motivations and methods for text simplification. In *COLING*, pages 1041–1044, 1996.

[13] R. Collobert. Deep learning for efficient discriminative parsing. *Journal of Machine Learning Research - Proceedings Track*, 15:224–232, 2011.

[14] G. Giannopoulos, N. Bikakis, T. Dalamagas, and T. K. Sellis. Gontogle: A tool for semantic annotation and search. In *ESWC*, pages 376–380, 2010.

[15] F. Giunchiglia, U. Kharkevich, and I. Zaihrayeu. Concept search. In *ESWC*, pages 429–444, 2009.

[16] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted wikipedia search. In *BIS*, pages 1–11, 2010.

[17] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.

[18] J. Pound, P. Mika, and H. Zaragoza. Ad-hoc object retrieval in the web of data. In *WWW*, pages 771–780, 2010.

[19] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[20] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *J. Web Sem.*, 6(3):203–217, 2008.

[21] T. Tran, P. Mika, H. Wang, and M. Grobelnik. Semsearch'11: the 4th semantic search workshop. In *WWW (Companion Volume)*, 2011.

[22] H. Wang, Q. Liu, T. Penin, L. Fu, L. Zhang, T. Tran, Y. Yu, and Y. Pan. Semplore: A scalable IR approach to search the web of data. *J. Web Sem.*, 7(3):177–188, 2009.

[23] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl. From keywords to semantic queries - incremental query construction on the semantic web. *J. Web Sem.*, 7(3):166–176, 2009.

Künstliche Intelligenz manuscript No.
(will be inserted by the editor)

A Quality Evaluation of KB+Text Search

Hannah Bast · Björn Buchhold · Elmar Haussmann

Received: date / Accepted: date

Abstract We provide a quality evaluation of KB+Text search, a deep integration of knowledge base search and standard full-text search. A knowledge base (KB) is a set of subject-predicate-object triples with a common naming scheme. The standard query language is SPARQL, where queries are essentially lists of triples with variables. KB+Text extends this search by a special *occurs-with* predicate, which can be used to express the co-occurrence of words in the text with mentions of entities from the knowledge base. Both pure KB-search and standard full-text search are included as special cases.

We evaluate the result quality of KB+Text search on three different query sets. The corpus is the the full version of the English Wikipedia (40 GB XML dump) combined with the YAGO knowledge base (26 million triples). We provide a web application to reproduce our evaluation, which is accessible via `http://ad.informatik.uni-freiburg.de/publications`.

Keywords Knowledge Bases · Semantic Search · KB+Text Search · Quality Evaluation

1 Introduction

KB+Text search combines structured search in knowledge bases with traditional full-text search.

Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
Tel.: +49 761 203-8163
E-mail: {bast,buchhold,haussmann}@cs.uni-freiburg.de

In traditional full-text search, the data consists of text documents. The user types a (typically short) list of keywords and gets a list of documents containing some or all of these keywords, hopefully ranked by some notion of relevance to your query. For example, typing *broccoli leaves edible* in a web search engine will return lots of web pages with evidence that broccoli leaves are indeed edible.

In knowledge base search, the data is a knowledge base, typically given as a (large) set of subject-predicate-object triples. For example, *Broccoli is-a plant* or *Broccoli native-to Europe*. These triples can be thought of to form a graph of entities (the nodes) and relations (the edges), and a language like SPARQL allows to search for subgraphs matching a given pattern. For example, find all plants that are native to Europe.

The motivation behind KB+Text search is that many queries of a more "semantic" nature require the combination of both approaches. For example, consider the query *plants with edible leaves and native to Europe*, which will be our running example in this paper. A satisfactory answer for this query requires the combination of two kinds of information:

(1) A list of plants native to Europe; this is hard for full-text search but a showcase for knowledge base search.

(2) For each plant the information whether its leaves are edible or not; this is easily found with a full-text search for each plant, but quite unlikely to be contained in a knowledge base.

In a previous work [4], we have developed a system with a convenient user interface to construct such

queries incrementally, with suggestions for expanding the query after each keystroke. We named the system Broccoli, after a variant of the example query above, which was our very first test query. Figure 1 shows a screenshot of Broccoli in action for this example query.

1.1 Our contribution

The main contribution of this paper is a quality evaluation of KB+Text search on three benchmarks, including a detailed error analysis; see Section 4. On the side, we recapitulate the basics of KB+Text search (Section 2) and we provide a brief but fairly broad overview of existing quality evaluations for related kinds of "semantic search" (Section 3).

2 The Components of KB+Text Search

We briefly describe the main components of a system for KB+Text search, as far as required for understanding the remainder of this paper. KB+Text search operates on two kinds of inputs, a text collection and a knowledge base. The text collection consists of documents containing plain text. The knowledge base consists of entities and their relations in the form of triples. This input is pre-processed, indexed, and then queried as follows.

2.1 Entity recognition

In a first step, mentions of entities from the given knowledge base are recognized in the text. Consider the following sentence:

(S) *The usable parts of rhubarb are the medicinally used roots and the edible stalks, however its leaves are toxic.*

Assuming the provided knowledge base contains the entity *Rhubarb*, the words *rhubarb* and *its* are references to it. When we index the English Wikipedia and use YAGO as a knowledge base, we make use of the fact that first occurrences of entities in Wikipedia documents are linked to their Wikipedia page that identifies a YAGO entity. Whenever a part or the full name of that entity is mentioned again in the same section of the document (for example, *Einstein* referring to *Albert Einstein*), we recognize it as that entity. We resolve references (anaphora) by assigning each occurrence of *he*, *she*, *it*, *her*, *his*, etc. to the last recognized entity of matching gender. For text without Wikipedia annotations, state-of-the art approaches for named entity recognition and disambiguation, such as Wikify! [20], can be used instead.

2.2 Text segmentation

The special *occurs-with* relation searches for co-occurren of words and entities as specified by the respective arc in the query; see Figure 1 and 2.4 below. We identify segments in the input text to which co-occurrence should be restricted. Identifying the ideal scope of these segments is non-trivial and we experiment with three settings: (1) complete sections, (2) sentences and (3) *contexts*, which are defined as sets of words that "belong together" semantically. The contexts for our example sentence (S) from above are:

(C1) *The usable parts of rhubarb are the medicinally used roots*
(C2) *The usable parts of rhubarb are the edible stalks*
(C3) *however rhubarb leaves are toxic*

Note that, because entities and references (underlined words) have been identified beforehand, no information is lost. The rationale behind contexts is to make the search more precise and "semantic". For example, we would not want *Rhubarb* to be returned for a query for *plants with edible leaves*, since its leaves are actually toxic. Nevertheless *Rhubarb*, *edible*, and *leaves* co-occur in sentence (S) above. However, they do not co-occur in either of (C1), (C2), (C3). To compute contexts, we follow an approach for Open Information Extraction (OIE) described in [8].

2.3 Indexing

An efficient index for KB+Text search is described in [5]. This index provides exactly the support needed for the system shown in Figure 1: efficient processing of tree-shaped KB+Text queries (without variables for relations), efficient excerpt generation, and efficient search-as-you-type suggestions that enable a fully interactive incremental query construction.

2.4 Query language

KB+Text extends classic KB-search by the special *occur with* predicate. This predicate can be used to specify co-occurrence of a class (e.g., plant) or instance (e.g.,

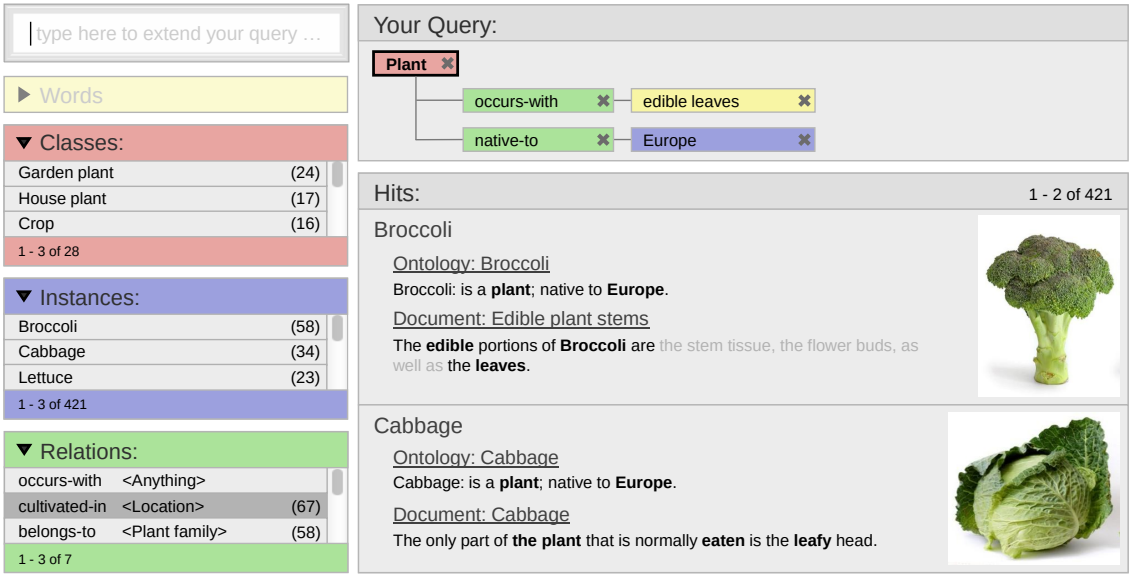

Fig. 1 A screenshot of Broccoli, showing the final result for our example query. The box on the top right visualizes the current KB+Text query as a tree. The large box below shows the matching instances (of the class from the root node, plant in this case). For each instance, evidence is provided for each part of the query. In the panel on the left, instances are entities from the knowledge base, classes are groups of entities with the same object according to the *is-a* predicate, and relations are predicates. The instances are ranked by the number of pieces of evidence (only a selection of which are shown). With the search field on the top left, the query can be extended further. Each of the four boxes below the search field provide context-sensitive suggestions that depend on the current focus in the query. For the example query: suggestions for sub*classes* of plants, suggestions for *instances* of plants that lead to a hit, suggestions for *relations* to further refine the query. Word suggestions are displayed as soon as the user types a prefix of sufficient length. These suggestions together with the details from the hits box allow the user to incrementally construct adequate queries without prior knowledge of the knowledge base or of the full text.

Broccoli) with an arbitrary combination of words, instances, and further sub-queries. When processing the query, this co-occurrence is restricted to the segments identified in the pre-processing step described in Section 2.2 above.

A user interface, like the one shown in Figure 1, guides the user in incrementally constructing queries from this language. In particular, a visual tree-like representation of the current query is provided after each keystroke, along with hits for that query and suggestions for further extensions or refinements.

3 Related Work

The literature on semantic search technologies is vast, and "semantic" means many different things to different researchers. A variety of different and hard-to-compare benchmarks have therefore emerged, as well as various stand-alone evaluations of systems that perform KB+Text or variants of it.

We briefly discuss the major benchmarks from the past decade, as well as the relatively few systems that explicitly combine full-text search and knowledge base search. A more comprehensive overview over the field of semantic search on text and knowledge bases is provided in [6].

3.1 TREC entity tracks

The goal of the TREC Entity Tracks were queries searching for lists of entities, just like in our KB+Text search. They are particularly interested in lists of entities that are related to a given entity in a specific way. Thus, the task is called "related entity finding". A typical query is *airlines that currently use boeing 747 planes*. Along

with the query, the central entity (*boeing 747*) as well as the the type of the desired target entities (*airlines*) were given.

For the 2009 Entity Track [3], the underlying data was the ClueWeb09 category B collection. ClueWeb09[1] is a web corpus consisting of 1 billion web pages, of which 500 million are in English. The category B collection is a sub-collection of 50 million of the English pages. Runs with automatic and manual query construction were evaluated. This task turned out to be very hard, and the overall best system achieved an NDCG@R of 31% and a P@10 of only 24% - albeit with automatic query construction. When restricting the results to entities from Wikipedia, the best system achieved an NDCG@R of 22% and a P@10 of 45% [12]. We use the queries from this track as one of our benchmarks in Section 4 (for later tracks no Wikipedia based groundtruth is available).

For the 2010 Entity Track [1], the full English portion of the ClueWeb09 dataset was used (500 million pages). The task remained hard, with the best system achieving an NDCG@R of 37% and a R-Precision (P@10 was not reported that year) of 32% even for manually tuned queries (and 30% for automatic runs).

In 2010, an additional task was added, Entity List Completion (a similar task but with an additional set of example result entities given for each query) with BTC 2009[2] as the underlying dataset. This is a dataset consisting of 1.14 billion triples crawled from the semantic web. The BTC dataset contains the complete DBpedia [18]. It turned out that the best performing approaches all boost triples from DBpedia to obtain good results. Still, working with the dataset turned out difficult, with the best systems achieving and R-Precision of 31% (NDCG@R was not reported).

In the 2011 track [2], another semantic web dataset was used (Sindice 2011 [13]). However, the number of participating teams was very low, and results were disappointing compared to previous years.

3.2 SemSearch challenges

The task in the SemSearch challenges is also referred to as *ad-hoc object retrieval* [17] The user inputs free-form keyword queries, e.g. *Apollo astronauts who walked on the moon* or *movies starring Joe Frazier*. Results are ranked lists of entities. The benchmarks were run on BTC 2009 as a dataset.

In the 2010 challenge [17], there were 92 queries, each searching only for a single entity. The best system achieved a P@10 of 49% and a MAP of 19%.

In the 2011 challenge [10], there were 50 queries. The best system achieved a P@10 of 35% and a MAP of 28%. The 2011 queries are one of our benchmarks in Section 4.

3.3 The INEX Series

INEX (Initiative for the Evaluation of XML Retrieval) has featured many search tasks. While the focus is on XML retrieval, among others, two tracks are remarkably similar to the other benchmarks discussed here.

The Entity Ranking Track (from 2007 to 2009) and the Linked-Data Track (2012 and 2013) work on the text from Wikipedia and use intra-Wikipedia links to establish a connection between entities and an ontology or an entire knowledge base (since 2012, entities are linked to their representation in DBPedia). Queries are very similar to those of the TREC Entity Track from above: given a keyword query (describing a topic) and a category, find entities from that category relevant for that topic. However, few participants actually made use of linked data in their approaches and the results were inconclusive.

3.4 Question Answering

Question answering (QA) systems provide a functionality similar to KB+Text. The crucial difference is that questions can be asked in natural language (NL), which makes the answering part much harder. Indeed, the hardest part for most queries in the QA benchmarks is to "translate" the given NL query into a query that can be fed to the underlying search engine.

In the TREC QA tracks, which ran from 1999 to 2007, the underlying data were corpora of text documents. An overview of this long series of tracks is given in [15]. The corpora were mainly newswire documents, later also blog documents. The series started with relatively simple factoid questions, e.g. *Name a film in which Jude Law acted*, and ended with very complex queries based on sequences of related queries, including, e.g., temporal dependencies. For list questions, such as *Who are 6 actors who have played Tevye in 'Fiddler*

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

[2] BTC = billion triple challenge, `https://km.aifb.kit.edu/projects/btc-2009/`

on the Roof'?, which are similar to the kind we consider in this paper, the best system in 2007 achieved an F-measure of 48%.

In the QALD (Question Answering over Linked Data) series of benchmarks [19], the underlying data is again a large set of fact triples. The task is to generate the correct SPARQL query from a given NL question of varying difficulty, e.g. *Give me all female Russian astronauts* [14]. This is very different from the other benchmarks described above, where a perfect query (SPARQL or keyword) typically does not exist.

Various tracks used different sets of facts triples from DBpedia and MusicBrainz (facts about music). In the last two runs, QALD-4 [24] and QALD-5 [25], the best system achieved an an F-measure of 72% and 63%, respectively.

3.5 Systems for KB+Text and similar paradigms

Systems for a combined search in text documents and knowledge bases were previously proposed in [7] (ESTER), [9] (Hybrid Search), [22] (Mímir), [26] (Semplore), and [16] (Concept Search). None of these systems consider semantic context as described in Section 2.2. For all these systems, only a very limited quality evaluation has been provided.

Hybrid Search is evaluated on a very specialized corpus (18K corporate reports on jet engines by Rolls Royce). For Concept Search, a similarly small dataset of 29K documents constructed from the DMoz web directory.

For ESTER, only two simple classes of queries are evaluated: *people associated with <university>* and *list of counties in <US state>*. A precision of 37% and 67%, respectively, is reported for each class.

Semplore is evaluated on a combination of DBpedia (facts from Wikipedia) and LUBM (an ontology for the university domain). A P@10 of more than 80% is reported for 20 manually constructed queries. For many of those, the text part is simply keyword search in entity names, e.g., all awards matching the keywords *nobel prize*. Those queries then trivially have perfect precision and recall. We have only a single such query in our whole quality evaluation, all other queries combine knowledge base and full-text search in a non-trivial manner.

Mímir is only evaluated with respect to query response times and in a user study where users were asked to perform four search tasks. For these tasks, success and user satisfaction with the system were tracked.

4 Evaluation

4.1 Input data

The text part of our data is all documents in the English Wikipedia, obtained via `download.wikimedia.org` in January 2013.[3] Some dimensions of this collection: 40 GB XML dump, 2.4 billion word occurrences (1.6 billion without stop-words), 285 million recognized entity occurrences and 200 million sentences which we decompose into 418 million contexts.

As knowledge base we used YAGO from October 2009.[4] We manually fixed 92 obvious mistakes in the KB (for example, the *noble prize* was a *laureate* and hence a *person*), and added the relation *Plant native-in Location* for demonstration purposes. Altogether our variant of YAGO contains 2.6 million entities, 19,124 classes, 60 relations, and 26.6 million facts.

We build a joint index over this full text and this KB, as described in [5]. As described there, the resulting index file has a size of 14 GB with query times typically well below 100 ms [5, Table 1].

4.2 Query Benchmarks

We evaluated the quality of our KB+Text search on the dataset just described on three query benchmark. Each benchmark consists of a set of queries, and for each query the set of relevant entities for that query on the dataset above. Two of these query benchmarks are from past entity search competitions, described in Section 3: the Yahoo SemSearch 2011 List Search Track [23], and the TREC 2009 Entity Track [3]. The third query benchmark is based on a random selection of ten Wikipedia featured *List of ...* pages, similarly as in [7].

To allow reproducibility, we provide queries and relevance judgments as well as the possibility to evaluate (and modify) the queries against a live running system for the SemSearch List Track and the Wikipedia lists under `http://broccoli.informatik.uni-freiburg.de` . The TREC Entity Track queries were used for an in-depth quality evaluation that does not allow for an easy reproduction. Therefore we do not provide them in our reproducibility web application.

[3] We chose this slightly outdated version for technical reasons. The corresponding Wikimedia data from 2017 is (only) about 50% larger but otherwise has the same characteristics and would not lead to principally different results.

[4] There is a more recent version, called YAGO2, but the additions from YAGO to YAGO2 are not really interesting for our search.

Table 1 Sum of false-positives and false-negatives and averages for other measures over all SemSearch, Wikipedia list and TREC queries for the evaluated system when running on sections, sentences or contexts. The $*$ and † denote a p-value of < 0.02 and < 0.003, respectively, for the two-tailed t-test compared to the figures for sentences.

		#FP	#FN	Prec.	Recall	F1	R-Prec	MAP	nDCG
SemSearch	sections	44,117	92	0.06	0.78	0.09	0.32	0.42	0.44
	sentences	1,361	119	0.29	0.75	0.35	0.32	0.50	0.49
	contexts	**676**	**139**	**0.39**	**0.67**	**0.43†**	**0.52**	**0.45**	**0.48**
WP lists	sections	28,812	354	0.13	0.84	0.21	0.38	0.33	0.41
	sentences	1,758	266	0.49	0.79	0.58	0.65	0.59	0.68
	contexts	**931**	**392**	**0.61**	**0.73**	**0.64***	**0.70**	**0.57**	**0.69**
TREC	sections	6,890	19	0.05	0.82	0.08	0.29	0.29	0.33
	sentences	392	38	0.39	0.65	0.37	0.62	0.46	0.52
	contexts	**297**	**36**	**0.45**	**0.67**	**0.46***	**0.62**	**0.46**	**0.55**

The SemSearch 2011 List Search Track consists of 50 queries asking for lists of entities in natural language, e.g. *Apollo astronauts who walked on the Moon*. The publicly available results were created by pooling the results of participating systems and are partly incomplete. Furthermore, the task used a subset of the BTC dataset (see Section 3), and some of the results referenced the same entity several times, e.g., once in DBpedia and once in OpenCyc. Therefore, we manually created a new ground truth consisting only of Wikipedia entities (compatible with out dataset). This was possible because most topics were inspired by Wikipedia lists and can be answered completely by manual investigation. Three of the topics did not contain any result entities in Wikipedia, and we ignored one additional topic because it was too controversial to answer with certainty (*books of the Jewish canon*). This leaves us with 46 topics and a total of 384 corresponding entities in our ground truth. The original relevance judgments only had 42 topics with primary results and 454 corresponding entities, including many duplicates.

The TREC 2009 Entity Track worked with the ClueWeb09 collection and consisted of 20 topics also asking for lists of entities in natural language, e.g. *Airlines that currently use Boeing 747 planes*, but in addition provided the source entity (*Boeing 747*) and the type of the target entity (*organization*). We removed all relevance judgments for pages that were not contained in the English Wikipedia; this approach was taken before in [12] as well. This leaves us with 15 topics and a total of 140 corresponding relevance judgments.

As a third benchmark we took a random selection of ten of Wikipedia's over 2,400 `en.wikipedia.org/wiki/List_of_...` pages[5]. For example, *List of participating nations at the Winter Olympic Games*. These lists are manually created by humans, but actually they are answers to semantic queries. The lists also tend to be fairly complete, since they undergo a review process in the Wikipedia community. This makes them perfectly suited for a quality evaluation of our system. For the ground truth, we automatically extracted the list of entities from the Wikipedia list pages. This leaves us with 10 topics and a total of 2,367 corresponding entities in our ground truth.

For all of these tasks we manually generated KB+Tex queries corresponding to the intended semantics of the original queries. We relied on using the interactive query suggestions of the user interface, but did not fine-tune queries towards the results. We want to stress that our goal is not a direct comparison to systems that participated in the tasks above. For that, input, collection and relevance judgments would have to be perfectly identical. Instead, we want to evaluate whether KB+Text can provide high quality results for these tasks.

4.3 Quality results

Table 1 evaluates the precision and recall for all three benchmarks. As described in Section 2, the key component of our KB+Text search is the *occurs-with* relation, which searches for co-occurrences of the specified words / entities. We compare three segmentations for determining co-occurrence, as described in Section 2.2: *sections*, *sentences*, and semantic *contexts*.

[5] `http://en.wikipedia.org/wiki/Wikipedia:Featured_lists`

Compared to sentences, semantic contexts decrease the (large) number of false-positives significantly for all three benchmarks.[6] Using co-occurence on the section level[7], we can observe a decrease in the number of false-negatives (a lot of them due to random co-occurrence of query words in a section). However, this does not outweigh the drastic increase of the number of false-positives. Overall, semantic contexts yield the best precision on all three benchmarks, and also the best F-measure. This confirms the positive impact on the user experience that we have observed.

Note that Table 1 does not include any ranking-related measures, like precision at 10 (P@10) or mean average precision (MAP). This is a common procedure in knowledge base search, where queries (typically formulated in SPARQL) deliver result sets, in no particular expected order; see [19] for a survey of benchmarks. Indeed, when you consider queries like *nicole kidmann siblings* or *continents in the world*, it's primarily about the set, not about the order. Also note that most semantic queries, including all from the TREC and Sem-Search benchmark, have a small set of relevant results. Nevertheless, in the TREC entity search benchmark, expected results were ranked lists of entities.[8] In our detailed analysis of this benchmark in Section 4.4 below, we therefore also provide measures for a simplistic ranking we have implemented. We simply ordered entities by the number of matching segments (= the snippets displayed in the screenshot of Figure 1).

4.4 Error analysis

KB+Text search, as described in Section 2 is a complex task, with many potential sources for errors. For the TREC benchmark, using contexts as segments, we manually investigated the reasons for the false-positives and false-negatives. We defined the following error categories.

For false-positives:

(FP1) a true hit was *missing* from the ground truth;
(FP2) the context has the *wrong meaning*[9];
(FP3) due to an error in the *knowledge base*;
(FP4) a mistake in the *entity recognition*;
(FP5) a mistake by the *parser*[10];
(FP6) a mistake in *computing contexts*.

For false-negatives:

(FN1) there seems to be *no evidence* for this entity in Wikipedia based on the query we used (the fact might be present but expressed using different words);
(FN2) the query elements are *spread* over two or more sentences;
(FN3) a mistake in the *knowledge base*;
(FN4) a mistake in the *entity recognition*;
(FN5) a mistake by the *parser* (analogous to FP5);
(FN6) a mistake in *computing contexts*.

Table 2 Breakdown of all errors (false-positives and false-negatives) by category.

#FP	FP1	FP2	FP3	FP4	FP5	FP6
297	55%	11%	5%	12%	16%	1%
#FN	FN1	FN2	FN3	FN4	FN5	FN6
36	22%	6%	26%	21%	16%	8%

Table 2 provides the percentage of errors in each of these categories. The high number in FP1 is great news for us: many entities are missing from the ground truth but were found by the system. Errors in FN1 occur when full-text search with our queries on whole Wikipedia documents does not yield hits, independent from semantic contexts. Tuning queries or adding support for synonyms can decrease this number. FP2 and FN2 comprise the most severe errors. They contain false-positives that still match all query parts in the same context but have a different meaning and false-negatives that are lost because contexts are confined to sentence boundaries. Fortunately, both numbers are quite small.

The errors in categories FP and FN 3-5 depend on implementation details and third-party components.

[6] For the TREC benchmark even the number of false-negatives decreases. This is because when segmenting into contexts the document parser pre-processes Wikipedia lists by appending each list item to the preceding sentence. These are the only types of contexts that cross sentence boundaries and a rare exception. For the Wikipedia list benchmark we verified that this technique did not cause any results that are in the lists from which we created the ground truth.

[7] Sections are indeed a super-set of semantic contexts.

[8] This saved participants of this benchmark the trouble of providing a cut-off value for what to include in the result and what not.

[9] This means, that the words occur in the context (otherwise this would not be a bit), but with a meaning different from what was intended by the query.

[10] The sentence parse are required to compute contexts.

Table 3 Quality measures for the TREC benchmark for the *orig*inal ground truth, with *miss*ing relevant entities, and with errors from categories FP and FN 3,4,5 *corr*ected.

	Prec.	Recall	F1	P@10	R-Prec	MAP	nDCG
TREC Entity Track, best	n/a	n/a	n/a	0.45	0.55	n/a	0.22
KB+Text, orig	0.45	0.67	0.46	0.58	0.62	0.46	0.55
KB+Text, orig + miss	0.67	0.73	0.65	0.79	0.77	0.62	0.70
KB+Text, orig + miss + corr	0.88	0.89	0.86	0.94	0.92	0.85	0.87

The high number in FN3 is due to errors in the used knowledge base, YAGO. A closer inspection revealed that, although the triples in YAGO are reasonably accurate, it is vastly incomplete in many areas. For example, the *acted-in* relation contains only one actor for most movies. This could be mitigated by switching to a more comprehensive knowledge base like Freebase [11]; indeed, our latest demo of Broccoli is using Freebase instead of YAGO [4]. To mitigate the errors caused by entity recognition and anaphora resolution (FP4+FN4), a more sophisticated state-of-the-art approach is easily integrated. Parse errors are harder. The current approach for determining contexts heavily relies on the output of a state-of-the art constituent parser. Assuming a perfect parse for every single sentence, especially those with flawed grammar, is not realistic. Still, those errors do not expose limits of KB+Text search with semantic contexts. The low number of errors due to the context computation (FP6+FN6) demonstrates that the current approach (Section 2.2) is already pretty good. Fine-tuning the way we decompose sentences might decrease this number even further.

Table 3 provides an updated evaluation, with all the errors induced by "third-party" components (namely FP and FN 3,4,5) corrected. The last row shows the high potential of KB+Text search and motivates further work correcting the respective errors. As argued in the discussion after Table 2, many corrections are easily applied, while some of them remain hard to correct perfectly.

The first line of Table 3 shows the best results from the TREC 2009 Entity Track (TET09), when restricted to entities from the English Wikipedia; see [12, Table 10]. There are a few things to note in this comparison. First, TET09 used the ClueWeb09 collection, category B. However, that collection contains the English Wikipedia, and participants were free to restrict their search to that part only. Indeed, the best systems strongly boosted results from Wikipedia. Second, results for TET09 were not sets but ranked lists of entities, hence absolute precision and recall figures are not available. Our results are for the simplistic ranking explained above. Third, we created our queries manually, as described at the end of Section 4.2 above. However, TET09 also permitted manually constructed queries, but those results were not among the best. Fourth, the ground truth was approximated via *pooling* results from the then participating systems [3]. This is a disadvantage for systems that are evaluated later on the same ground truth [21]. Still, our quality results are better even on the original ground truth, and much better with missing entities (FP1) added.

5 Conclusions and Future Work

We have evaluated the quality of KB+Text search on three benchmarks, with very promising results. A detailed error analysis has pointed out the current weak spots: missing entities in the knowledge base, missing evidence in the full text, errors in the entity recognition, errors in the full parses of the sentences. Promising directions for future research are therefore: switch to a richer knowledge base (e.g., Freebase), switch to a larger corpus than Wikipedia (e.g., ClueWeb), develop a more sophisticated entity recognition, try to determine semantic context without full parses.

References

1. Balog, K., Serdyukov, P., de Vries, A.P.: Overview of the TREC 2010 Entity Track. In: TREC (2010)
2. Balog, K., Serdyukov, P., de Vries, A.P.: Overview of the TREC 2011 Entity Track. In: TREC (2011)
3. Balog, K., de Vries, A.P., Serdyukov, P., Thomas, P., Westerveld, T.: Overview of the TREC 2009 Entity Track. In: TREC (2009)
4. Bast, H., Bäurle, F., Buchhold, B., Haußmann, E.: Semantic full-text search with broccoli. In: SIGIR, pp. 1265–1266. ACM (2014)

5. Bast, H., Buchhold, B.: An index for efficient semantic full-text search. In: CIKM (2013)
6. Bast, H., Buchhold, B., Haussmann, E.: Semantic search on text and knowledge bases. Foundations and Trends in Information Retrieval **10**(2-3), 119–271 (2016). DOI 10.1561/1500000032. URL http://dx.doi.org/10.1561/1500000032
7. Bast, H., Chitea, A., Suchanek, F.M., Weber, I.: Ester: efficient search on text, entities, and relations. In: SIGIR, pp. 671–678 (2007)
8. Bast, H., Haussmann, E.: Open information extraction via contextual sentence decomposition. In: ICSC (2013)
9. Bhagdev, R., Chapman, S., Ciravegna, F., Lanfranchi, V., Petrelli, D.: Hybrid search: Effectively combining keywords and semantic searches. In: ESWC, pp. 554–568 (2008)
10. Blanco, R., Halpin, H., Herzig, D.M., Mika, P., Pound, J., Thompson, H.S., Duc, T.T.: Entity search evaluation over structured web data. In: SIGIR Workshop on Entity-Oriented Search (JIWES) (2011)
11. Bollacker, K.D., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: SIGMOD, pp. 1247–1250 (2008)
12. Bron, M., Balog, K., de Rijke, M.: Ranking related entities: components and analyses. In: CIKM, pp. 1079–1088 (2010)
13. Campinas, S., Ceccarelli, D., Perry, T.E., Delbru, R., Balog, K., Tummarello, G.: The sindice-2011 dataset for entity-oriented search in the web of data. In: Workshop on Entity-Oriented Search (EOS), pp. 26–32 (2011)
14. Cimiano, P., Lopez, V., Unger, C., Cabrio, E., Ngomo, A.C.N., Walter, S.: Multilingual question answering over linked data (QALD-3): Lab overview. In: CLEF, pp. 321–332 (2013)
15. Dang, H.T., Kelly, D., Lin, J.J.: Overview of the TREC 2007 Question Answering Track. In: TREC (2007)
16. Giunchiglia, F., Kharkevich, U., Zaihrayeu, I.: Concept search. In: ESWC, pp. 429–444 (2009)
17. Halpin, H., Herzig, D.M., Mika, P., Blanco, R., Pound, J., Thompson, H.S., Tran, D.T.: Evaluating ad-hoc object retrieval. In: Workshop on Evaluation of Semantic Technologies (WEST) (2010)
18. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web **6**(2), 167–195 (2015). DOI 10.3233/SW-140134. URL http://dx.doi.org/10.3233/SW-140134
19. Lopez, V., Unger, C., Cimiano, P., Motta, E.: Evaluating question answering over linked data. J. Web Sem. **21**, 3–13 (2013)
20. Mihalcea, R., Csomai, A.: Wikify! Linking documents to encyclopedic knowledge. In: CIKM, pp. 233–242 (2007)
21. Sanderson, M.: Test collection based evaluation of information retrieval systems. Foundations and Trends in Information Retrieval **4**(4), 247–375 (2010)
22. Tablan, V., Bontcheva, K., Roberts, I., Cunningham, H.: Mímir: An open-source semantic search framework for interactive information seeking and discovery. J. Web Sem. **30**, 52–68 (2015). URL http://www.sciencedirect.com/science/article/pii/S1570826814001036
23. Tran, T., Mika, P., Wang, H., Grobelnik, M.: Semsearch'11: the 4th Semantic Search Workshop. In: WWW (Companion Volume) (2011)
24. Unger, C., Forascu, C., López, V., Ngomo, A.N., Cabrio, E., Cimiano, P., Walter, S.: Question answering over linked data (QALD-4). In: Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014., pp. 1172–1180 (2014). URL http://ceur-ws.org/Vol-1180/CLEF2014wn-QA-UngerEt2014.pdf
25. Unger, C., Forascu, C., López, V., Ngomo, A.N., Cabrio, E., Cimiano, P., Walter, S.: Question answering over linked data (QALD-5). In: Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum, Toulouse, France, September 8-11, 2015. (2015). URL http://ceur-ws.org/Vol-1391/173-CR.pdf
26. Wang, H., Liu, Q., Penin, T., Fu, L., Zhang, L., Tran, T., Yu, Y., Pan, Y.: Semplore: A scalable IR approach to search the web of data. J. Web Sem. **7**(3), 177–188 (2009)