CONSTRUCTING LARGE PROPOSITION DATABASES

Peter Exner

June 10, 2011

Abstract

Using semantic parsing or related techniques, it is possible to extract knowledge from text in the form of predicate–argument structures. Such structures are often called propositions. With the advent of massive corpora such as Wikipedia, it has become possible to apply a systematic analysis of a wide range of documents covering a significant part of human knowledge and build large proposition databases from them.

While most approaches focus on shallow syntactic analysis and do not capture the full meaning of a sentence, semantic parsing goes deeper and discovers more information from text with a higher accuracy. This deeper analysis can be applied to discover temporal and location-based propositions from documents. Medical researchers could, for instance, discover articles regarding the interaction of bacteria in a specific body part.

Christensen et al. (2010) showed that using a semantic parser in information extraction can yield extractions with higher precision and recall in areas where shallow syntactic approaches have failed. This accuracy comes at a cost of parsing time. However, in the recent years, statistical parsing and especially semantic parsing have become increasingly accurate and efficient in analyzing text.

This Master's thesis describes the creation of multilingual proposition databases using generic semantic dependency parsing. Using a broad domain corpus, Wikipedia, we extracted, processed, clustered, and evaluated a large number of propositions. We built an architecture to provide a complete pipeline dealing with the input of text, extraction of knowledge, storage, and presentation of the resulting propositions. Furthermore, our system is able to handle large-scale extractions, wide domains, and multiple input languages. Wherever possible, the handling of information is automated such that manual labor is kept to a minimum.

Proposition databases like the one we constructed, combined with other lexical databases, are expected to be key components in semantic search technology, machine translation, and question and answer (Q&A) systems.

Sammanfattning

Genom användning av semantisk parsning eller relaterade tekniker, är det möjligt att utvinna kunskap från text i form av predikat – argument strukturer. Dessa strukturer kallas ofta för påståenden. Med intåget av massiva korpus som Wikipedia, har det blivit möjligt att tillämpa en systematisk analys av ett brett spektrum av dokumenter, som omfattar en stor del av den mänskliga kunskapen, och från dem bygga stora databaser innehållande påståenden.

Medan de flesta strategier fokuserar på ytlig syntaktisk analys och inte fångar den fulla innebörden av en mening, går semantisk tolkning djupare och upptäcker mer information från text och med en högre noggrannhet. Denna djupare analys kan tillämpas för att upptäcka tidsmässiga och platsbaserade påståenden från dokument. Medicinska forskare kan till exempel upptäcka artiklar som rör samverkan mellan bakterier i en viss kroppsdel.

Christensen et al. (2010) visade att en semantisk parser kan ge extraktioner med högre precision och täckning i områden där ytliga syntaktiska metoder misslyckats. Denna noggrannhet kommer till ett pris av längre parsningstid. Under de senaste åren, har däremot statistisk och särskilt semantisk parsing blivit allt mer noggrann och effektiv i att analysera text.

Detta examensarbete beskriver hur flerspråkliga påstående databaser kan skapas genom användning av generell semantiska parsning. Genom applicering på ett korpus av bred domän, Wikipedia, har vi utvunnit, bearbetat, klustrat, och utvärderat ett stort antal påståenden. Vi byggde en arkitektur för att tillhandahålla en komplett pipeline som hanterar inmatning av text, utvinning av kunskap, lagring och presentation av de resulterande påståendena. Dessutom kan vårt system hantera storskaliga extraktioner av breda domäner, och flera inmatningsspråk. När det så varit möjligt, har hanteringen av information automatiserats på ett sådant sätt att manuellt arbete hållts till ett minimum.

Databaser innehållande påståenden, som de vi skapat, kan tillsammans med andra lexikala databaser förväntas bli nyckelkomponenter i semantisk sökteknik, maskinöversättning och Q&A-system.

Acknowledgements

My utmost gratitude goes to my supervisor, Associate Professor Pierre Nugues, for his outstanding support, inspiration and guidance. The discussions I had with him were insightful and encouraging.

I specially thank Anders Björkelund for sharing his insights on parsing and programming.

I would like to say thanks to Associate Professor Anders Ardö for sharing his knowledge on Lucene.

My gratitude also goes to Joachim Hein and Jonas Lindemann at Lunarc for providing me with the supercomputing power and expertise needed to carry out this project.

My loving thanks to my family for their love and support throughout my life.

Above all, I thank my wife, Jie, for her unconditional support, encouragement and love.

Contents

Contents

1 Introduction **1**
1.1 Background 1
1.2 Aim of the Thesis 2

2 Semantic Parsing **3**
2.1 Filtering 3
2.2 Sentence Detection 4
2.3 Tokenization 5
2.4 Lemmatization 5
2.5 Part-of-Speech Tagging 5
2.6 Syntactic Dependency Parsing 6
2.7 Semantic Role Labeling 6

3 Previous Work **9**
3.1 TextRunner 9
3.2 SRL-IE 11
3.3 Discussion 12

4 Applications **13**
4.1 Information Extraction 13
4.2 IBM – Watson 13

5 Experimental Setup **17**
5.1 Semantic Parser 17
5.2 Data Source 17
5.3 Cooperation with Lunarc 18
5.4 Approach 19

6 System **21**
6.1 Parsing Framework 21
6.2 Proposition Database 25
6.3 Statistics Module 25

6.4 Frontend 27
6.5 Ranking Algorithm 30

7 Scaling-up **37**
7.1 Broad Domain Knowledge 37
7.2 Design Challenges 38

8 Discussion **41**

9 Conclusion **43**

10 Future Work **45**

Bibliography **47**

Chapter 1

Introduction

1.1 Background

Natural language processing (NLP) is a significant scientific challenge with the vision of having computers understanding and interacting in natural language. As people daily use language to exchange information, be it in a written or spoken form, it is only natural to expect that one day computers will be able to fully interact with human beings using natural language (Nugues, 2006). The uses of NLP are ever expanding and as it matures, it is incorporated in applications that help further science and business. Interactive online assistants, speech recognition, spell checking, search technology both for broad domains such as the web and narrow domains such as medical science are all areas, where NLP plays a key role.

Rather than storing knowledge as plain text, some NLP applications rely on having a large database of propositions stored in the shape of structured information. The proposition databases, combined with other lexical databases form the basis upon which systems for semantic search technology, machine translation, Q&A systems can be built (Ferrucci et al., 2010; Etzioni et al., 2004).

Such proposition databases are created through the process of parsing massive corpora such as Wikipedia. Some of these databases are obtained using only shallow syntactic analysis. However, this does not capture the full meaning of a text. Semantic parsing enables us to go deeper in analysis and discover more information with higher accuracy. This deeper analysis could be applied to discover temporal and location- based propositions from articles. As an example, a medical researcher could discover medical articles regarding the interaction of bacteria in a specific body part.

Christensen et al. (2010) showed that using a semantic parser in information extraction can yield extractions with higher precision and recall in areas where shallow syntactic approaches had failed. This accuracy comes at a cost of longer parsing time. However, in the recent years, statistical parsing and especially semantic parsing have become increasingly accurate and efficient in analyzing text.

This Master's thesis proposes to create multilingual proposition databases using generic semantic dependency parsing (Surdeanu et al., 2008). We will apply it on a broad domain corpus, Wikipedia, extract and evaluate a large number of propositions.

1.2 Aim of the Thesis

The creation of a proposition bank can be achieved manually by hand-annotating a corpus (Palmer et al., 2005) and also automatically by applying a parser (Banko et al., 2007). In this paper, we focus on creating a proposition database by using generic semantic dependency parsing. The system will provide a complete pipeline from the input of text, the extraction of knowledge, to storage and presentation of the extracted propositions. Furthermore, the system will be able to handle large-scale extractions, wide domains, and multiple input languages. Wherever possible, the handling of information will be automated such that manual labor will be kept to a minimum.

Wikipedia is a popular source for data mining featuring a broad set of domains and articles written in multiple languages, our goal is to parse it.

To achieve the goal of attaining a proposition bank with high-quality propositions, we will create a ranking algorithm that assigns a higher probability based on the redundancy of the propositions.

Our goals are thus to:

- Explore how semantic parsing can be scaled to process large heterogeneous corpora (i.e. corpora ranging from 100,000 to a few million articles).
- Construct a framework capable of large scale semantic parsing.
- Parse a substantial part of Wikipedia and create large, semantically annotated, and multilingual proposition databases.
- Create a ranking algorithm that extracts high-quality propositions.
- Construct an interface to query the proposition database.

Chapter 2

Semantic Parsing

Through years of training, we develop our skill of reading text, which allows us to distinguish characters, words, sentences, and to ultimately understand their meaning. In computer science, the analysis and subsequent extraction of structured information from text is called parsing. It is a computationally intensive procedure that at first glance might seem trivial, but requires careful and extensive analysis.

This chapter gives an overview of the steps involved in complete semantic parsing as performed on English and Chinese text. The steps are composed of filters and parsers that increasingly refine, add layers of information, and extract structure from text. Figure 2.1 shows an overview of the complete parsing pipeline for English MediaWiki text.

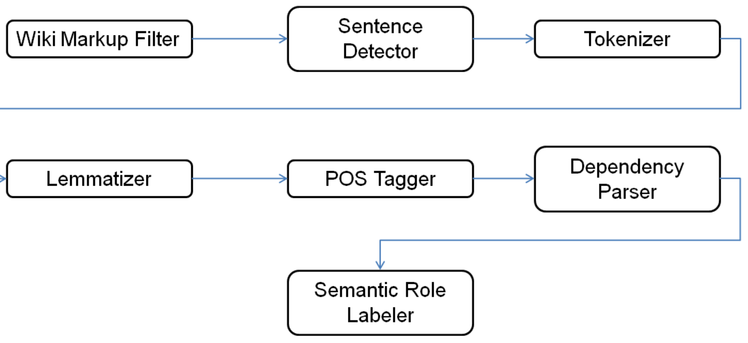


Figure 2.1: The complete parsing pipeline for English MediaWiki text

2.1 Filtering

Prior to any analysis, the text must be filtered. This is an essential step that seeks to remove annotations and markups without affecting the running text. Without this step, subsequent modules may fail in their analysis leading to erroneous extractions.

MediaWiki[1] text, as used in Wikipedia, is composed of text written in natural language annotated with a special markup called wikitext or wiki markup. It is a simple markup language that allows among other things the annotation of categories, templates, and hyperlinking to other MediaWiki pages. MediaWiki also allows the use of common HTML tags.

By filtering MediaWiki text, we aim at removing all annotations, sections that contain only links and references, and keeping only the running text. This process is difficult since the HTML syntax in many cases is invalid, most commonly, tags are left unclosed or are incorrectly nested. Filtering is further complicated by the fact that unwanted sections, such as 'See Also', go under many different names making them hard to locate and filter out. The decision falls between removing too little and accepting a certain amount of 'noise', or removing too much including all annotations and possibly even valuable text written in natural language.

To properly remove invalid HTML and wiki markup would require a project similar to building a full-fledged HTML parser as in a browser. Given the time constraints of this project, we instead chose a simpler approach and decided to use regular expressions for filtering. By using regular expressions, we limit the parsing scope to a certain subset of nested structures and therefore accept that a certain amount of noise will propagate through the parsing pipeline.

2.2 Sentence Detection

Sentence detection is the task of locating the beginnings and endings of sentences for the purpose of extraction. It is a nontrivial task, made difficult by the fact that boundary markers such as the period often have more than one purpose. For instance, the period is used to mark abbreviations, initials, and numbers. These situations are illustrated in the following examples

> *Today, www.EquityMarketsInc.com announced its research report highlighting Barnes & Noble, Inc. (NYSE: BKS) and Books-A-Million, Inc. (NASDAQ: BAMM).*

and

> *The Nasdaq composite rose 13.94 points, or 0.5 percent, to 2,796.86.*

We use the sentence detector from OpenNLP Tools[2] to divide text into sentences.

[1]http://www.mediawiki.org/
[2]http://incubator.apache.org/opennlp/

2.3 Tokenization

Following sentence detection, the sentences are divided into text elements called tokens through a process called tokenization. Tokens may represent words, numbers, abbreviations, punctuation, and also any other strings that mix characters and symbols (Nugues, 2006). A naive approach to tokenizing a sentence by considering tokens as being delimited by white spaces would fail. For instance, consider the sentence

The phone number to the post office is: (0046) 46 12 34 56.

If the tokens are delimited by white spaces, the phone number would be incorrectly broken into five tokens instead of only one.

In this project, we use the tokenizer from OpenNLP Tools to perform tokenization on English text. For tokenization of Chinese text, we use the Stanford Chinese Word Segmenter[3], version 2008-05-21.

2.4 Lemmatization

Lemmatization is the process of transforming a word from an inflected or derived form into its lexical or base form. For the purpose of parsing, the goal of lemmatization is to simplify the work for subsequent parsers by reducing the number of words that need to be analyzed. For instance, the words *running, runs,* and *ran* are all transformed into *run*. The lemmatizer used in this project comes from the mate-tools[4] framework.

2.5 Part-of-Speech Tagging

A part-of-speech tagger categorizes words into grammatical classes, such as nouns, verbs, pronouns, determiners etc. The words in a sentence are tagged, i.e. an extra layer of information is added. This layer is then used as input by subsequent parsers to carry out their analysis. As an example, the words in the sentence

$$He_{PRP}\ won_{VBD}\ the_{DT}\ Nobel_{NNP}\ Prize_{NNP}\ in_{IN}\ Literature_{NNP}\ in_{IN}\ 1954_{CD}.$$

have been tagged with their part-of-speech. We use the part-of-speech tagger from the mate-tools framework.

[3]http://nlp.stanford.edu/software/segmenter.shtml

[4]http://code.google.com/p/mate-tools/

2.6 Syntactic Dependency Parsing

During syntactic dependency parsing, the structure of a sentence is described by finding the syntactic links between the words. The links describe the syntactic relation between a head word and a dependent word. Syntactic dependency parsers build upon the information gathered from previous modules, such as lemmatizers and part of speech taggers to build the syntactic tree of a sentence. Figure 2.2 shows an example of a syntactic tree.

In this project, we use a dependency parser that features high accuracy and short parsing times (Bohnet, 2010) from the mate-tools framework.

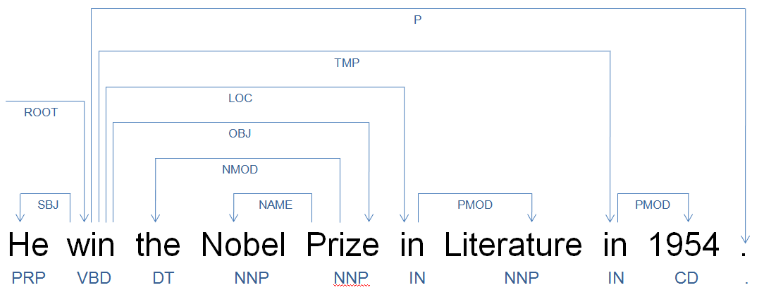

P
TMP
LOC
ROOT
OBJ
NMOD
SBJ
NAME
PMOD
PMOD
He win the Nobel Prize in Literature in 1954 .
PRP VBD DT NNP NNP IN NNP IN CD .

Figure 2.2: The syntactic relation between words in a sentence.

2.7 Semantic Role Labeling

Semantic role labeling (SRL) assumes that the meaning of a sentence is represented by predicates together with their arguments. During SRL, predicate-argument structures are extracted from sentences. SRL goes beyond extracting the subject and object for a predicate by identifying all the roles that arguments play in a sentence. Arguments may also include modifiers, such as temporal, locational, and manner adjuncts. Four steps are involved in the process: Predicate identification, predicate disambiguation, argument identification, and argument classification (Björkelund et al., 2010).

During predicate identification, a classifier determines if a noun or a verb is a predicate and identifies their possible sense. Predicate disambiguation is performed on all predicates that have multiple senses. Argument identification and classification, identify the corresponding arguments to a predicate and label them with their roles. Each predicate together with its arguments form a proposition. Figure 2.3 shows an example of semantic role labeling.

Predicates may have different senses together with a different set of arguments. In the example in Figure 2.3, the predicate is identified as win.01. This sense describes

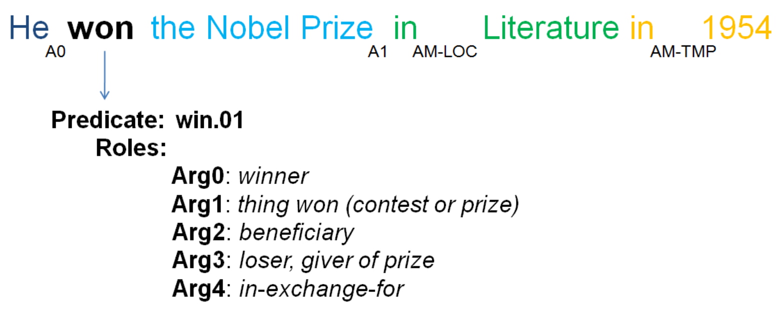

He won the Nobel Prize in Literature in 1954
A0
A1
AM-LOC
AM-TMP
Predicate: win.01
Roles:
Arg0: winner
Arg1: thing won (contest or prize)
Arg2: beneficiary
Arg3: loser, giver of prize
Arg4: in-exchange-for

Figure 2.3: An example of semantic role labeling, the yields of arguments are marked with different colours.

winning a prize. This differs from the predicate sense, win.02, which means ”causing a change of opinion”.

Arguments have a label and one word denoted as the head of the argument that projects a yield. For instance, in the example in Figure 2.3, the second argument has label *A1* describing the thing won and the head word *Prize* projecting the yield *the Nobel Prize*.

For the purpose of performing SRL, we use a multilingual semantic role labeler that obtained top scores in the CONLL-2009 shared task (Björkelund et al., 2009).

Chapter 3

Previous Work

Banko et al. (2007) recently introduced an information extraction paradigm, open information extraction (Open-IE), where new relations are automatically discovered independently of the domain and without the need of manual labor. Breaking with traditional information extraction (IE) methods that use domain-specific patterns and hand-annotated training data, Open-IE scales to large heterogeneous corpora such as the web.

This chapter examines two information extraction systems: TextRunner and SRL-IE, which employ Open-IE and traditional IE, respectively.

3.1 TextRunner

TextRunner is an Open-IE system that extracts relations from plain text (Banko et al., 2007). The output from the system are tuples of the form

$$t = (e_i, r_{ij}, e_j), \quad i < j$$

where e_i and e_j are arguments to the relation r_{ij}. For instance, given the sentence, *The Egyptians built pyramids*, TextRunner extracts the tuple (Egyptians, built, pyramids). TextRunner performs the extractions without having the patterns explicitly predefined for all relations. In this sense, it is domain independent and does not require any manual labor when shifting from one domain to another. TextRunner uses three key components: a **self-supervised learner**, a **single-pass extractor**, and a **redundancy-based assessor**.

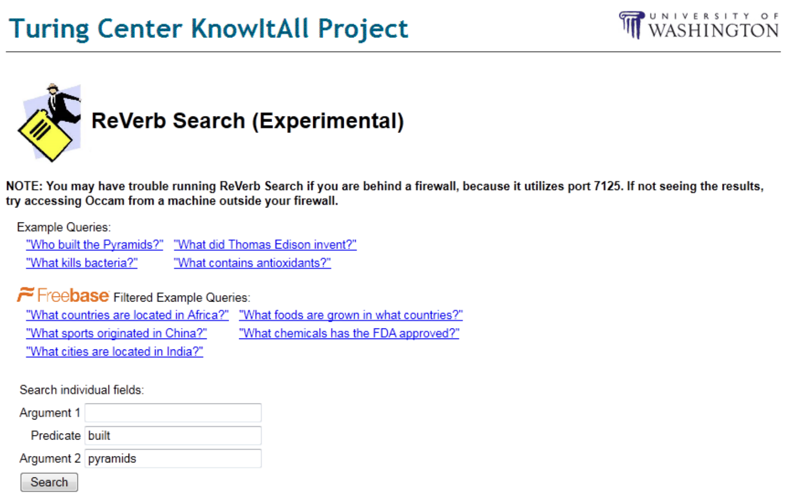
The **self-supervised learner** (SSL) is used before running a full-scale relation extraction. It uses a dependency parser to build a naive Bayes classifier, which is then used during the extraction phase. The parser creates dependency graphs from a corpora of several thousand sentences. By traversing these graphs and locating base noun phrases, the arguments, e_i and e_j, and subsequently the relation r_{ij} between them are identified. Given certain constraints that seek to filter out any errors made by the parser, the tuples are labeled either as a positive or a negative example. The labeled sets of tuples are mapped to a feature vector, which is then used in the training of the classifier. The features are selected in a way that keeps the classifier domain-independent. However, it is important to note that the classifier and subsequently the whole system is language-dependent.

During extraction, the **single-pass extractor** makes a single pass over the text and uses a noun-phrase chunker to extract noun phrases. The noun phrases together with the text are used to form candidate tuples, using heuristics, they are then filtered and presented to the classifier. If the classifier labels the candidate tuple as trustworthy, it is then stored in the system.

Following the extraction phase, the **redundancy-based assessor** uses a redundancy-based probabilistic model to assign probabilities to the extracted tuples. Tuples that occur multiple times in text are thus given a higher probability and regarded as correct extractions.

Banko et al. (2007) applied TextRunner to a corpus of 9 million web pages. Extraction was performed within 68 CPU hours, with each sentence taking on average 0.036 CPU seconds to process.

TextRunner creates a relation-centric inverted index using Lucene[1], allowing TextRunner to query the stored tuples. A web interface presents a way to create queries by specifing either of two arguments and the relation between them. Figure 3.1 shows how searching is performed in TextRunner. The user enters two arguments and the relation between. In this case searching for who built the pyramids was done by entering "built" as the predicate and "pyramids" as the second argument. Figure 3.2 shows the result of searching for the question.


Turing Center KnowItAll Project

UNIVERSITY OF WASHINGTON

ReVerb Search (Experimental)

NOTE: You may have trouble running ReVerb Search if you are behind a firewall, because it utilizes port 7125. If not seeing the results, try accessing Occam from a machine outside your firewall.

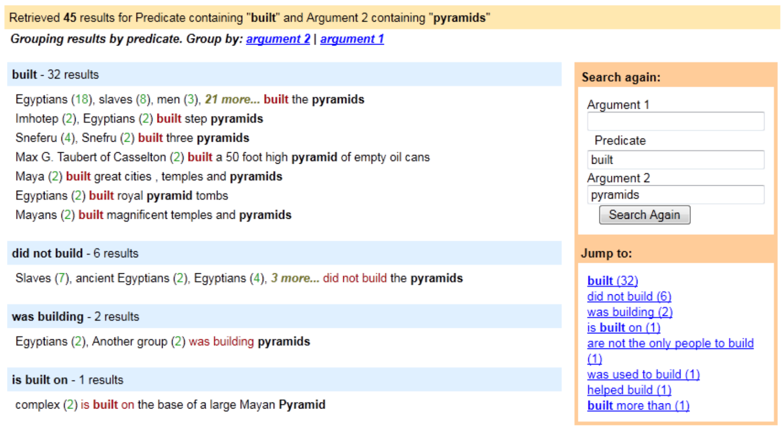
Example Queries:
"Who built the Pyramids?" "What did Thomas Edison invent?"
"What kills bacteria?" "What contains antioxidants?"

Freebase Filtered Example Queries:
"What countries are located in Africa?" "What foods are grown in what countries?"
"What sports originated in China?" "What chemicals has the FDA approved?"
"What cities are located in India?"

Search individual fields:
Argument 1
Predicate built
Argument 2 pyramids
Search

Figure 3.1: Searching in TextRunner.

[1]http://lucene.apache.org/



Retrieved **45** results for Predicate containing "**built**" and Argument 2 containing "**pyramids**"

Grouping results by predicate. Group by: ***argument 2*** | ***argument 1***

built - 32 results

Egyptians (18), slaves (8), men (3), ***21 more...*** **built** the **pyramids**
Imhotep (2), Egyptians (2) **built** step **pyramids**
Sneferu (4), Snefru (2) **built** three **pyramids**
Max G. Taubert of Casselton (2) **built** a 50 foot high **pyramid** of empty oil cans
Maya (2) **built** great cities , temples and **pyramids**
Egyptians (2) **built** royal **pyramid** tombs
Mayans (2) **built** magnificent temples and **pyramids**

did not build - 6 results

Slaves (7), ancient Egyptians (2), Egyptians (4), ***3 more...*** did not build the **pyramids**

was building - 2 results

Egyptians (2), Another group (2) was building **pyramids**

is built on - 1 results

complex (2) is **built** on the base of a large Mayan **Pyramid**

Search again:

Argument 1

Predicate
built

Argument 2
pyramids

Search Again

Jump to:

built (32)
did not build (6)
was building (2)
is **built** on (1)
are not the only people to build (1)
was used to build (1)
helped build (1)
built more than (1)

Figure 3.2: TextRunner search results for who built pyramids.

3.2 SRL-IE

With the creation of resources such as the Penn Treebank (Marcus et al., 1993) and more recently Propbank (Palmer et al., 2005), the area of statistical parsing has significantly advanced leading to semantic parsing capable of producing usable extractions.

The semantic role labeling information extraction (SRL-IE) system investigates the benefits of using traditional deep semantic analysis in large scale information extraction (Christensen et al., 2010). SRL-IE utilizes University of Illinois at Urbana-Champaign's semantic role labeler (Punyakanok et al., 2008) to extract predicate-argument relations from sentences. By mapping semantically labeled arguments to arguments in Open IE extractions, the relations are converted to relational tuples, a format comparable with TextRunner.

In a quantitative evaluation of SRL-IE and TextRunner, extractions for five target relations were compared in a dataset with redundant information consisting of 29,842 sentences. Extractions from both systems were tagged as either correct or error with a definition that favored neither of the systems. Christensen et al. (2010) showed that overall SRL-IE outperformed TextRunner in terms of higher precision and recall, the results are shown in Table 3.1. However, in terms of extraction, SRL-IE had over 2.5 orders of magnitude longer extraction time.

It is important to note that while SRL-IE uses a semantic parser to extract relations, semantic information such as argument roles is discarded in the conversion to relational tuples. It is thereby unclear if SRL-IE can harness the benefits of queries based on modifying arguments such as temporal or locational arguments.

	TextRunner			SRL-IE		
	Precision	Recall	F1	Precision	Recall	F1
Binary	51.9	27.2	35.7	64.4	85.9	73.7
N-ary	39.3	28.2	32.9	54.4	62.7	58.3
All	47.9	27.5	34.9	62.1	79.9	69.9
Time	6.3 minutes			52.1 hours		

Table 3.1: Comparison between TextRunner and SRL-IE.

3.3 Discussion

SRL-IE clearly shows that applying semantic parsing to information extraction yields high-quality extractions and is therefore a viable option. Compared to TextRunner, SRL-IE has a longer extraction time and might not be suitable to apply on web scale. However, in the two last years more high performing and efficient parsers have become available making it feasible to apply semantic parsing on large corpora in the range of 100,000 to a few million articles.

Although, TextRunner and SRL-IE both perform n-ary extractions, neither allows searching of more than two arguments at the same time. The search of adjuncts such as locative, temporal, manner, cause, etc. is not possible either. This limits the usefulness of the systems, especially in areas where temporal or location- based search is of interest, such as in history or medicine. Additionally, TextRunner's extraction method relies on heuristics and a classifier that is language-specific, which makes multilingual information extraction more challenging.

We believe that by parallelizing a high-performance multilingual semantic parser (Björkelund et al., 2010), large multilingual proposition databases can be created. These would allow for deeper knowledge acquisition and analysis within the area of NLP.

Chapter 4

Applications

4.1 Information Extraction

Information extraction (IE) systems analyze text to identify entities and specific events. By filling in pre-defined templates with the extracted information, texts are transformed into tabulated data. The purpose of an IE system is to find specific and relevant information from a large body of text.

Previous IE systems, such as FASTUS (Appelt et al., 1993), have instead of performing a complete analysis on the entire text, relied on shallow methods using specific phrases and patterns to extract information. This approach has resulted in high performance systems with short processing time, with the drawback of being language and domain specific. With the advent of more accurate and efficient semantic parsers, traditional semantic analysis is becoming a viable approach for the task of IE.

IE within the biomedical field aims at extracting relations, such as interactions between proteins. Finding phrases that describe the location, manner, and time is of great importance in this field. For instance, the sentence

> *The interaction between the protein and the regulatory gene in the cell may influence the expression level.*

describes the location of the interaction – the cell. Tsai et al. (2007) investigate how a SRL system can be used to perform IE in the biomedical field. They constructed a biomedical SRL system, called BIOSMILE, and trained it on an annotated biomedical proposition bank. Going beyond simple entity-verb relation extraction, by using a SRL, they successfully extracted relations including argument modifiers with high precision and recall.

4.2 IBM – Watson

On February 16 2011, Watson, a question answering (QA) system developed by IBM won Jeopardy by defeating two former champions, Ken Jennings and Brad Rutter. To win Jeopardy, Watson had to answer questions that covered a broad domain in natural language.

Behind Watson is a research team of specialists in the area of natural language processing (NLP), question analysis, information management, speech recognition, linguistics, and many more. The utilization of NLP takes a prominent place in the architecture that powers Watson. NLP is used to analyze questions and extract relations and entities. Furthermore, NLP plays a large role in collecting and extracting propositions from a large amount and variety of texts, including encyclopedias, dictionaries, and news articles (Ferrucci et al., 2010). Figure 4.1 shows how Watson reads and extracts propositions from texts.

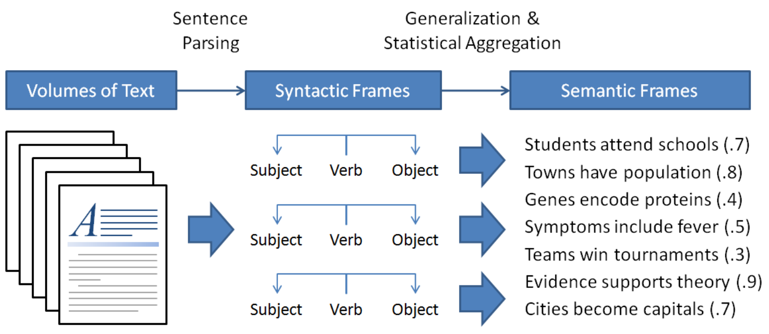

Sentence Parsing
Generalization & Statistical Aggregation
Volumes of Text
Syntactic Frames
Semantic Frames
Subject
Verb
Object
Subject
Verb
Object
Subject
Verb
Object
Students attend schools (.7)
Towns have population (.8)
Genes encode proteins (.4)
Symptoms include fever (.5)
Teams win tournaments (.3)
Evidence supports theory (.9)
Cities become capitals (.7)

Figure 4.1: Watson, automatic learning from reading.

The resulting proposition bank is used by Watson's many algorithms to discover and evaluate relations between questions and propositions. By using temporal and spatial inference, Watson can explore propositions and ultimately link them to a question. These operations are supported by a semantically annotated proposition bank. Figure 4.2 illustrates how Watson links questions to propositions.

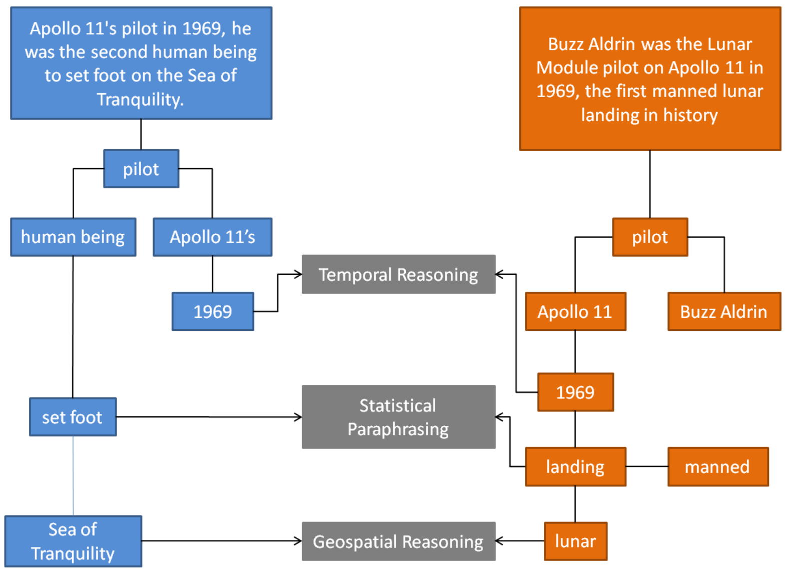

Apollo 11's pilot in 1969, he was the second human being to set foot on the Sea of Tranquility.
pilot
human being
Apollo 11's
1969
set foot
Sea of Tranquility
Temporal Reasoning
Statistical Paraphrasing
Geospatial Reasoning
Buzz Aldrin was the Lunar Module pilot on Apollo 11 in 1969, the first manned lunar landing in history
pilot
Apollo 11
Buzz Aldrin
1969
landing
manned
lunar

Figure 4.2: Watson, the linking of questions to propositions.

Chapter 5

Experimental Setup

5.1 Semantic Parser

Every year, the Conference on Computational Natural Language Learning (CoNLL) proposes a shared task with the aim of promoting NLP applications and evaluating them in a standardized environment. CoNLL provides the participants with corpora used for training, developing, and evaluating the applications.

For this project, we have chosen a high-performance multilingual semantic parser (Björkelund et al., 2010). Our choice is based on the fact that the parser reached high scores in the CoNLL 2009 (Hajič et al., 2009) shared task, has fast processing time, and the code is open source and freely available. The English data models used in our parser have been created from the corpus provided in the CoNLL 2008 (Surdeanu et al., 2008) shared task. The CoNLL 2008 corpus used for training is based on an annotated version of the Wall Street Journal, it is thus limited to a narrow domain. The Chinese data models have been created from a semantically annotated Chinese Treebank (Palmer and Xue, 2009).

5.2 Data Source

Wikipedia[1] is a popular source for data mining and is used by many NLP applications (Auer et al., 2007; Hoffart et al., 2010). Our choice of data source is motivated by the fact that Wikipedia is multilingual, has an appropriate size for this project, has articles with a diverse range of domains, and is available for free.

There are currently 279 editions of Wikipedia in different languages: at the time of writing, the English edition has 3,629,542 articles and is the largest edition, the Chinese edition has 354,166 articles. We believe that this is an appropriate size for this project and depending on available resources, we hope to parse a substantial part of Wikipedia.

As seen in Figure 5.1, articles in Wikipedia cover a wide range of subjects. This wide coverage will provide a challenge for our parser that has been trained on

[1]http://www.wikipedia.org/

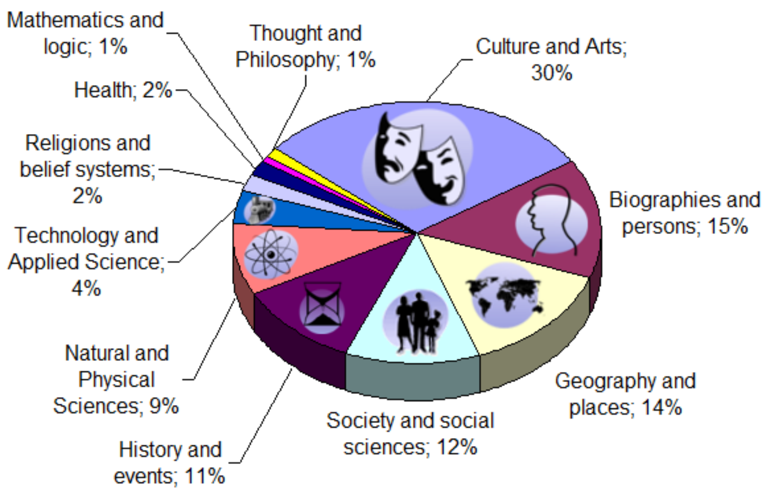


Figure 5.1: The various subjects in Wikipedia.

a single domain. The accuracy of the extractions will be evaluated based on the correctness of the extracted propositions. Since articles are selectable by subjects, it may be possible to determine both the in-domain and the out-domain accuracy of the extractions.

Many of the articles in Wikipedia are available in different language editions and connected to each other through links. This multilingual property provides easy access to data sources that can be used to build our multilingual proposition databases. We also believe that this multilingual property of Wikipedia articles can be used to explore syntactic and semantic links, which may advance the development of semantic parsers for new languages.

The Wikimedia Foundation[2] offers free copies of all the content in Wikipedia. The Wikipedia databases used in this project have been downloaded from `http://en.wikipedia.org/wiki/Wikipedia:Database_download`.

5.3 Cooperation with Lunarc

Parsing a large corpus requires appropriate resources in terms of computing power and infrastructure. Knowing that the parser has a processing time of 10 to 1000 milliseconds per sentence (Björkelund et al., 2010), we realized at the beginning of the project that parallelization of the parser was necessary.

Lunarc[3], a center for scientific and technical computing for research at Lund University, provides computational resources for academical use. The systems at Lunarc

[2]http://wikimediafoundation.org/
[3]http://www.lunarc.lu.se/

have a multiple node and multicore architecture with a high-throughput interconnection suitable for parallel processing. Lunarc generously provided the resources needed by this project.

5.4 Approach

Creating a complete IE system, capable of parsing at large scale, was at the beginning of the project a completely new area for us. Rather than investing time into designing for future unknown challenges, we decided to create fast prototypes that would handle the challenges as they were presented. Each prototype provided us with results and experience, which we used in the subsequent phase. Furthermore, with each prototype, we created new infrastructure to overcome obstacles, and increased the capacity of the system to parse and handle the data.

Central to this approach, was the use of continuous testing, which when applied on the same range of data, assured us that new infrastructure produced the same parsed data, and thus did not introduce any new errors.

Chapter 6

System

The complete system consists of several components that each fills a specific task in reading, extracting, transforming, and analyzing knowledge from a Wikipedia database.

The parsing framework reads articles from a Wikipedia database, filters, parses, and then stores the data in a semantically annotated structure. The task of parsing the entire database is parallelized by the use of scripts, which subdivide a range of articles and launch parsing jobs that work on smaller and more manageable ranges.

The proposition database is created by gathering the many small databases created during parsing and assembling them into one large database. With the use of a statistics module, the proposition database can be queried to provide statistics such as the number of and redundancy of propositions.

The ranking algorithm makes use of the statistics from the proposition database and assigns a higher ranking to propositions that are considered to have a higher probability of being correct extractions.

Finally, the frontend uses the index builder to collect the parsed data and create a flat document structure that is indexable using Lucene. The Lucene index is then queried through a web interface.

6.1 Parsing Framework

6.1.1 Parser

The parser can be viewed as a framework of components that together deliver a parsing system for a variety of data sources. It is responsible for doing all the processor intensive work by using a complete semantic parsing pipeline, as shown in Figure 2.1. Given a trained data model, the pipeline can be extended to parse a number of different languages including English and Chinese. Figure 6.1 shows an overview of the essential parts of the parsing framework.

The parsing pipeline is supplied with content from a content provider, which can easily be extended to various corpora. The Wikipedia content provider reads articles from a Wikipedia database and then uses a language specific filter to remove markups

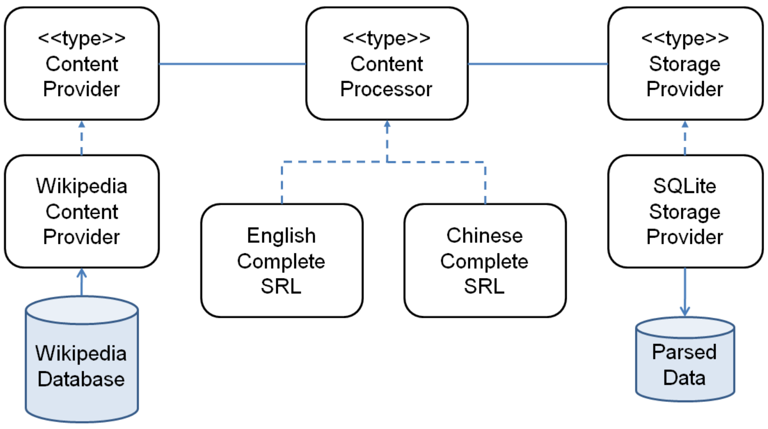

<<type>> Content Provider
<<type>> Content Processor
<<type>> Storage Provider
Wikipedia Content Provider
English Complete SRL
Chinese Complete SRL
SQLite Storage Provider
Wikipedia Database
Parsed Data

Figure 6.1: An overview of the parser

and other items that would otherwise impede the process of parsing. Wikipedia is available in the form of XML dump files[1] provided by the Wikimedia Foundation. Although XML is a suitable format for sharing data, it is less suitable for searching a certain element within the file. For this purpose, we developed a converter that takes a Wikimedia XML file and converts it to a SQLite[2] database. When converted to a SQLite database, it provides fast random access to any article, something that would otherwise not be possible using only the XML dump file. The database also allows for the storage and individual updating of the articles.

The parsed output from the pipeline is stored by a storage provider. This framework uses a SQLite database as end storage of the semantically parsed text. Extensions to other databases is possible by extending the storage provider interface.

6.1.2 HPC script

The high-performance computing (HPC) system at Lunarc[3] features a multinode and multiprocessor resource available through a time-sharing system. Using a submit script, it is possible to reserve a number of nodes for a limited time and to launch jobs on the given nodes. The systems allow jobs on different nodes to communicate through the use of the message passing interface (MPI) API. Figure 6.2 shows an overview of the HPC architecture.

To parallelize the work of parsing through Wikipedia, we developed a script for use on a HPC platform. A master job accepts a desired range of Wikipedia article identifiers. These are then subdivided by the master job into suitable subranges and distributed among the slave jobs that run the parsers on different nodes. Each node

[1]http://en.wikipedia.org/wiki/Wikipedia:Database_download
[2]http://www.sqlite.org/
[3]http://www.lunarc.lu.se/

can run up to three slave jobs. After completion, a new master job is launched by the previous master job using a new submit script and article range. This is repeated until all the desired documents have been parsed.

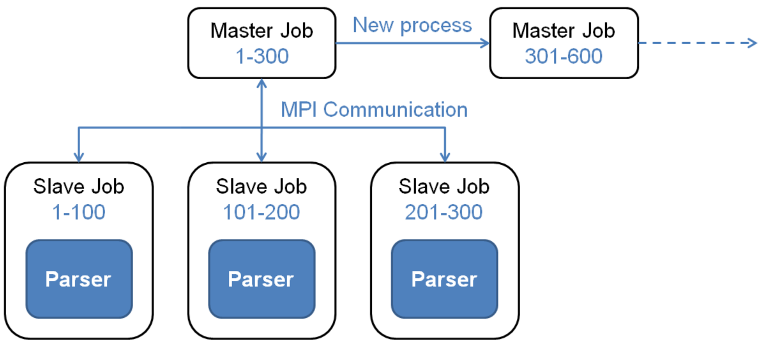


Figure 6.2: An overview of the HPC architecture

6.1.3 Results

Milestone 1

At the end of the first week, we had created a proof of concept system, with which we managed to parse over 1,000 articles from the English Wikipedia. It ran on a single computer using MySQL[4] for storing propositions and provided a simple web interface written in PHP[5]. The system allowed us to benchmark the parser and thus calculate preliminary resource requirements.

Milestone 2

By the end of week 6, still using only a single computer, we had parsed more than 10,000 English articles. We extended the system to support both English and Chinese parsing, and parsed a small amount of Chinese articles. By this time, searching through thousands of propositions had become too slow and response times could be measured in seconds. This prompted the replacement of MySQL and PHP in the frontend to a new architecture using Lucene and Apache Tomcat.

Milestone 3

We parallelized and adopted parsing for the HPC platform at Lunarc. Running on multiple nodes, we had parsed 100,000 English articles by the end of week 12. Parsing was now running in full production mode with a complete pipeline from the

[4]http://www.mysql.com/
[5]http://www.php.net/

Wikipedia articles to the parsed and indexed propositions. The daily parsing capacity was between 10,000 and 50,000 articles, depending on the available resources at Lunarc.

Milestone 4

By the end of week 13, we had parsed 378,453 English articles, 10% of the English edition of Wikipedia.

Open source framework

The parsing framework will be released as open source, allowing researchers to create large-scale multilingual proposition databases.

6.2 Proposition Database

The proposition database provides a unified schema for storing and retrieving propositions. It is used for storing parsed data by the parsing jobs, retrieving statistics, and building Lucene indexes. The schema is designed to handle semantically annotated sentences as defined by CoNLL 2009 (Hajič et al., 2009). It is in this sense a generic structure capable of handling parsed data from more than one parsing configuration. Figure 6.3 shows the data model.

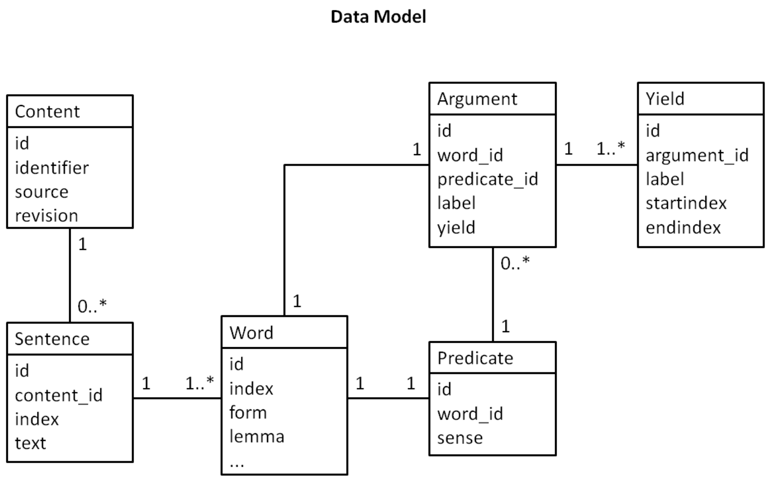


Figure 6.3: An overview of the data model

The proposition database also features a simple API, which allows the creation of databases, and storing and retrieving propositions. It uses SQLite, an ACID (Haerder and Reuter, 1983) compliant database engine, as the database backend for storing. The API makes good use of transactions, ensuring data integrity by making sure that parsed content is stored in its entirety. The database is thus protected against software crashes and power failures.

The database aggregator assembles the many smaller databases created by the HPC parsing jobs into one large database. The smaller databases are read one by one from a folder and added to the final database. This large database is more suitable for data processing tasks such as retrieving statistics and building the Lucene index.

6.3 Statistics Module

The statistics module uses the schema provided by the proposition bank to calculate statistics. The statistics are not only interesting as a measurement of progress, but are also essential in the process of designing the ranking algorithm.

English Wikipedia	
Articles	378,453
Sentences	13,428,114
Propositions	53,694,899

Table 6.1: An overview of parsing statistics at week 13.

In order to improve the speed of the calculations, the module makes a transformation of the structure of the proposition database. The hierarchical data model shown in Figure 6.3 is suitable for efficiently storing parsed data and also for providing simple statistics, such as the number of propositions in the database. However, if one wishes to look at propositions based on the properties of its arguments, the queries will quickly become complicated and inefficient. The problem stems from the fact that propositions have a variable number of arguments. To overcome this problem, the data model needs to be flattened, as shown in Figure 6.4. This flattened data model allows the efficient querying based on the number of and type of argument roles, and thereby also the grouping and counting of propositions.

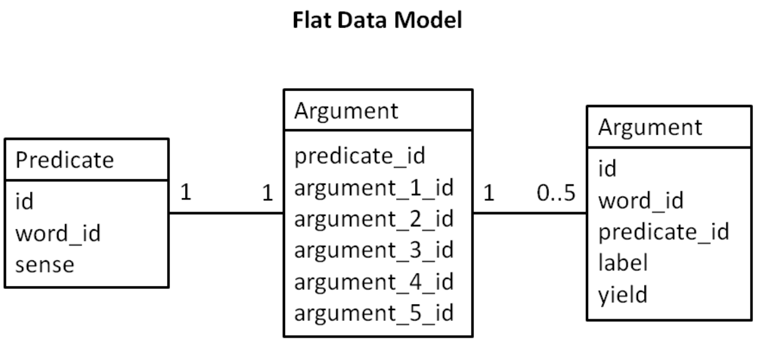


Figure 6.4: An overview of the flat data model, created by transforming part of the hierarchical data model in 6.3.

6.3.1 Results

At the time of writing, we have parsed more than 10% of the English Wikipedia. A few hundred articles from the Chinese edition of Wikipedia have also been parsed. The statistics generated from this data is vital in determining the focus for our efforts and also the approach for creating the ranking algorithm. An overview of the number parsed articles, sentences, and propositions is shown in Table 6.1.

6.4 Frontend

The hierarchically structured databases created during parsing are well suited for tasks that do not require low response time such as delivering statistics. To support end user search through millions of propositions, the database must be transformed into a format that allows rapid search and filtering. This task is handled by the Index Builder, which creates an inverted index using Lucene. This index can then be queried based on all the predicate-semantic roles that are discovered during parsing. Through a Java Servlet, a web interface is provided where it is possible to enter simple queries and view results. Figure 6.5 shows an overview of the frontend architecture, and Figure 6.6 shows the search page.

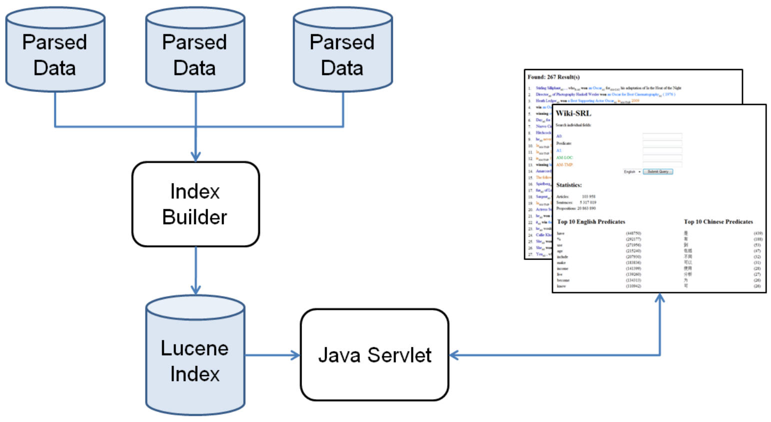


Figure 6.5: An overview of the frontend architecture

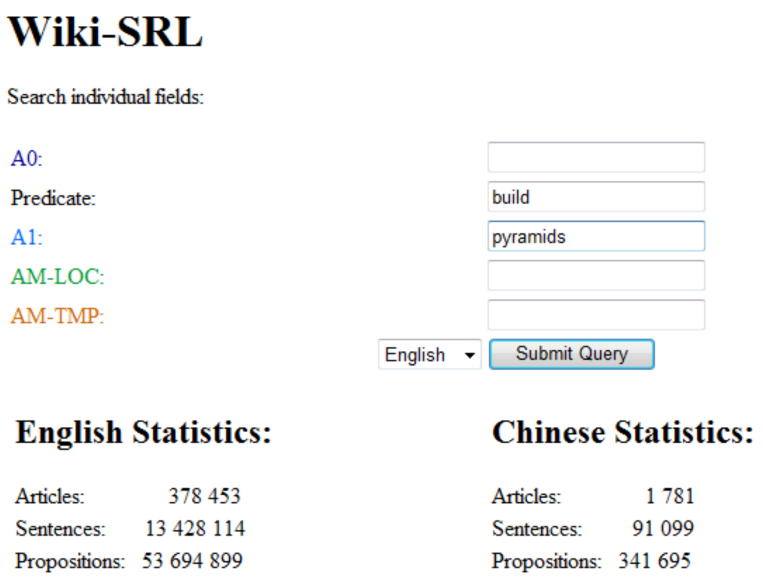
6.4.1 Index Builder

Storing the parsed data in a hierarchical database model is an efficient way of managing slow running tasks such as delivering statistics on word usage. However, a hierarchical model is less suitable for delivering results for complex queries that allow for a variable number of fields. Consider the data model shown in Figure 6.7. As previously discussed in Chapter 2, a proposition has a variable number of arguments with different roles depending on the predicate and its sense. For instance, the sentence

The orchestra played music.

has two arguments, labeled A0 and A1: $\{The\ orchestra_{A_0}\}$ *played* $\{music_{A_1}\}$.

Using a naive approach to search for the sentence, a SQL query could require the 'Argument' field 'label' to be 'A0' and 'A1' at the same time. This approach would of course return zero results. A more sophisticated approach could build tables



Wiki-SRL

Search individual fields:

A0:	
Predicate:	build
A1:	pyramids
AM-LOC:	
AM-TMP:	

English Statistics:

Articles:	378 453
Sentences:	13 428 114
Propositions:	53 694 899

Chinese Statistics:

Articles:	1 781
Sentences:	91 099
Propositions:	341 695

Figure 6.6: Searching the proposition database.

containing values from the 'predicate_id' field in the 'Argument', and join them to the 'Predicate' table. However, the complexity of that approach would grow with the complexity of the queries, resulting in slow and unresponsive search when applied on a database containing millions of propositions.

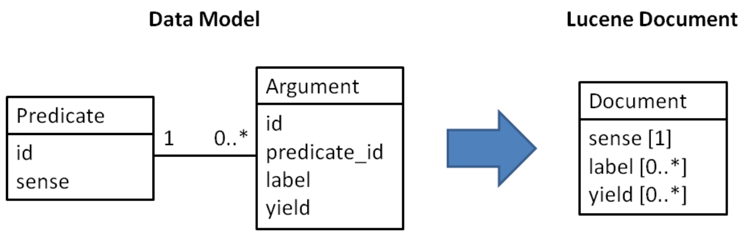


Figure 6.7: Flattening a hierarchical structure to a Lucene document

To make this query possible and more importantly to return results from a large set of propositions within a reasonable time, we used an inverted index. For this task we chose Lucene, a powerful search engine featuring high-performance indexing. Lucene enables a developer to perform powerful query searching including ranked and fielded searching, and sorting. It also has support for multiple languages.

We developed the index builder to bridge the gap between the hierarchical structure attained from parsing and the flat- and field-varying structure featured in Lucene. It collects the parsed data created during the parsing, creates a flat document structure as shown in Figure 6.7, and adds them to a Lucene index. The statistics module is used to calculate the ranking for each proposition added to the index.

6.4.2 Web Interface

We developed a web interface to query the millions of extracted propositions. Figure 6.6 shows an example of a search. It is possible to search for propositions from the English and Chinese Wikipedia. Searches are made by entering the predicate and arguments in lexical form. For instance, to search for who built the pyramids, one enters the lexical form of 'built', 'build', into the predicate field. Figure 6.8 shows the results of the query, the arguments in the sentences are coloured differently depending on their semantic roles.

A preliminary version of the web page is available from a server located at `http://exjobb22.ludat.lth.se`. Please note that this web page is only accessible from within the LTH network.

6.5 Ranking Algorithm

The first purpose of a ranking algorithm is to rank propositions that are more likely to be correct extractions. We implemented a ranking algorithm in the web interface when searching for propositions. All other parameters being equal, the highest ranking proposition is shown first.

A second and equally important use is to extract high-quality propositions. These will then be used to create a high-quality multilingual database of generic knowledge. The uses of such a database include being a source of knowledge for Q&A systems like the one demonstrated by IBM's Watson (Ferrucci et al., 2010), and also to increase the number of propositions in existing semantic knowledge databases such as YAGO2 (Hoffart et al., 2010).

6.5.1 Development

Creating a ranking algorithm for millions of propositions is nontrivial. The database and the queries used for generating the underlying statistics must be carefully designed to deliver results with accuracy and within reasonable time (i.e. hours, to fit within the scope of this thesis). There are many possible designs to consider for this task. Focusing on a certain group of propositions is one such approach.

As described in Section 6.3, by working with propositions that have only a fixed number of arguments, it is possible to create a flat table structure that has a fixed number of fields on which queries can be executed efficiently. The statistics in this section have all been calculated by using the statistics module.

In order to find the largest group of propositions, we first counted the number of propositions per number of arguments. We found that 71(!) was the maximum number of arguments. These propositions were however considered to be erroneous extractions. Table 6.2 shows the ten largest groups of propositions ranked by their number of arguments. Motivated by these numbers, as a first step, we focused on creating a ranking algorithm that works on propositions having only two arguments. Our aim was to extract meaningful generic knowledge in the form of predicate-argument tuples, such as *(universities, offer, degrees)* where *universities* and *degrees* are heads of the arguments, and *offer* is a predicate. These propositions would then be assigned a higher score reflecting that they have higher confidence value.

We believe that propositions that are repeated many times in a text have a higher probability of being correctly extracted. This is supported by findings in prior work (Downey et al., 2005). To determine if redundancy in propositions is a feasible approach, we first calculated the number of times a proposition had been found in a sentence. We then calculated the percentage of redundant propositions for a given predicate and sense. The redundancy percentage is calculated as the number of redundant propositions divided by the total number of propositions found. As an example, consider the data shown in Table 6.3. We have two tuples with redundancy, together they have (5+2) = 7 propositions. In all, there are (5+2+1) = 8 propositions for the predicate, *describe*. This gives a redundancy percentage of 7 / 8 = 87.5%.

Arguments	Propositions
0	1,489,769
1	17,115,058
2	19,187,436
3	10,646,902
4	4,009,519
5	1,016,695
6	189,464
7	29,466
8	5,030
9	1,526

Table 6.2: The number of propositions grouped by the number of arguments.

Argument 1	Predicate	Argument 2	Count
equations	describe.01	laws	5
methods	describe.01	approach	2
papers	describe.01	algorithm	1

Table 6.3: The number of propositions grouped by the predicate and its arguments.

We proceeded by creating a histogram showing the redundancy for all propositions. It is shown in Figure 6.9. As can be seen in the histogram, approximately 20,000 of the predicates[6] have no redundancy at all. However, a large mass of predicates show redundant propositions with a slight shift towards low redundancy as seen in the slope of the histogram body.

Looking at the propositions, we saw a significant amount of tuples with noun predicates, (their, disclosure, of), and tuples with pronoun arguments, (he, likes, it). Although the pronouns could be resolved with a coreference solver (Nugues, 2006), it would fall out of the scope for this project, and we therefore deemed both types of tuples as having low information content and discarded them. We focused on redundant propositions with noun arguments and verb predicates. This left us 599,447 redundant propositions representing 146,784 unique propositions which we believe are of high quality. A summary of the types and number of propositions can be seen in Table 6.4;

Our ranking algorithm assigns a confidence number to all propositions with noun arguments and verb predicates based on their redundancy. The confidence value is queried from the statistics module by other modules for the purpose of extraction or ranking, such as done by the Index Builder in Section 6.4.1.

[6]Here we make a difference between a predicate and an instance of a predicate. Predicate means a unique predicate.

Type	Count
All propositions with two argument	19,187,436
Propositions with redundancy	10,212,311
All (Noun, Verb, Noun) Propositions	2,712,134
(Noun, Verb, Noun) Propositions without redundancy	2,112,687
(Noun, Verb, Noun) Propositions with redundancy	599,447
(Noun, Verb, Noun) Unique propositions with redundancy	146,784

Table 6.4: The number of propositions grouped by proposition type.

6.5.2 Evaluation

Instead of evaluating all of the 2,712,134 (Noun, Verb, Noun) propositions, which when done manually would have taken years, we selected and analyzed a sample from both redundant and non-redundant propositions.

The non-redundant propositions were selected by randomly extracting 1,000 propositions from the set of 2,112,687 propositions without redundancy. We selected the redundant propositions by first taking the 500 most redundant propositions out of the 146,784 unique propositions, and then randomly choosing two propositions from each unique proposition. This was done in order to lower the risk that a single outlying proposition would incorrectly signal correctness and/or meaningfulness.

To evaluate the ranking algorithm, we assessed the correctness and meaningfulness of each proposition. By correctness, we mean if the proposition was correctly extracted from a sentence. This determination is objective and based on our linguistic knowledge. For instance, the sentence

The school has around 400 students.

is correctly extracted with the tuple: *(school*$_{A_0}$*, has, students*$_{A_1}$*)* , while the sentence

They are also good mountaineers and trekking guides.

is incorrectly extracted with the tuple: *(mountaineers*$_{A_0}$*, trekk, guides*$_{A_1}$*).* When we encountered an extraction error, we also marked the cause of error being either due to erroneous parsing or inadequate filtering.

By marking a proposition as meaningful, we mean that it is meaningful from a broad and general perspective. We do not require a proposition to be useful, which is something that would require a deeper analysis. The determination of meaningfulness is subjective; we do however use the same convention on all propositions. For instance, even though the sentence

Males constitute 51% of the population and females 49%.

Type	Correct	Meaningful	Parsing error	Filtering error
Redundant	88%	79.6%	5%	6.4%
Non-Redundant	70.8%	62.1%	25.4%	3.4%

Table 6.5: The number of propositions grouped by proposition type.

Redundant	Non-Redundant
Party won elections	Beaton replaced Hammond
Island has population	Boardwalk has attractions
City became capital	Halifax became settlement
School provides education	Faculty trains specialists
Students attend schools	Students follow regulations

Table 6.6: Examples of redundant and non-redundant propositions.

is correctly extracted with the tuple: *(Males$_{A_0}$, constitute, %$_{A_1}$)* . It is marked as not meaningful and it would require further analysis to discover that % refers to population. However, the sentence

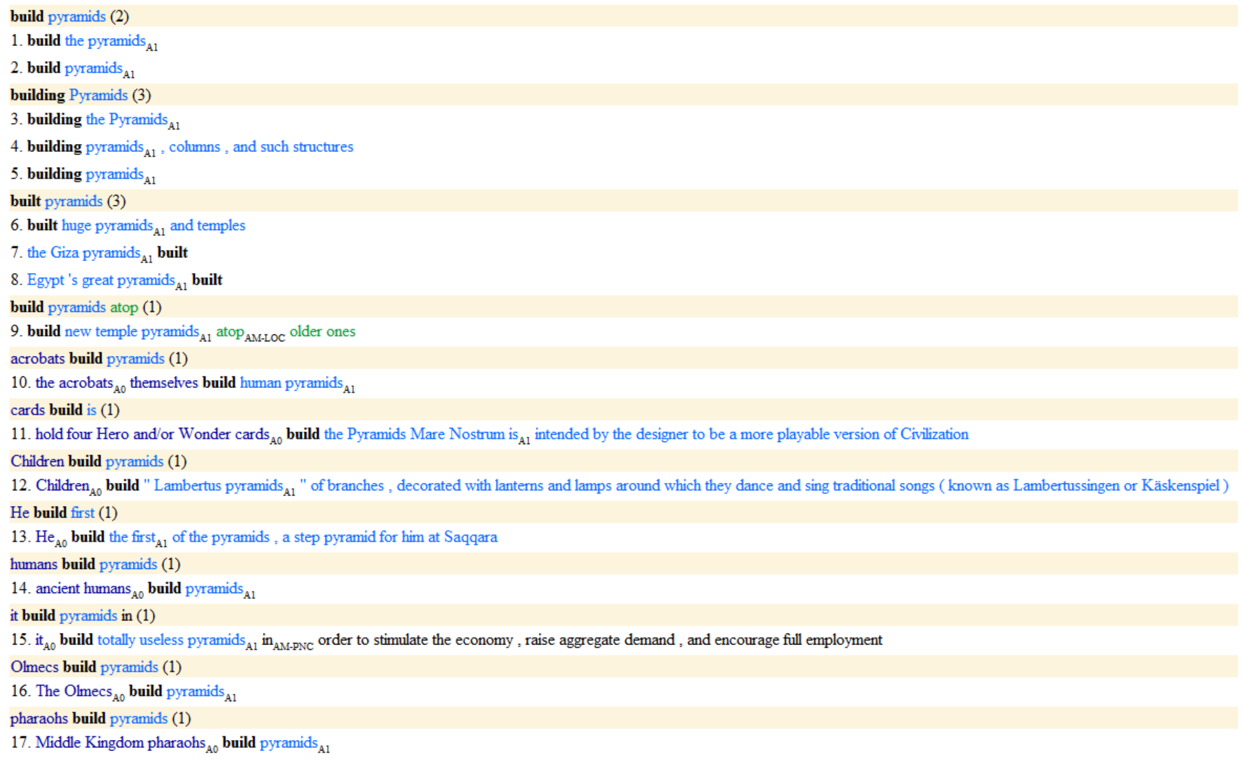
Males had a median income of $29,830 versus $22,553 for females.

correctly extracted with the tuple: *(Males$_{A_0}$, have, income$_{A_1}$)* , is marked as meaningful, even though further analysis is required to extract the amount of income and the context. The results of the analysis are presented in Table 6.5;

6.5.3 Discussion

The results clearly show that our ranking algorithm selects propositions that are more correctly extracted and are more meaningful than non-redundant propositions. Although the occurrence of filtering error slightly increased, we found that parsing errors dramatically decreased. In our opinion, this proves that propositions that are repeated many times in a text are more likely to be parsed correctly. This is also supported by the fact that the non-redundant propositions are extracted from sentences that use unusual and complicated expressions.

When comparing the redundant and non-redundant propositions, we found great differences in their character. Redundant propositions represent more generic facts such as: “Couples have children” and “Teams win games”, while non-redundant propositions are more specific: “Jamadagni has children” and “Tigers won final”. Table 6.6 shows examples of both redundant and non-redundant propositions that we extracted.



build pyramids (2)

1. **build** the pyramids$_{A1}$

2. **build** pyramids$_{A1}$

building Pyramids (3)

3. **building** the Pyramids$_{A1}$

4. **building** pyramids$_{A1}$, columns , and such structures

5. **building** pyramids$_{A1}$

built pyramids (3)

6. **built** huge pyramids$_{A1}$ and temples

7. the Giza pyramids$_{A1}$ **built**

8. Egypt 's great pyramids$_{A1}$ **built**

build pyramids atop (1)

9. **build** new temple pyramids$_{A1}$ atop$_{AM-LOC}$ older ones

acrobats **build** pyramids (1)

10. the acrobats$_{A0}$ themselves **build** human pyramids$_{A1}$

cards **build** is (1)

11. hold four Hero and/or Wonder cards$_{A0}$ **build** the Pyramids Mare Nostrum is$_{A1}$ intended by the designer to be a more playable version of Civilization

Children **build** pyramids (1)

12. Children$_{A0}$ **build** " Lambertus pyramids$_{A1}$ " of branches , decorated with lanterns and lamps around which they dance and sing traditional songs (known as Lambertussingen or Käskenspiel)

He **build** first (1)

13. He$_{A0}$ **build** the first$_{A1}$ of the pyramids , a step pyramid for him at Saqqara

humans **build** pyramids (1)

14. ancient humans$_{A0}$ **build** pyramids$_{A1}$

it **build** pyramids in (1)

15. it$_{A0}$ **build** totally useless pyramids$_{A1}$ in$_{AM-PNC}$ order to stimulate the economy , raise aggregate demand , and encourage full employment

Olmecs **build** pyramids (1)

16. The Olmecs$_{A0}$ **build** pyramids$_{A1}$

pharaohs **build** pyramids (1)

17. Middle Kingdom pharaohs$_{A0}$ **build** pyramids$_{A1}$

Figure 6.8: Results from searching the proposition database.

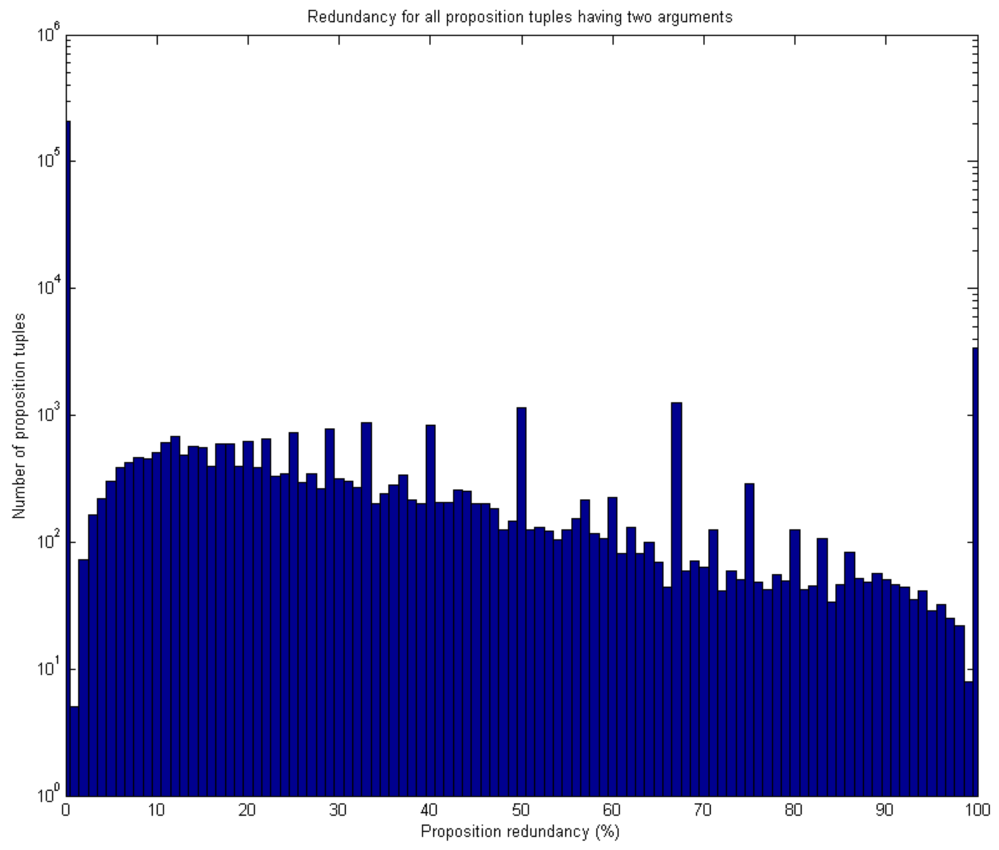


Figure 6.9: Redundancy for all predicates. The left part of the histogram corresponds to predicates that have unique propositions only. The right part corresponds to predicates that have redundant propositions only. *Describe.01* in our invented example would be on x=87.5%

Chapter 7

Scaling-up

Scaling up is much like exploring new land – you do not know what lies beyond the horizon. Will the memory and storage be enough or will some exceptional data cause an error, bringing the parsing to a halt? A lot of fine tuning of all the components is needed to get the best performance when taking on this big challenge. It becomes much of a game of being able to handle unforeseen errors and system requirements; a challenge that requires experience when designing the architecture and also the creation of supporting software.

7.1 Broad Domain Knowledge

To create a parsing framework capable of handling large amount of data, a broad knowledge is required in a diverse range of domains.

Databases are used for storing the articles and the extracted propositions. When parsing is done at a large scale, faults in the form of software errors and system halts can and will occur. It is therefore important that the databases used are capable of handling these faults in such a way that parsed data is left uncorrupted and in a consistent shape, such that parsing may be resumed. By choosing an ACID (Haerder and Reuter, 1983) compliant database and taking advantage of the transactional models that it provides, we are assured that parsed data is saved in its entirety or not at all, thus leaving the database in a consistent state.

The parsing jobs on the HPC communicate and coordinate their work by utilizing the message passing interface (MPI) API. A master job receives requests from parsing jobs and sends out parsing instructions. As in all concurrent programming, one must take careful steps in order to avoid undesirable situations where single jobs are left locked up indefinitely. For instance, should the master job for some reason fail to send parsing instructions to an awaiting parsing job, the parsing job will wait indefinitely. Since HPC resources are allocated for a certain time period and not released until all running jobs are finished, the resources will be wasted until the locked parsing job is terminated by force. Given the large scale of our undertaking, an accumulated amount of wasted resources may prove to be very costly.

As described in Chapter 2, Wikipedia articles include markup annotation. These must be filtered to obtain clean text that is easily parsed. This can be done by using regular expressions and other filtering techniques provided by the Java programming language.

To query propositions, we built a web interface which allows multilingual searching, grouping, and presenting of search results. The platform must be carefully chosen in order to accommodate the techniques and software libraries involved during querying. We based our first prototype on PHP, which does not provide an updated way of searching a Lucene index. We therefore created our final web interface using Java Servlets hosted on a Apache Tomcat server.

7.2 Design Challenges

During the development of the parsing framework, we were presented with a number of challenges that to a large degree stemmed from the large scale of the project. Some of these challenges were foreseen at the start of the project. Others showed up as the parsing grew in scale. The challenges are summed up in the following list:

- Growing resource needs,
- Handling errors,
- Deciding how much information to store, the type of information, and in which structure.

The growing amount of data naturally increases the requirements put on storage and processing. These can to a certain extent be handled by allocating more hardware resources. In other cases, a redesign of the algorithms and methods is required. What is perhaps unexpected is the growing amount of administrative work. This involves starting and monitoring parsing jobs, moving database files, and running all the modules required by the framework for generating statistics and search indexes. The manual labor can of course be automated by developing a supporting framework. However, one must not forget that new software introduces new error sources, which of course increase in amount along with the parsed data.

Sources for errors are numerous and increase with the scale of parsing. Sources include parsing errors, inadequate filtering, and memory leaks. Running hundreds of thousands of articles through a complete parsing pipeline, tests each part extensively and exotic and rare errors are bound to appear. While an inadequate filtering increases the parsing errors, strict filtering may filter out real data. Parsing and index building also requires heavy memory usage. It is therefore important to monitor this usage and ensure that unused blocks of memory are deallocated. In all cases, it is important that errors are handled and the time and place of the occurrence is recorded, such that parsing can be resumed.

During the design of databases and indexes, it is important to consider three aspects: the redundancy, the type, and the structure of the data.

Redundancy, in this case, deals with the question of how many times the same data is stored in different formats. This has negative consequences for storage usage and transmission times. However, it can greatly reduce the data processing time of modules and algorithms that make use of the data. As a simple example, consider the sentence

Snow leopards have eyes that are pale green or grey in colour.

Assuming projectivity, the yields for predicate have.03 are: *(Snow leopards*$_{A_0}$*, eyes*$_{A_1}$ *that are pale green or grey in colour)* . To reduce the storage requirements one may choose to store only the indexes pointing to the beginning and ending word of a yield, in this case indexes 4 and 12 for the second yield. This approach can have great effects when yields are large and the number of stored propositions is counted in millions. However, since the yield now must be rebuilt from the sentence using the indexes, this approach will also increase the processing time of any module that makes heavy use of the yield.

When building the search index, one must decide the type of data to include in the index. If a certain type of data, such as the complete article text, is not required during searching one may add only a reference to the data and perform a database lookup as required. Adding more data fields to an index creates longer searching time. However, performing a database lookup may also require extensive time. The challenge lies in balancing the amount of data to include in the index and looking up additional information in a database post-search. Trying different solutions is complicated by the fact that index building time may take up to one day when dealing with millions of propositions. Small scale testing may be misleading due to system factors, such as caching, when scaled up.

Finally, the structure of the data determines the ease and in some cases the feasibility of how modules may access and process data. In our case, the hierarchically structured data that follows a linguistic model, proved useful for general usage and for determining basic statistics. However, for other uses, such as determining the redundancy of propositions, the data structure required a reshaping in order for efficient calculation of the statistics.

Chapter 8

Discussion

The nature of this project made it many times overwhelming. Not only has it covered a wide range of disciplines, but it has also challenged us by its large scale. Although we feel that all of our goals have been reached, we also feel that some parts of the project might have deserved some more depth.

The choice of creating rapid prototypes versus doing a proper design for the parsing framework, was made to meet the fast approaching deadline set to give us time to parse a substantial part of Wikipedia. This approach also allowed us to handle the challenges that came with scaling up the parsing process, as they appeared. By not designing for future possible challenges and features, we saved the time needed for the actual parsing. This may have resulted in less elegant and less modular code, which is something we hope to improve in the future.

Our decision to use regular expressions to filter text resulted in a text that still has some HTML tags and wiki markup present. This affected the correctness of the extracted propositions, and to our surprise, it also showed an increase of parsing errors in the redundant propositions. Ideally, these annotations would all be removed and we would very much have liked to put more effort in doing so.

At the start of the project, we also hoped to parse the entire English Wikipedia. Besides bragging rights, this would have provided us with more data to evaluate our ranking algorithm. For similar reasons, we would have hoped to parse more than two languages. Provided with the proper resources, we hope to continue in our effort to completely parse Wikipedia.

Our ranking algorithm proved to increase the correctness of the obtained propositions. However, it also resulted in substantially fewer propositions. We had hoped to use a coreference solver and other lexical databases to cluster even more propositions. Unfortunately this did not fit within the time limits of this project. It would have also been interesting to explore how the ranking algorithm would have performed on a larger set of propositions.

The web interface, although fully functional, feels a bit Spartan and user-unfriendly. In addition, the results page might confuse users since the most redundant propositions from the search result are not always presented on the top of the page. This occurs when an argument keyword matches another part of the argument yield than the head. In those cases, the most redundant proposition overall is shown at the top. We hope to clarify this situation and also hope to develop the web interface further by allowing the entering of questions in natural language.

In the end, we hope our project will provide valuable material for further research.

Chapter 9

Conclusion

Prior work (Banko et al., 2007) on the extraction of propositions has focused on shallow syntactic processing that cannot consider the full meaning of sentences. Although there have been small-scale experiments done using a SRL (Christensen et al., 2010), it was unclear how they would perform on large-scale parsing. Furthermore, the full benefits of using a SRL system have not been explored. Additionally previous large scale IE systems have focused on extraction for only one language, and have not demonstrated how multilingual extraction would be performed.

We have created a framework for multilingual information extraction capable of large-scale multilingual parsing, including both English and Chinese corpora. By using trained language models, our framework demonstrates how semantic parsing can be adapted to new languages without the need of reworking extraction algorithms or patterns. Furthermore, we have shown how to add large-scale parsing capability to parsing by adapting a semantic parsing to a HPC platform.

The occurrence of erroneous extractions is a problem found in all extraction systems. In order to filter out less likely propositions, we have developed a ranking algorithm based on the redundant occurrences of propositions in text. This algorithm can be used for ranking semantic searches and also to create new corpora containing higher quality propositions. Although our algorithm retains only a small subset of propositions, we believe a higher yield can be achieved through the use of a coreference solver (Nugues, 2006) and other lexical databases.

To show the benefits of doing full semantic extraction, we developed a web interface capable of querying large multilingual proposition databases. The web interface allows the use of temporal and location based searches. This makes use of the semantic properties of the proposition database and creates new possibilities in semantic search.

Chapter 10

Future Work

Our framework demonstrates how multilingual parsing can be performed using a trained semantic parser. However, being a relatively new field, for many languages, including Swedish, there are no trained semantic parsing models. In the future, we hope to create Swedish semantic parsing capabilities by taking advantage of the multilingual property of Wikipedia by using an English and a Swedish edition of the same article. Since a semantic level builds upon feature sets from a syntactic level, we hypothesize that the semantic features for a Swedish semantic model can be gleaned from comparing the English-Swedish syntactic features and somehow translating the semantic properties. Figure 10.1 shows an overview of the idea.

We also hope to improve our ranking algorithm by using a coreference solver and possibly other lexical databases. This could resolve the pronouns found in a large number of propositions, link synonymous predicates, and yield even higher quality propositions. We also hope to export our proposition databases to other formats, such as RDF [1], and thereby contribute to existing semantic knowledge databases.

[1]http://www.w3.org/RDF/

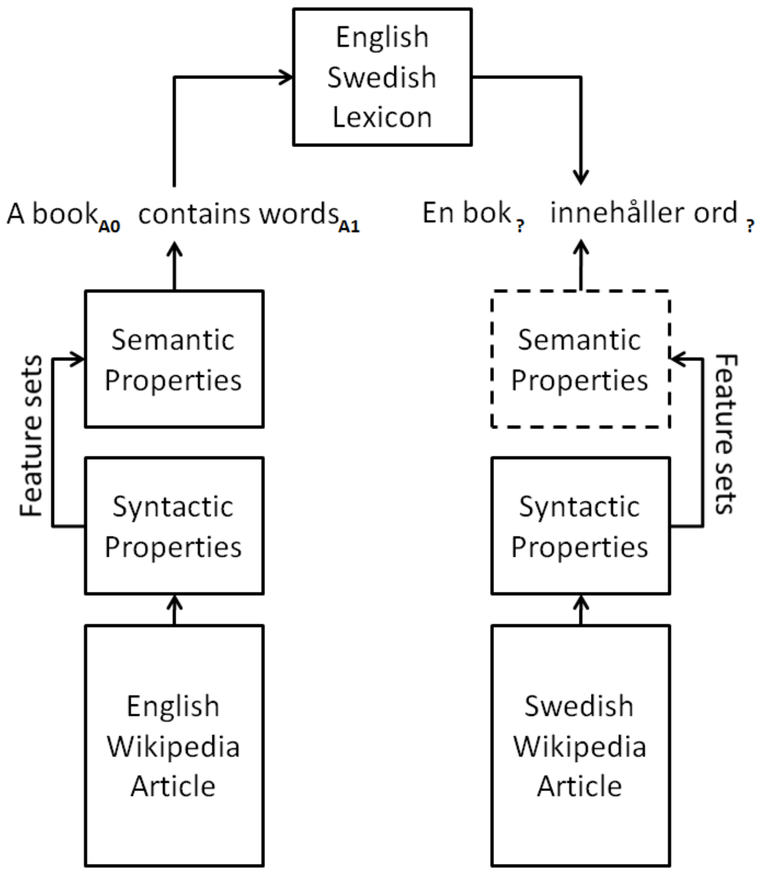


Figure 10.1: Developing semantic models for Swedish.

Bibliography

Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D. J., and Tyson, M. (1993). FASTUS: A finite-state processor for information extraction from real-world text. In *IJCAI*, pages 1172–1178.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). DBpedia: A nucleus for a web of open data. In Aberer, K., Choi, K.-S., Noy, N. F., Allemang, D., Lee, K.-I., Nixon, L. J. B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., and Cudré-Mauroux, P., editors, *6th International Semantic Web Conference (ISWC 2007)*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735, Busan, Korea.

Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open information extraction from the web. In Veloso, M. M., editor, *IJCAI*, pages 2670–2676.

Björkelund, A., Bohnet, B., Hafdell, L., and Nugues, P. (2010). A high-performance syntactic and semantic dependency parser. In *COLING (Demos)*, pages 33–36. Demonstrations Volume.

Björkelund, A., Hafdell, L., and Nugues, P. (2009). Multilingual semantic role labeling. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, CoNLL '09, pages 43–48, Stroudsburg, PA, USA. Association for Computational Linguistics.

Bohnet, B. (2010). Top accuracy and fast dependency parsing is not a contradiction. In Huang, C.-R. and Jurafsky, D., editors, *COLING*, pages 89–97. Tsinghua University Press.

Christensen, J., Mausam, Soderland, S., and Etzioni, O. (2010). Semantic role labeling for open information extraction. In *Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading*, FAM-LbR '10, pages 52–60, Stroudsburg, PA, USA. Association for Computational Linguistics.

Downey, D., Etzioni, O., and Soderland, S. (2005). A probabilistic model of redundancy in information extraction. In Kaelbling, L. P. and Saffiotti, A., editors, *IJCAI*, pages 1034–1041. Professional Book Center.

Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A.-M., Shaked, T., Weld, S. S. D. S., and Yates, A. (2004). Web-scale information extraction in knowitall. In *Proceedings of WWW-2004*.

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

Haerder, T. and Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., Straňák, P., Surdeanu, M., Xue, N., and Zhang, Y. (2009). The conll-2009 shared task: syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, CoNLL '09, pages 1–18, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hoffart, J., Suchanek, F. M., Berberich, K., and Weikum, G. (2010). YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. Research Report MPI-I-2010-5-007, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Nugues, P. (2006). *An Introduction to Language Processing with Perl and Prolog: An Outline of Theories, Implementation, and Application with Special Consideration of English, French, and German*. Cognitive Technologies. Springer.

Palmer, M., Kingsbury, P., and Gildea, D. (2005). The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Palmer, M. and Xue, N. (2009). Adding semantic roles to the Chinese Treebank. *Natural Language Engineering*, 15(1):143–172.

Punyakanok, V., Roth, D., and tau Yih, W. (2008). The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

Surdeanu, M., Johansson, R., Meyers, A., Màrquez, L., and Nivre, J. (2008). The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08, pages 159–177, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tsai, R., Chou, W.-C., Su, Y.-S., Lin, Y.-C., Sung, C.-L., Dai, H.-J., Yeh, I., Ku, W., Sung, T.-Y., and Hsu, W.-L. (2007). BIOSMILE: A semantic role labeling system for biomedical verbs using a maximum-entropy model with automatically generated template features.