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

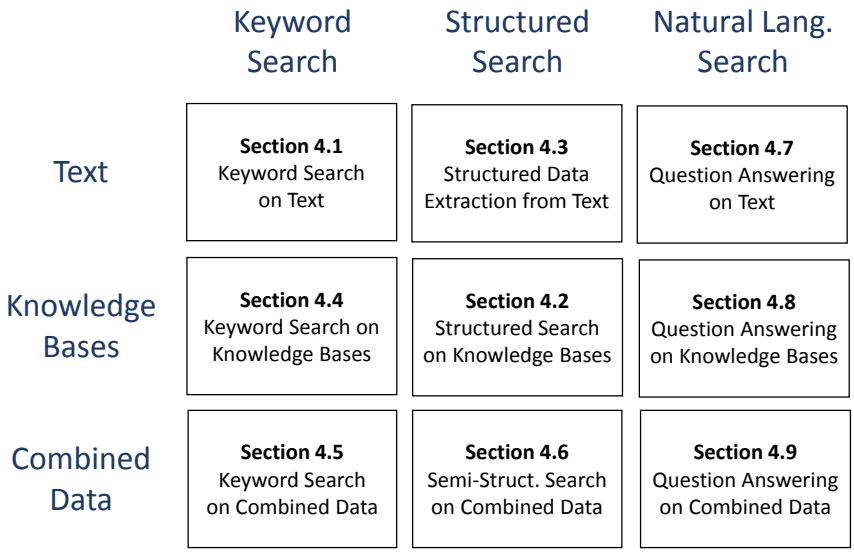

Figure 1.1: Our basic classification of research on semantic search by underlying data (rows) and search paradigm (columns). The three data types are explained in Section 2.1, the three search paradigms are explained in Section 2.2. Each of the nine groups is discussed in the indicated subsection of 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 $s = (w_1, \ldots w_n)$ and a set of available POS tags T, POS tagging outputs a sequence $t_1, t_2, \ldots t_n$ 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 $s = (w_1, \ldots w_n)$ 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* $\rightarrow$ *house* or *hous*), synonyms (*search* $\leftrightarrow$ *retrieval*), error correction (*algoritm* $\leftrightarrow$ *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.