Information Extraction and Data Acquisition for News Articles
Filtering and Recommendation

by

Ivo Lašek

A thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

PhD programme: Informatics

Prague, September 2014
Thesis Supervisor:
prof. RNDr. Peter Vojtáš, DrSc.
Department of Software Engineering
Faculty of Mathematics and Physics
Charles University
Malostranské nám. 25
118 00 Prague 1
Czech Republic

Copyright © 2014 Ivo Lašek
Abstract and contributions

Ever growing information overload on the web poses new challenges to information processing and filtering. In this doctoral thesis we implement and test our idea of a semantic information filtering systems based on co-occurrence analysis of named entities. We use information extraction techniques to link facts from unstructured documents (texts we want to filter) to Linked Data web resources (e.g. DBpedia or other Wikipedia based resources). The document enrichment with structured data enables new ways of documents filtering, searching and exploration.

The doctoral thesis addresses three topics: Information filtering in documents linked to a backing knowledge base, data acquisition from semi-structured web resources and the main focus is given to named entity recognition, disambiguation and linking in web documents. All three discussed topics support further evolution of intelligent information filtering systems.

In particular, the main contributions of the doctoral thesis are as follows:

1. We proposed, developed and tested a framework for on demand crawling of Linked Data.
2. We proposed and evaluated various context based approaches to named entity disambiguation.
3. We compared two state of the art frameworks for named entity recognition on non-English texts.
4. We constructed a comprehensive dataset based on all English Wikipedia articles for named entity disambiguation. The dataset reflects link structure and named entity co-occurrences in paragraphs from Wikipedia. These co-occurrences are then used for entity linking based on the context of an entity represented as the group of entities co-occurring in the same paragraph.
5. We designed a new approach to effectively deal with data obtained from Wikipedia. We work with about $10^8$ records reflecting entity occurrences in Wikipedia. Instead of dealing with up to $10^{16}$ combinations of records, we reduced the workload dramatically using sorted datasets and in-memory computation.
6. We provide deep insides in the structure of links between Wikipedia articles and evaluate link usage patterns.
7. Publicly available benchmarks testing named entity linking often contain records that do not need very sophisticated disambiguation approaches. Most frequent sense of an entity is often a correct guess. We compiled two new named entity linking benchmarks based on data extracted from Wikipedia that test the ability to disambiguate rather rare meanings of entities, where the real context has to be taken into account.
Keywords:
Named Entity Recognition and Linking, Linked Data, Semantic Web, Information Filtering
Acknowledgements

First of all, I would like to express my gratitude to my dissertation thesis supervisor, prof. RNDr. Peter Vojtáš, DrSc.. He has been a constant source of encouragement and insight during my research and helped me with numerous problems.

My research has also been partially supported by by the grant of The Czech Science Foundation (GAČR) P202/10/0761, by the grant of Czech Technical University in Prague registration number OHK3-002/11, by the grant of Czech Technical University in Prague (SGS12/093/0HK3/1T/18) and by the European Community’s Seventh Framework Programme (FP7-ICT) under grant agreement n° 287911 LinkedTV.
Dedication

To my family for their endless patience
## Contents

| List of Figures | xi |
| List of Tables  | xiv |
| Abbreviations   | xvii |

1 Introduction 1
  1.1 Motivation 1  
  1.2 Problem Statement 2  
  1.3 Objectives of the Thesis 2  
  1.4 Structure of the Thesis 3  

2 Use Cases and Early Attempts 5
  2.1 Semantic News Filtering Use Case 5  
    2.1.1 Semantic Information Filtering Workflow 6  
    2.1.2 User Profile 8  
    2.1.3 Storage 9  
    2.1.4 Evaluation 10  
      2.1.4.1 Used Data 10  
      2.1.4.2 The Course of Evaluation 11  
      2.1.4.3 Results – Prepared User Profile 11  
  2.2 Video Content Linked to the Web Use Case 13  
  2.3 Summary 14  

3 Background and State-of-the-Art 17
  3.1 Formal Models, Terminology and Metrics 17  
    3.1.1 Resource Description Framework 17  
    3.1.2 Linked Data 17  
    3.1.3 Parallel Processing with MapReduce 18  
    3.1.4 Information Extraction and Natural Language Processing 19  
      3.1.4.1 Text Processing 19  
      3.1.4.2 Vector Space Model and Bag of Words Representation 20  
      3.1.4.3 Weighting Terms in Vector Space Model 21  

5 Entity Linking with Wikipedia

5.1 Problem Description .................................................................................. 53
5.2 Data Structures Based on Wikipedia .......................................................... 54
5.3 Working Hypothesis .................................................................................... 56
5.4 Indexing Wikipedia for Disambiguation ...................................................... 56
  5.4.1 Basic Indexes .......................................................................................... 57
  5.4.2 In-memory Indexes ................................................................................. 57
  5.4.3 Indexing Pipeline .................................................................................... 58
  5.4.4 Observation ........................................................................................... 61
  5.4.5 Used Technologies ................................................................................. 62
  5.4.6 Performance Consideration .................................................................... 62
    5.4.6.1 Prerequisites ..................................................................................... 64
5.5 Recognition of Named Entities Surface Forms ............................................. 65
5.6 Various Context Representations for Disambiguation ................................ 66
  5.6.1 Bag of Words Context Representation – Algorithm 4 ......................... 66
  5.6.2 Sentence Structure Context Representation – Algorithm 5 ................. 67
    5.6.2.1 Subject Verb Object Extraction ....................................................... 68
  5.6.3 Structural Context Representation – Algorithm 6 ............................ 68
  5.6.4 Summary .............................................................................................. 69
5.7 Co-occurrence Based Algorithms ............................................................... 71
  5.7.1 Default and Normalized Most Popular Meaning Disambiguation – 71
    Algorithm 7 .................................................................................................. 71
  5.7.2 Sum Co-occurrence Disambiguation with Enhanced Co-occurrence Index – Algorithm 8 .......................................................... 72
  5.7.3 Maximum Co-occurrences Disambiguation – Algorithm 9 ............... 74
  5.7.4 Summary .............................................................................................. 74
5.8 SemiTags – Our Entity Linking Tool ............................................................ 74
  5.8.1 SemiTags architecture ......................................................................... 76

6 Evaluation ........................................................................................................ 79

6.1 Wikipedia Statistics .................................................................................... 79
6.2 Various Context Representations for Disambiguation ............................... 83
  6.2.1 Testing Dataset Description .................................................................. 83
  6.2.2 Comparison of Various Context Representations ............................... 84
6.3 Multilingual support ................................................................................... 85
  6.3.1 Named Entity Recognition in Non-English Texts .................................. 85
  6.3.2 German Disambiguation ...................................................................... 89
6.4 TAC Entity Linking Task Dataset ................................................................ 90
    6.4.0.1 Observation ..................................................................................... 90
6.5 Construction of Highly Ambiguous Benchmarks ........................................ 91
6.6 Co-Occurrence Based Algorithms Evaluation .......................................... 92
  6.6.1 TAC Evaluation – context aware approaches ...................................... 92
  6.6.2 Highly Ambiguous Benchmarks Evaluation ....................................... 94
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Schema of an information filtering system (see also Figure 7.1 in Chapter 7).</td>
</tr>
<tr>
<td>2.1</td>
<td>Semantic Information Filtering Workflow [A.10]. Different colors denote different parts of the system presented in Figure 1.1. Blue is the data enrichment component, red is the crawler, green is a general storage that stores apart from news articles also the knowledge base. The main focus is here given to the information filter (the orange part).</td>
</tr>
<tr>
<td>2.2</td>
<td>LinkedTV related content identification workflow [A.12]. The part that we contribute to is marked with the grey color.</td>
</tr>
<tr>
<td>3.1</td>
<td>MapReduce data processing diagram [90].</td>
</tr>
<tr>
<td>3.2</td>
<td>Calculation of precision and recall. The relevant part builds the golden standard and is usually marked by a human annotator or multiple annotators.</td>
</tr>
<tr>
<td>4.1</td>
<td>Infobox example taken from Czech Wikipedia. On the right side, there is a source code of this Wikipedia infobox written using Wiki markup.</td>
</tr>
<tr>
<td>4.2</td>
<td>Robots.txt example.</td>
</tr>
<tr>
<td>4.3</td>
<td>Getting host resource crawling delay for MapReduce nodes [90].</td>
</tr>
<tr>
<td>4.4</td>
<td>Sitemap.xml example with Sitemap Semantic extension data. Comments denote the data source locations processed by our crawler.</td>
</tr>
<tr>
<td>4.5</td>
<td>Pseudo-code Map() and Reduce() for crawling.</td>
</tr>
<tr>
<td>4.6</td>
<td>MapReduce Job Launching [90].</td>
</tr>
<tr>
<td>4.7</td>
<td>Hadoop Cluster [90].</td>
</tr>
<tr>
<td>4.8</td>
<td>Dependency of time to finish a crawling task on a number of cluster nodes. Time is shown in minutes [90].</td>
</tr>
<tr>
<td>4.9</td>
<td>Time consumption of URL processing tasks [90].</td>
</tr>
<tr>
<td>5.1</td>
<td>Named entities recognition and linking process. Surface forms $S_t$ denoting potential named entities are identified in a text $t$. Surface forms can be marked manually or using an algorithm such as Stanford Named Entity Recognizer. For all surface forms possible candidate meanings $C_t$ are generated (we describe the candidate generation process in Section 5.4.2) and from the candidates correct entities are finally selected $E_t$ (see Section 5.6 and Section 5.7).</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Wikilinks markup example. The top part shows a text on Wikipedia seen by ordinary users. The part below shows an example of the Wiki markup source code of the text above.</td>
</tr>
<tr>
<td>5.3</td>
<td>Structure of the candidate index for ( s_1, s_2 \in S^W ). Each record contains a sorted set of candidates (see Equation 5.6) together with counts of their occurrences under a given surface form (see Equation 5.5).</td>
</tr>
<tr>
<td>5.4</td>
<td>Structure of the co-occurrence index.</td>
</tr>
<tr>
<td>5.5</td>
<td>Wikipedia indexing pipeline.</td>
</tr>
<tr>
<td>5.6</td>
<td>Count of records in the page table compared with the count of distinct titles.</td>
</tr>
<tr>
<td>5.7</td>
<td>Number of redirects in Wikipedia.</td>
</tr>
<tr>
<td>5.8</td>
<td>Disambiguation and indexing architecture – used technologies. Third party Stanford Named Entity Recognizer is marked with grey color. It is wrapped to our own web service.</td>
</tr>
<tr>
<td>5.9</td>
<td>SQL query to retrieve counts of entity co-occurrences.</td>
</tr>
<tr>
<td>5.10</td>
<td>Structure of the paragraph index.</td>
</tr>
<tr>
<td>5.11</td>
<td>Structure of the enhanced co-occurrence index.</td>
</tr>
<tr>
<td>5.12</td>
<td>SemiTags – Graphical user interface.</td>
</tr>
<tr>
<td>5.13</td>
<td>Example of SemiTags web service JSON response.</td>
</tr>
<tr>
<td>5.14</td>
<td>SemiTags architecture.</td>
</tr>
<tr>
<td>6.1</td>
<td>Count of entities per paragraph. Graph is constructed according to Equation 6.2.</td>
</tr>
<tr>
<td>6.2</td>
<td>Count of entities per surface form (Equation 6.3) – default exact match of surface forms. Graph is constructed according to Equation 6.3.</td>
</tr>
<tr>
<td>6.3</td>
<td>Count of entities per surface form (Equation 6.3) – normalized surface forms. Graph is constructed according to Equation 6.3.</td>
</tr>
<tr>
<td>6.4</td>
<td>Count of entities per surface form – normalized surface forms compared to the exact match.</td>
</tr>
<tr>
<td>6.5</td>
<td>Illustration of inadequate usage of links in Wikipedia.</td>
</tr>
<tr>
<td>6.6</td>
<td>Surface forms per entity. Graph is constructed according to Equation 6.5.</td>
</tr>
<tr>
<td>6.7</td>
<td>Candidates percentage histogram counted according to Equation 6.6.</td>
</tr>
<tr>
<td>6.8</td>
<td>Comparison of precision of all three described methods on manually assembled dataset based on news articles from New York Times and Wall Street Journal. Used methods: Bag of Words (Section 5.6.1), Structural Measure (Section 5.6.3), Sentence Structure (Section 5.6.2).</td>
</tr>
<tr>
<td>6.9</td>
<td>Comparison of recall of all three described methods on manually assembled dataset based on news articles from New York Times and Wall Street Journal. Used methods: Bag of Words (Section 5.6.1), Structural Measure (Section 5.6.3), Sentence Structure (Section 5.6.2).</td>
</tr>
<tr>
<td>6.10</td>
<td>Precision and recall of entity identification in Dutch Texts using Stanford Named Entity Recognizer (Stanford) and OpenNLP library (ONLP).</td>
</tr>
<tr>
<td>6.11</td>
<td>Precision and recall of type determination using Stanford Named Entity Recognizer (Stanford) and OpenNLP library (ONLP) on Dutch texts.</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>6.12</td>
<td>Precision and recall of entity identification – Named entity recognition for Dutch and German.</td>
</tr>
<tr>
<td>6.13</td>
<td>Precision and recall of type determination – Named entity recognition for Dutch and German.</td>
</tr>
<tr>
<td>6.14</td>
<td>Precision and recall of German disambiguation.</td>
</tr>
<tr>
<td>6.15</td>
<td>Most frequent sense disambiguation results manually categorized on a sample of 100 queries.</td>
</tr>
<tr>
<td>7.1</td>
<td>Schema of an information filtering system – Conclusions.</td>
</tr>
<tr>
<td>7.2</td>
<td>Clever News main screen with news articles list</td>
</tr>
<tr>
<td>7.3</td>
<td>Clever News – detail of the relatedness classification of two articles</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of False Positives, True Positives, False Negatives, True Negatives, Precision and Recall by various approaches.</td>
<td>12</td>
</tr>
<tr>
<td>5.1</td>
<td>Results of cleaning of service pages – size of basic indexes.</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>Results of removing disambiguation pages – size of basic indexes.</td>
<td>60</td>
</tr>
<tr>
<td>5.3</td>
<td>Results of removing links to parts of articles – size of basic indexes.</td>
<td>60</td>
</tr>
<tr>
<td>5.4</td>
<td>Size of basic indexes after preserving only entities contained in the page table.</td>
<td>61</td>
</tr>
<tr>
<td>5.5</td>
<td>Entity mentions SQL table structure.</td>
<td>63</td>
</tr>
<tr>
<td>5.6</td>
<td>Processing times of co-occurrence computation.</td>
<td>64</td>
</tr>
<tr>
<td>5.7</td>
<td>In-memory co-occurrence storage.</td>
<td>64</td>
</tr>
<tr>
<td>6.1</td>
<td>Count of entities per paragraph – from one to ten entities per paragraph.</td>
<td>80</td>
</tr>
<tr>
<td>6.2</td>
<td>Count of entities per normalized surface form – from one to five entities per surface form.</td>
<td>82</td>
</tr>
<tr>
<td>6.3</td>
<td>Count of surface forms per entity – from one to ten surface forms per entity.</td>
<td>85</td>
</tr>
<tr>
<td>6.4</td>
<td>TAC dataset evaluation – comparison cleaned and non cleaned indexes:</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>structural, uncleaned (Section 6), default, cleaned and normalized cleaned.</td>
<td></td>
</tr>
<tr>
<td>6.5</td>
<td>TAC Dataset evaluation – default, normalized (Section 5.7.1, Algorithm 7)</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9) and Spotlight disambiguation (Section 3.4.2).</td>
<td></td>
</tr>
<tr>
<td>6.6</td>
<td>TAC Dataset evaluation – default, normalized (Section 5.7.1, Algorithm 7)</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9) and Spotlight disambiguation (Section 3.4.2). Selected queries recognized by Spotlight.</td>
<td></td>
</tr>
<tr>
<td>6.7</td>
<td>TAC Dataset evaluation – default, normalized (Section 5.7.1, Algorithm 7)</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9) and Spotlight disambiguation (Section 3.4.2). Selected queries linked by both Spotlight and normalized most frequent sense algorithm.</td>
<td></td>
</tr>
<tr>
<td>6.8</td>
<td>TAC Dataset evaluation – default, normalized (Section 5.7.1, Algorithm 7)</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9) and Spotlight disambiguation (Section 3.4.2). Only queries with links in the golden standard.</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.9</td>
<td>Soft Wikipedia benchmark evaluation – default, normalized (Section 5.7.1,</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Algorithm 7) co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.7.3, Algorithm 9) and Spotlight disambiguation (Section 3.4.2).</td>
<td></td>
</tr>
<tr>
<td>6.10</td>
<td>3d Candidate Wikipedia benchmark evaluation – co-occurrence sum (Section 5.7.2,</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Algorithm 8), max (Section 5.7.3, Algorithm 9) and Spotlight disambiguation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Section 3.4.2).</td>
<td></td>
</tr>
<tr>
<td>6.11</td>
<td>Comparison of B$^{3+}$ F1 measures with different TAC dataset subsets: TAC</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>(complete set of queries), TAC-SPOT (selected only queries where Spotlight</td>
<td></td>
</tr>
<tr>
<td></td>
<td>assigned a link), TAC-Spot+MFS (selected only queries where both Spotlight</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and our most frequent sense algorithm assigned a link), TAC-links (selected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>only queries with non-nil links in the golden standard).</td>
<td></td>
</tr>
<tr>
<td>6.12</td>
<td>Comparison of results on both highly ambiguous benchmarks. Used methods:</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>default, normalized (Section 5.7.1, Algorithm 7) co-occurrence sum</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9).</td>
<td></td>
</tr>
</tbody>
</table>
Abbreviations

Sets and signs

\( e \) A named entity
\( s \) A surface form
\( t \) A plain text where we search named entities and link them
\( m \) An entity mention in the knowledge base
\( q \) A query – an entity mention in the disambiguated text
\( \hat{e}_s^t \) A correct entity meaning \( e \) for surface form \( s \) in the context of text \( t \)
\( C_s \) Set of candidates for surface form \( s \)
\( C_t \) Set of candidates for all surface forms identified in a text \( t \)
\( S_a \) Set of ambiguous surface forms (defined in Equation 5.10)
\( N_e \) Set of mentions of a named entity \( e \) (defined in Equation 5.2)
\( W \) Wikipedia knowledge base
\( M^W \) Set of all entity mentions in Wikipedia

Number Notations

\( n_e \) Number of occurrences of an entity \( e \) (defined in Equation 5.3)
\( n^s_e \) Number of mentions of an entity \( e \) under a given surface form \( s \) (defined in Equation 5.5)
\( n_{e_i,e_j} \) Number of co-occurrences of entities \( e_i \) and \( e_j \) (defined in Equation 5.11)
\( n^s \) Total number of surface form \( s \) mentions under any meaning (defined in Equation 6.7)
\( n_s \) Number of candidates for surface form \( s \) (defined in Equation 5.7)

Miscellaneous Abbreviations

xvii
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma Separated Values</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>NAS</td>
<td>Network-Attached Storage</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-Access Memory</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFa</td>
<td>Resource Description Framework in Attributes</td>
</tr>
<tr>
<td>RSS</td>
<td>Really Simple Syndication</td>
</tr>
<tr>
<td>TAC</td>
<td>Text Analysis Conference</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

With the growth of the Internet, we are confronted with huge amounts of information every day. In our daily lives we process more data than ever before. The information come from everywhere. The main sources include news, blogs or social networks. We have still more powerful tools to increase our productivity. But the information overload, we are facing, poses a serious risk. It is crucial to develop new ways of information filtering which would effectively decrease the information overload.

Existing information filtering approaches use among others techniques of information retrieval. They treat texts as bags of words. These techniques work very well, when we are looking for similar articles in terms of similar texts or similar keywords. However, the lack of semantics makes it difficult to track a certain story line – not articles about exactly the same thing, but articles about same concepts.

In an average news article, a huge portion of information is not mentioned explicitly, but a reader knows what is going on thanks to her or his previous experience. For example in an article about Olympic Games in London, there does not have to be explicitly mentioned, that the Olympic Games took place in Great Britain, but most readers know that Olympic Games took place in London and it is a city in Great Britain.

So called named entities (names of locations, people, organizations etc.) are often bearers of meaning in news articles. Our hypothesis is that identification of these entities in texts and their connection to a backing knowledge base with additional machine readable data about these entities can improve results of information filtering. Our preliminary experiments have shown promising results (see Section 2.1).

In this doctoral thesis, we focus on enriching unstructured texts with semantic concepts. We examine the possibilities of linking these concepts with data obtained from freely available web information sources like Wikipedia, or structured Linked Data [19] resources like DBpedia [20] or Freebase [23].

In Chapter 2 we describe two basic use cases that show how semantic concepts in the form of named entities can be further used to improve information filtering and recommendation.
1.2 Problem Statement

We propose a system that enables information filtering for users. It uses data available on the web to decrease information overload. Nowadays, part of the web is available in the structured or semi-structured form (we elaborate this topic in more detail in Section 3.1.2). Our hypothesis is that this structured content can be used to improve information filtering.

![Figure 1.1: Schema of an information filtering system (see also Figure 7.1 in Chapter 7).](image)

In Figure 1.1 we show the schema of such information filtering system that crawls data from the Web and uses it to enrich unstructured textual or multimedia content (more on this topic in Chapter 2). The content enrichment is then used for improved information filtering not only based on textual information but also based on the semantic information in the form of data linked to texts.

We argue that named entities can be the clue connecting plain text to structured knowledge bases. An example of successful integration of structured and semi-structured content to information processing pipeline is IBM Watson [58]. It uses among all resources like DBpedia [20], Yago [152] or WordNet [110].

1.3 Objectives of the Thesis

The main objective of this doctoral thesis is to examine individual parts of the system proposed in Figure 1.1. We aim at evaluating their feasibility and providing prototypes validating proposed approach on public benchmarks.
We aim to propose a technical solution of the data enrichment process that facilitates structured-data based information filtering. Concretely we plan:

1. To examine possibilities of crawling data from the web and their storage.

2. To propose an approach to textual information enrichment based on statistics obtained from Wikipedia as one of the biggest semi-structured knowledge bases on the web.

3. To evaluate several methods of named entity recognition for textual information enrichment.

4. To examine technical feasibility of proposed approaches. We consider also performance challenges of the whole Wikipedia datasets summarization. We analyze the structure of $10^8$ occurrences of links in Wikipedia paragraphs that can build up to $10^{16}$ co-occurrences.

5. To evaluate named entities relations on a real world dataset.

6. To construct an alternative benchmark that tests among all the disambiguation of highly ambiguous and not very common entities. Real world datasets often contain many texts where the most frequent sense is simply the correct guess. Our benchmark focuses on the opposite. How well a disambiguation method works in difficult cases when the correct entity is a rare sense of its name and thus its context has to be used for a correct disambiguation.

### 1.4 Structure of the Thesis

The thesis is organized into 7 chapters as follows:

1. *Introduction*: Describes the motivation behind our efforts together with our goals. We describe the basic idea of an information filtering system.

2. *Use Cases and Early Attempts*: Shows two real world scenarios where the information filtering system is the essential component.

3. *Background and State-of-the-Art*: Introduces the reader to the necessary theoretical background and surveys the current state-of-the-art.

4. *Building the Knowledge Base*: Describes our approach to structured data crawling from the web and data storage. We summarize our lesson learned from dealing with more extensive web data resources.


7. *Conclusions*: Summarizes the results of our research, suggests possible topics for further research, and concludes the thesis.
Chapter 2

Use Cases and Early Attempts

This chapter describes two basic scenarios where we expect our information filtering system can be useful. We include here these examples to illustrate our direction in the system development.

In Section 2.1 we describe our early attempts to information filtering. We evaluate the impact of text enrichment with structured data on the overall user experience. Our approach to text enrichment and named entity linking is in detail described in Chapter 5.

Section 2.2 briefly describes information filtering in the context of much broader European project LinkedTV, where the author of this doctoral thesis is involved.

2.1 Semantic News Filtering Use Case

At the beginning we were examining possibilities, how to decrease every day human information overload.

One possible way is to exploit the advantages of semantic web technologies. We can try to identify real world entities in texts and maintain some background knowledge about their properties. With this semantic information about the data, we can not only effectively suggest interesting topics and filter out the unimportant ones. We can also give the user the explanation why we recommended the particular topic or the news article. This is usually not possible when using collaborative filtering or similar techniques, where the mechanism of recommendation remains hidden from users (because of its complexity).

Another problem by collaborative filtering is, that we need to keep track of many users whose interests overlap. From this point of view the content based recommender systems are more flexible, while incorporating the news feeds from social networks for instance. There are often only few users reading the same message. Actually sometimes you may be the only person who reads it.

But when we talk about the content based recommending by news articles, we do not have many attributes which would serve us to check against a particular user profile. We can compare the articles based on the text properties using information retrieval techniques. But knowing at least some semantic information about the articles would help.
We introduced a basic workflow to improve the results of named entity recognition with the help of a user feedback. Our workflow incorporated also social aspects of user motivation to annotate information. The annotations then serve as a base for further improvements of automated information extraction. We proposed the representation of a user profile for semantic information filtering.

A large-scale human interaction study on a personal content information retrieval system, carried out by Microsoft [48], demonstrated that: "The most common query types in the logs were people / places / things, computers / internet and health / science. In the people / places / things category, names were especially prevalent. Their importance is highlighted by the fact that 25% of the queries involved people’s names. In contrast, general informational queries are less prevalent.”

In our use case we focus on collecting data about people, organizations and places from news articles. We consider not only newspapers. Under news articles we understand also the posts of your favourite blogs, tweets of your friends or information from other social networks (e.g. news feeds on Facebook). In the articles, we then try to identify persons, organizations and places the particular article is writing about. We enrich the article with this extracted semantic information. Additionally, we register the time and the source of the information. We believe most information from news can be very well described precisely by the combination of the subject (a person or an organization), the time (when it happened) and the place (where it happened). If we have these information, we can effectively search and filter the news.

We proposed a system where users can set up their profile indicating what they are interested in – selecting persons and organizations they want to read about. They can further indicate their favourite information sources. Optionally they can define a time period, they are interested in. The system will then deliver the information matching their criteria. It can deliver information from similar data sources added by other users with similar profiles, too.

Finally users are provided with the possibility to rate, how well the offered information fits their needs. The rating is then used to enhance their profiles – to precise their preferences. Also it is possible to mark any particular information (a word, a name or a sentence) which is interesting for them and annotate this information. The annotations of the same article from other users can be displayed.

This is important to motivate people to annotate the articles. One gain is the refinement of the own user profile to get more accurate results. The other motivation is the social aspect of sharing the information among users of the systems. You can display what other users marked as interesting in the same article you are reading.

### 2.1.1 Semantic Information Filtering Workflow

Our approach to semantic information filtering was presented during the SemSearch Workshop by World Wide Web conference [A.10]. Figure 2.1 shows the general workflow of the whole system.

We consider three types of inputs.
2.1. SEMANTIC NEWS FILTERING USE CASE

Figure 2.1: Semantic Information Filtering Workflow. Different colors denote different parts of the system presented in Figure 1.1. Blue is the data enrichment component, red is the crawler, green is a general storage that stores apart from news articles also the knowledge base. The main focus is here given to the information filter (the orange part).

- **Unstructured data:** In our use case the unstructured data is represented by web pages with news articles.

- **Semantic web entities:** When users create their profiles, they need to indicate what they are interested in. They have to select some topics (among all companies or, persons or their properties in our case) – named entities. To construct the knowledge base, the structured data from semantic web data sources are crawled. There are many well managed data sources we may obtain the information from. The data sources relevant for our use case include among others DBpedia\(^1\) Freebase\(^2\) or New York Times news vocabulary\(^3\).

\(^1\)http://www.dbpedia.org
\(^2\)http://www.freebase.com
\(^3\)http://data.nytimes.com
• Annotations: Annotations result from the feedback of users of the proposed system. Any time the user sees something important or interesting in any article, she or he can mark the information and indicate what that information means.

2.1.2 User Profile

Users can select entities they are interested in. They can select entities directly or by defining a structured query which is then performed against the semantic repository and the resulting set of entities is added to the user profile. The system then selects only articles that fit such criteria – write about the selected entities.

In a similar way, users can determine the set of entities they are not interested in and form a blacklist. Articles containing these entities are filtered out.

The user profile (after its first initialization) is maintained automatically, using a user feedback.

For calculation of the ratings we use a weight similar to the tf-idf weight [139] used in information retrieval. Analogically we compute the ef-idef (entity frequency - inverse document entity frequency) as follows:

\[
e_{i,d_j} = \frac{\sum_k e_{k,d_j}}{\sum_k e_{k,d_j}}
\] (2.1)

where \(e_{i,d_j}\) is the number of occurrences of the considered entity \(e_i\) in a particular document \(d_j\) and the denominator is the sum of the number of occurrences of all entities identified in the document.

The importance of the entity idef \(i\) is then obtained by dividing the total number of documents by the number of documents containing the entity, and then taking the logarithm of that quotient.

\[
idef_i = \log\frac{|D|}{|\{d : e_i \in d\}|}
\] (2.2)

Algorithm 1 shows update of the user profile, after the user has rated one particular article \((d_j)\).

Any time an article \(d_j\) is rated\(^4\) by the user \(u\), the contained entities \(e_i\) are extracted and the user rating \(r_{i,j}^u\) for each entity is stored. Additionally, we store the information whether the user rated a particular entity or the whole article.

Each entity in the user profile is characterized by its significance for the user \(u\). Equation 2.3 shows the calculation of the significance \(s_{i}^{t,u}\).

\[
s_{i}^{t,u} = \sum_j (r_{i,j}^u \times e_{i,j}^t \times idef_i)
\] (2.3)

\(^4\)Here we assume explicit user rating. However in the future it could be evaluated also with implicit user interactions like clicking on the link of the article or reading an article for a period of time longer than a threshold.
The significance $s_{t,u}^i$ of individual entities contained in an article is always evaluated in the time $t$ of comparison of this article to the user profile. The ratings $r_{t,j}^{u,i}$ come from all the articles $d_j$ containing entity $e_i$ rated by the user $u$.

It is necessary to evaluate the significance dynamically in order to take into account the current state of the corpus of all articles. By $e_{t,j}^f$ and $idef_{t,i}$ we denote the values of these metrics in a particular point in time $t$ as both can change in time.

The importance of entities $idef_{t,i}$ may change, because the count of all articles $|D|$ grows in time. The proportion of occurrences of an entity in an article $e_{t,j}^f$ may also change. Also interests of the users may change over time, so it is desirable to link the ratings also to time. Then we can for example omit too old ratings.

In case that the user rates individual annotated entities and not the whole articles, same rules apply, but $e_{t,j}^f$ is always equal to 1.

In Chapter 5 we describe our approach to named entity recognition and disambiguation. We are also able to compute the confidence of the entity disambiguation (see Section 5.8). The significance defined in Equation 2.3 can be thus corrected by the confidence defined in Equation 5.26.

Algorithm 2 shows the filtering process. Every article is compared to the user profile based on the extracted entities.

The comparison of an article to the user profile is based on the computed significance $s_{t,u}^i$ of individual entities. The significance $s_{t,u}^i$ of extracted entities $e_i$ is summed up. If the article rating $ar_u$ is positive, the article is considered relevant for the user.

### 2.1.3 Storage

The storage contains collected entities in the semantic repository and all the crawled articles in a local cache.

The articles are stored in two copies. One is a snapshot of the original page. The other copy contains the snapshot with all annotations we were able to discover during the
entities identification process and all annotations coming from users. The preservation of annotations directly in a web page holds not only the information about the content of the annotation, but also the information about its location on the web page.

Additionally the annotated articles keep track of the time of its publication, time of its crawling and the source they were crawled from. Every annotated article contains a pointer to its original snapshot. Annotated articles are indexed by entities they contain.

Entities are indexed by the content of their textual attributes (e.g. names, labels, descriptions) to enable their fulltext search.

2.1.4 Evaluation

We performed several experiments to evaluate the impact of exploiting a semantic knowledge base in connection with information filtering.

In our use case scenario we assume a user (let’s call him Michael) is interested in newly emerging technology startup companies. Exactly, Michael wants the system to deliver the news about the companies set up in 2009 or later. Probably he just wants to collect an inspiration for his own projects. Michael expects the system to filter out all the general talks like ”How to start a company”, ”What should you do, to run a successful startup”. Actually this kind of articles rather bothers him.

On the other hand he expects the system to deliver the articles about new emerging companies, how quick is their growth, what is their business model and also the information about acquisitions of these companies. An example of the potentially interesting article can be ”ReadyForZero Launches Debt Management Platform To The Public”. ReadyForZero is the name of the company set up in 2010.

2.1.4.1 Used Data

Having this scenario in mind we collected 150 random articles about technology companies from TechCrunch\footnote{http://techcrunch.com}. These articles were carefully read by a human and 32 articles potentially interesting for Michael were selected. As the knowledge base for this case, we
used CrunchBase\footnote{\url{http://www.crunchbase.com}}. CrunchBase is the free database of technology companies, people, and investors that anyone can edit. CrunchBase is freely available also in a machine readable format for developers using its CrunchBase API.

2.1.4.2 The Course of Evaluation

Our aim is to compare traditional information retrieval techniques for information filtering with our method supported by a semantic knowledge base. We simulate the behaviour of the system right after the initialization of the user profile. By testing the information retrieval approach we suppose use of a system like Google Alerts\footnote{\url{http://www.google.com/alerts}}. Google Alerts enables you to define your own search query. Whenever some new document fitting the search criteria is discovered by Google, you get an alert by e-mail.

After the investigation of texts of 150 collected articles we identified the word ”startup” (and its other forms ”start-up”, ”startups” etc.) as the most significant keyword fitting Michael’s needs. We could also use for example dates (Michael is interested in companies set up in 2009, 2010 and 2011). But since the dates are very often mentioned in the articles and only in few cases there is a connection to the founding date of a company, the incorporation of dates gave us very poor results and resulted in too many wrong suggestions.

Further we had to define the important area of each web page which we dealt with. Since TechCrunch focuses mainly on startup companies, the word startup occurs on almost any web page. In fact from our data set counting 150 pages, 144 of them contained a word startup at least once. Startups were mentioned in advertisements, or links to other articles from the server. So we constrained the search criteria to include only two kinds of input. First only the body of the article text and in a second case the body together with the attached comments.

For the evaluation of the approach using the semantic knowledge base we set up the Michael’s profile to collect companies that have the property ”founded” (read from CrunchBase) set to the date after 1.1.2009. In our experiment we queried the CrunchBase to get all the companies set up after 1.1.2009. The resulting entities (companies) were used to form Michael’s user profile. During the evaluation we performed the named entity extraction with help of names of the companies obtained from CrunchBase. Basically CrunchBase served us as a gazetteer. Because most startups have rather uncommon names, this basic method works very well for this particular use case.

2.1.4.3 Results – Prepared User Profile

We evaluated three types of approaches:

- Semantic approach – using the semantic background knowledge (represented by CrunchBase knowledge base) in combination with the defined user profile.
• Traditional information retrieval approach applied to the article bodies - using the most significant keywords.

• Traditional information retrieval approach applied to article bodies and attached comments.

We summarize results of the evaluation in Table 2.1. The semantic approach outperformed the keyword based information retrieval approach mainly in those articles, where there was just a description of the company products or activities, but it was not explicitly mentioned in the text that it is a newly started company. However, interesting is that this fact was often mentioned by the users in the comments. So the searching also in the comments brought better results counting 23 correct suggestions.

Table 2.1: Comparison of False Positives, True Positives, False Negatives, True Negatives, Precision and Recall by various approaches.

<table>
<thead>
<tr>
<th>Method</th>
<th>FP</th>
<th>TP</th>
<th>FN</th>
<th>TN</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic approach</td>
<td>1</td>
<td>27</td>
<td>5</td>
<td>117</td>
<td>0.96</td>
<td>0.84</td>
</tr>
<tr>
<td>IR body</td>
<td>27</td>
<td>19</td>
<td>13</td>
<td>91</td>
<td>0.41</td>
<td>0.59</td>
</tr>
<tr>
<td>IR with comments</td>
<td>49</td>
<td>23</td>
<td>9</td>
<td>69</td>
<td>0.32</td>
<td>0.72</td>
</tr>
</tbody>
</table>

When we looked on missed articles (i.e. false negatives), we discovered that practically all the misses in the case of the semantic approach resulted from an incomplete knowledge base. There simply was no information about the founding date of the mentioned company. We found this by examining records of individual companies in CrunchBase. We did not find any case, where the article would describe a company but did not mention its name. Similarly the names of all the examined startups were quite distinctive and there was no problem with the disambiguation with the help of the knowledge base. In case of more general entities there would be certainly more errors resulting from wrong identification of entities.

Finally we examined the sum of wrong suggestions (i.e. false positives). That means those, which do not correspond to the articles selected manually. The semantic approach was wrong only in one case. It was in the article about a venture fund. There were enumerated companies financed by the fund. The companies were correctly identified as startups, but the article was not about them. On the other hand the keyword based information retrieval approach did not filter articles very well. It got confused by statements like: "The company had more influence over our day to day lives than any other startup this decade." Which describes the company set up in 2004. Despite many occurrences of the word startup in the text, the article does not describe the company, which is a startup as well. This was the source of most mistakes. The situation gets even worse, when we dive into comments from users. The comments are full of statements like: "Nice chance for my startup..." under the article which describes some type of funding.

From the results we can see that the semantic approach (with our ef-idef measure) delivers good results in terms of precision and recall. However, we have to state that
the user profile was quite simple and focused on one particular use case. In real world scenarios the user profiles would be much more complex. Also the used knowledge base is well managed and only rarely the needed information was missing. This does not have to be true when we consider using data obtained from semantic web generally.

This evaluation provides only preliminary results. It served us as a basic proof of concept before we go deeper in examination of individual components of the information filtering system. Even though we used a basic manual evaluation in this case results are promising and show that there is a huge potential in enriching plain texts with additional facts from backing knowledge bases. However, these systems are crucially dependent on the named entity recognition and disambiguation process and on the quality of the backing knowledge bases.

### 2.2 Video Content Linked to the Web Use Case

Named entity recognition is an important part of the LinkedTV European project [151]. The author of this doctoral thesis contributes to the second work package of this project, which focuses on video content enrichment and its linking to web resources [A.12, A.13]. Concretely, we focus on the named entity recognition and linking part of the whole video processing workflow. There are three scenarios, how enriching the videos can improve overall viewer experience while reducing the demand for massive editorial work. Mainly in the News Broadcast scenario named entities pose a very valuable part of information.

Figure 2.2 shows the basic workflow of identification of related content using named entity recognition. As a source of data on one side serves an existing database of videos. Videos are processed by automatic speech recognition engine. The resulting texts are combined with subtitles and additional meta data about the video (if available).

The LinkedTV scenario considers more video processing techniques, including visual concept detection, face detection etc. However, in this text, we focus on the textual data that are extracted from the sound track of the video or video meta data such as subtitles.

The second type of input is formed by web content. In the particular case of the News Broadcast scenario, German broadcaster RBB defined relevant sources of information like their homepage, blogs, Facebook pages, web pages of other departments of the company. These resources are regularly crawled together with textual information about multimedia content – pictures, videos and podcasts (if available on web pages).

Both inputs are preprocessed by keyword identification service and named entity recognition service. Annotated named entities are then enriched with information from backing knowledge bases. The relations between identified entities are used to identify web content which might be related to videos from the broadcaster’s database.

The whole entity extraction and linking workflow is automatic, but final check of the results by a human annotator is still required. Finally, results of the automatic annotation and related content detection are displayed in an annotation tool that is used by human

---

8 [http://www.rbb-online.de](http://www.rbb-online.de)
Figure 2.2: LinkedTV related content identification workflow [A.12]. The part that we contribute to is marked with the grey color.

editors for confirmation of the linking between videos and web content. Even when human assistance is still involved to ensure high quality of the content, huge amount of manual work is saved by automatic preprocessing of web content matching it with videos.

Apart from this information filtering use case, named entities and identified concepts are used also in other LinkedTV scenarios. They are used to improve automatic recommendation engines on the user side. But these additional scenarios are out of the scope of this doctoral thesis.

2.3 Summary

The idea from [A.10] is here used as a use case. We keep the user model defined in this scenario, but later we decided to use more general knowledge bases based on Wikipedia. CrunchBase is too specific for general news texts and messages that we focus on. Also it showed up that for proper content based recommendation the entities identification part is
2.3. SUMMARY

crucial. We need to have the process of information enrichment under control rather than using an existing tool as a black box. The confidence score of the named entity recognition and disambiguation can be incorporated in entity significance scoring in the future.

In certain cases it is desirable to take into account not only the best entity match but also the other matches (candidates). For example if we are interested only in some of their common property.

In LinkedTV scenario we showed another use case, where named entity recognition and linking to general knowledge bases is a crucial part of information filtering workflow. In this case named entities as concepts are used to link videos to relevant web content.
Chapter 3

Background and State-of-the-Art

This chapter is divided in four sections. In Section 3.1 we introduce the terminology and cover basic formal models and metrics used later in the text of this doctoral thesis. Then related work is listed. Our use case touches several topics, so the rest of the chapter is divided accordingly. First we list available options for data acquisition from the web in Section 3.2, then we move to information filtering and information extraction in Section 3.3, finally in Section 3.4 we describe related work in the area of named entity recognition, disambiguation and linking.

3.1 Formal Models, Terminology and Metrics

We focus on web data processing, so in this section we described basic semantic web principals and touch the topic of scalable data processing with map reduce. Next we turn more to information extraction and information retrieval principles relevant to this work.

3.1.1 Resource Description Framework

Resource Description Framework (RDF) is a metadata data model widely used on semantic web to represent information. The RDF data model represents data as statements about resources using a graph connecting resource nodes. An RDF statement has the form of a triple consisting of subject, predicate and object.

3.1.2 Linked Data

Long time has passed since Tim Berners-Lee published his vision of semantic web [16]. The most distinctive movement that has evolved since the time is Linked Data [75, 19]. It defines four basic principles of publishing raw data on the web:

1. Use URIs to denote things.
CHAPTER 3. BACKGROUND AND STATE-OF-THE-ART

2. Use HTTP URIs so that these things can be referred to and looked up ("dereferenced") by people and user agents.

3. Provide useful information about the thing when its URI is dereferenced, leveraging standards such as RDF, SPARQL.

4. Include links to other related things (using their URIs) when publishing data on the web.

These four principles enable universal crawler design for data acquisition on the web.

3.1.3 Parallel Processing with MapReduce

In Section 4.2, we discuss possibilities of data crawling from the web. The MapReduce paradigm is used. Here we provide a brief introduction in this type of parallel data processing.

Horizontal application scaling brings benefits in distributing CPU, memory or network load among many computer nodes. There are many frameworks and programming models providing architecture for parallel processing. For our crawler we have chosen a MapReduce programming model.

It is a proven concept for this type of activity used by Google [44], Apache Nutch [87] or other crawlers.

The main principle is very simple. An algorithm which performs required work is divided into \textit{map} and \textit{reduce} tasks. Both of them consume and produce key-value pairs, where key and value can be various data types.

The map task consumes an input of similar records\footnote{Similar in this context means, that every record has a similar information value, e.g. a set of web pages.} which are modified in requested way. For example a key can be a URL and value can be the content of the target document. The map task then emits a new key value pair. Such a pair is then forwarded to reducer. Before data are delivered, they are sorted according to their key value. This step is one of the most powerful features of MapReduce, because as keys are sorted, reducer receives list of all values which belong to the same key. Then it is up to the reducer to process merged values and produce final key-value output records. In the mentioned context, the mapper might emit some content extracted from the web page (e.g. all annotations) and the reducer then creates an inverted index of these annotations. Such results are then written to distributed file system (DFS).

Figure 3.1 shows how MapReduce works in general. Concrete implementations like Hadoop\footnote{Hadoop is the most popular MapReduce implementation.} can differ in some details. Most important are numbers of map() and reduce() nodes. The default model in Hadoop splits a dataset into chunks according to DFS block size, each map() receives one block of data, which is by default 64MB. Then there is one

\footnote{Probably the most successful open source web pages crawler available on \url{http://nutch.apache.org}.}
3.1. FORMAL MODELS, TERMINOLOGY AND METRICS

reducer which processes sorted map() outputs. The concrete setup can be customized in Hadoop’s configuration files.

The distributed file system is another very useful feature of MapReduce implementations. There is always one or more master nodes, which hold an index of data locality on the cluster. Each node in the cluster then has own local storage with blocks of data. If a file is larger then a DFS block, it is probable that some of the blocks are stored on different nodes. To make such a file system reliable, each stored block is replicated on few nodes. How many nodes hold same data, is determined by replication factor, which is usually an optional value. Data replication, then makes cluster more fail safe. So when a node breaks down, it’s data are stored on other disk and can be easily retrieved.

3.1.4 Information Extraction and Natural Language Processing

Further in the text we use some techniques and terms form the area of information extraction, natural language processing and information retrieval. Here we list basic terms and principals necessary to understand our approach.

3.1.4.1 Text Processing

Here we list basic text normalization steps that are common to many natural language processing tasks:

**Tokenization** – Tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. The tokenization in many languages (except for example Chinese or Japanese) may be relatively straightforward process of splitting words by spaces. However when special characters and shortcuts come into play the situation may be more challenging. Shortcuts require special handling. Usually these problems are solved using dictionaries containing lists of shortcuts or using regular expressions. Natural language processing frameworks like OpenNLP usually
implement this feature. The list of tokens becomes input for further processing or text representation (e.g. bag of words and vector model).

Normalization – Normalization is the process of transforming text into a single canonical form. Usually we omit characters that are not relevant for our concrete text processing tool. Normalization is also the way how to reduce dimensionality in vector space model text representation or increase recall by some applications. Typical text normalization tasks include:

- Diacritics removal.
- Removal of special characters.
- Text conversion to lowercase.
- Deletion of stop words.
- Misspell correction.
- Stemming [132] (the process for reducing inflected – or sometimes derived – words to their stem, base or root form).
- Lemmatization (the process of grouping together the different inflected forms of a word so they can be analysed as a single item).

Normalized tokens are often called terms and are used as identifiers for example in vector space model.

3.1.4.2 Vector Space Model and Bag of Words Representation

Vector space model [145] is a widely used algebraic model in information retrieval. Documents or generally texts are represented as vectors (Equation 3.1).

\[ d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j}) \] (3.1)

Each dimension corresponds to a separate term. If the term occurs in the document its value in the vector is non-zero weight of the term.

The similarity of two documents or a document and a query in information retrieval cosine similarity is often used because it is not biased to the length of documents. Cosine similarity compares the deviation of angles between documents. In practise the cosine of the angle is computed (Equation 3.2).

\[ \cos \theta = \frac{d_1 \cdot d_2}{\|d_1\| \|d_2\|} \] (3.2)

Where \( d_1 \cdot d_2 \) is the intersection (i.e. the dot product) of both document vectors, \( \|d_1\| \) is the norm of vector \( d_1 \), and \( \|d_2\| \) is the norm of vector \( d_2 \).

Because the order of words is disregarded this kind of model is often called also bag of words representation.
3.1.4.3 Weighting Terms in Vector Space Model

The most straightforward measure of term weight is the count of its occurrences in a document. The count is often normalized in order to prevent a bias towards longer documents and is often denoted as term frequency \( tf(t, d) \) – frequency of occurrences of a term \( t \) in a particular document \( d \).

However, the term frequency is usually high for words with little semantic meaning such as articles, prepositions or conjunctions (e.g. ”the”, ”a”, ”and”, ”at”). Therefore tf-idf \([139]\) (term-frequency times inverse document frequency) weight was introduced. The idf part is calculated according to Equation 3.3.

\[
idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{3.3}
\]

In Equation 3.3 \( D \) is the whole document corpus and \( |D| \) is the count of documents in the corpus.

The final tf-idf weight of a term is then calculated according to Equation 3.4.

\[
tf \ast idf(t, d, D) = tf(t, d) \times idf(t, D) \tag{3.4}
\]

Alternative weights for terms can be obtained by using BM25 \([138]\) measure.

3.1.4.4 Named Entity

In the following text under named entity we understand a phrase that clearly identifies one item from a set of other items that have similar attributes. Named Entity was defined by Grishman et al. as an information unit described by the name of a person or an organization, a location, a brand, a product, a numeric expression including time, date, money and percent found in a sentence \([65]\).

3.1.5 Evaluation Measures

In our experiments we often use classical information retrieval measures to compare results of various methods. Here we shortly recap, how these measures are counted.

One of the mostly used metrics in information retrieval are precision and recall. Figure 3.2(a) shows the sets that are compared when calculating precision and recall in normal information retrieval systems. Figure 3.2(b) shows the assumption for named entity recognition tasks that we use in our experiments.

The measures are then computed according to Equations 3.5 and 3.6.

\[
precision = \frac{|\{relevant\} \cap \{retrieved\}|}{|\{retrieved\}|} \tag{3.5}
\]

\[
recall = \frac{|\{relevant\} \cap \{retrieved\}|}{|\{relevant\}|} \tag{3.6}
\]
CHAPTER 3. BACKGROUND AND STATE-OF-THE-ART

(a) Precision and recall in information retrieval
(b) Precision and recall for named entity recognition and disambiguation

Figure 3.2: Calculation of precision and recall. The relevant part builds the golden standard and is usually marked by a human annotator or multiple annotators.

An alternative measure used to evaluate disambiguation is B-cubed scoring \( [8] \). B-cubed measure is used among all for evaluation of clustering tasks. In entity disambiguation or linking tasks given a set of queries \( Q \) (entities mentions) our task is to assign to each entity mention the correct identifier \( e_i \). The disambiguation might be seen as clustering of mentions \( q_i \) into equivalence classes given by the assigned identifiers \( e_i \). Each entity identifier \( e \) then builds an equivalence class. B-cubed precision is then counted for each query according to the Equation 3.7 and B-cubed recall according to the Equation 3.8.

\[
\text{b}_3^{\text{precision}}_{q_i} = \frac{\# \text{ correct links in the response equivalence class containing } q_i}{\# \text{ entities in the same response equivalence class containing } q_i} \quad (3.7)
\]

\[
\text{b}_3^{\text{recall}}_{q_i} = \frac{\# \text{ correct entities in the response equivalence class containing } q_i}{\# \text{ entities in the same golden standard equivalence class containing } q_i} \quad (3.8)
\]

Then the final B-cubed precision and recall is counted according to Equation 3.9 and Equation 3.10 respectively.

\[
\text{final } \text{b}_3^{\text{precision}} = \sum_{i=1}^{N} w_i \times \text{b}_3^{\text{precision}}_{q_i} \quad (3.9)
\]

\[
\text{final } \text{b}_3^{\text{recall}} = \sum_{i=1}^{N} w_i \times \text{b}_3^{\text{recall}}_{q_i} \quad (3.10)
\]

Here \( N \) is the number of queries (number of disambiguated items) and \( w_i \) is the weight assigned to the query. As advised by the authors of \( [8] \) for information extraction tasks equal weighting is used and thus \( w_i = 1/N \).

The stress should be given on the definition of a correct entity in Equation 3.7 and Equation 3.8. Consider the following example text:

“We will go to New York. New York is the city in United States.”

Assume we have a golden standard that contains two mentions of an entity with the identifier New_York and one mention of the entity United_States. Let’s take an example of
3.2 RELATED WORK IN WEB DATA ACQUISITION

A poor algorithm that recognizes both mentions of New York, but assigns a wrong identifier of London to them. United States are correctly linked to United States.

According to the original definition this result would still get a 100% B-cubed precision and recall. Because we correctly clustered both mentions of New York in the same equivalence class. The B-cubed measure disregards the identifier. It measures only the correctness of clustering.

Therefore an extended B-cubed+ measure was proposed for Text Analysis Conference Knowledge Population Task [156]. B-cubed+ takes into account also concrete identifiers of the disambiguated entities. The correct disambiguation must have also the correct identifier. So in our example this means that we would get 0.33 B-cubed+ precision and recall (0 for the two New York links to London entity and 1 for the United States linked to United States).

3.1.6 Summary

In this section we introduced basic terminology, principles of Linked Data and map reduce data processing that we use for crawling of semantic web resources described in Chapter 4. We described information retrieval and information extraction models that we build on in our named entity recognition and linking algorithms described in Chapter 5. Finally, evaluation measures used in our experiments (Chapter 6) were described.

3.2 Related Work in Web Data Acquisition

In this section we list available options and techniques that can be used to obtain structured information from the web. We focus more on the semantic web resources that provide data already in a structured form. However, overview of general approaches to web crawling is provided as well. We discuss known issues with semantic web data processing and data cleaning.

3.2.1 General Web Crawling

General purpose search engines use web crawlers to maintain their indices [5]. A general architecture of such a general purpose crawler is described in [26].

An example of a scalable distributed web crawler is UbiCrawler [22]. It uses consistent hashing to partition URLs according to their host component across crawling machines, leading to graceful performance degradation in the event of the failure of a crawling machine. UbiCrawler was able to download about 10 million pages per day using five crawling machines.

IRLbot [94] is a single process crawler able to scale to extremely large web collections without performance degradation. IRLbot was running over two months and downloaded about 6.4 billion web pages. In addition, the authors address the issue of crawler traps, and propose ways to ameliorate the impact of such sites on the crawling process.
Heritrix [112] is an open source crawler used by the Internet Archive. It is written in Java. Its design is similar to earlier crawler Mercator [78, 115]. Heritrix is multithreaded, but not distributed, and as such suitable for conducting moderately sized crawls. Another very popular crawler is the Nutch crawler [87] also written in Java. It supports distributed operation and has a highly modular architecture, allowing developers to create plug-ins for media-type parsing, querying and clustering. Nutch is a great source of inspiration for our own semantic web crawler described in Section 4.2. However, it is designed to crawl ordinary web pages and not data sources or even query SPARQL endpoints for data. Therefore we decided to implement our own solution [A.9].

3.2.2 Queuing for Crawling and Focused Crawling

There is a demand for universal crawlers to crawl as much data as possible and at the same time to keep locally stored versions of crawled information as fresh as possible. Of course, these are conflicting requirements. Therefore a trade off between freshness and completeness has to be found.

Crawlers maintain a frontier queue, containing links to web pages that need to be processed. In order to utilize resources more effectively a link ordering in a frontier queue was proposed in [36, 76]. Pages with higher rank are processed with higher priority. The ordering is performed either based on a PageRank [123] value or based on an indegree of a web page represented as a node in a link graph.

In [116] it was shown that a simple breadth-first crawling algorithm will tend to fetch the pages with the highest PageRank. In [77] authors validate that a random walk with acceptance probability proportional to the inverse of frequency that the crawler has encountered a link to a considered web page yields a sample of the graph that is statistically representative of the original.

Rather than crawling pages from the entire web, we may want to crawl only pages in certain categories. Chakrabarti et al. [32] proposed a focused crawler based on a classifier. The idea is to first build a text classifier using labeled example pages from a training set. Then the classifier would guide the crawler by preferentially selecting from the frontier those pages that appear most likely to belong to the categories of interest, according to the classifier’s prediction. Another examples of focused crawlers use naïve Bayesian classifiers [45] and support vector machines or neural networks [124].

Several variants of focused crawlers were implemented and evaluated in [14]. These include variants of classic, semantic and learning crawlers. Particular emphasis is given to learning crawlers based on the Hidden Markov Model (HMM) [100, 101].

Focused crawlers are important for our scenario where we are usually interested in a predefined set of resources. Often we need also extensive crawling of a selected resource in order to obtain the knowledge base as complete as possible. By semantic web resources it is possible to use additional structured data features such as semantic sitemaps (see Section 4.2).
3.2. RELATED WORK IN WEB DATA ACQUISITION

3.2.3 Semantic Web Crawling

In order to obtain data from semantic web resources, we need to explore available resources.

On the web, a crawler is an essential part of a search engine [26]. Similar situation is on the Semantic Web. Probably one of the first adopters of crawling technologies were authors of semantic web search engines. When Watson [142] was developed, one of the biggest crawling problems was, how to actually discover semantic data resources. The authors proposed several heuristics including exploring well-known ontology repositories and querying Google with special type of queries. The crawling of the discovered content relies on Heritrix crawler [4].

A similar issue was addressed by Swoogle [46], where three specific crawlers were developed: Google Crawler (for querying Google and crawling search results), Focused Crawler for crawling documents within a given website and Swoogle Crawler, which follows URIs of resources identified in discovered semantic web documents.

With emergence of Linked Data [15], still more and more resources are interlinked and the location of new data sources is not so difficult to find.

A pipelined crawling architecture was proposed for MultiCrawler [72] employed in SWSE semantic search engine [71, 70]. MultiCrawler deals also with performance scaling. It was achieved by distributing processed pages to individual computers based on a hashing function. However, the authors do not deal with a fail over scenario, where some computers in the cluster might break down.

A multithreaded crawler was used to obtain data for Falcons search engine [35]. A more sophisticated crawling infrastructure is employed in Sindice project [121]. It proposes a processing pipeline similar to SWSE, but uses parallel Hadoop architecture.

LDSpider [81] is a Java project that enables performing custom crawling tasks. The spider performs concurrent crawling by starting multiple threads.

3.2.4 Semantic Web Data Modelling

Semantic web search engines that aggregate data from multiple semantic web resources can be classified into two categories as stated in [71]:

- Systems that operate on a document abstraction (we call them document centric).
- Systems that operate on an object-oriented model (we call them entity centric).

The representatives of the first group are Swoogle [46] and Sindice [121]. While Swoogle does not seem to display continuous crawling capabilities, Sindice is actively maintained and serves as a service for public use [Sindice]. On top of Sindice runs an interesting extension called Sigma [158]. Sigma enables end users to search the semantic documents by keywords. The results matching the keyword are collected from multiple documents and user can further work with them (select trusted data sources, reorder or remove the returned statements).

---

[Sindice]: http://www.sindice.com
[Heritrix](http://crawler.archive.org)
These search results can be then embedded in another application (thanks to the output in RDF, JSON and several other formats) or as a widget in any web page (using JavaScript). User’s changes to the search results are persisted and a user can return to them any time using a unique permanent link. But the changes are still related to a particular search result and not to a practical entity. The aggregation of the user feedback is thus difficult and ambiguous. In case of keywords with more than one meaning, different users may have different requirements for search results.

On the other side of the barricade stand entity centric search engines. To the best of our knowledge, there are two running examples on the web: SWSE [70] and Falcons [34]. The idea of the entity centric approach is mentioned also in earlier works about SemSearch [96] and TAP [67]. The ultimate goal of this approach is to consolidate and clean the data from diverse data sources before they are presented to the end user or passed to other systems.

The problem by this approach is that matching same entities from different data sources is a difficult task. SWSE relies on the use of matching based on unique identifiers and inverse functional properties (that are already present in original data sources) in combination with reasoning. Falcons adds linguistic matching and structural matching techniques [83]. Still many entities are not recognized as same.

For the automatic processing, the ambiguity of search results poses a real problem. Therefore it seems to be more feasible to turn to individual, often manually organized semantic data sources that are part of the Linked Data cloud [19]. In these data sources, individual entities are presented together with their properties in machine readable form.

The central point and probably the mostly used data source in this Linked Data cloud is DBpedia. Another advantage of DBpedia is its tight connection to Wikipedia. Each contained entity corresponds to an article in Wikipedia, which makes DBpedia an ideal source of identifiers for named entity recognition and disambiguation [107].

The whole philosophy of DBpedia and technical details are provided in [20, 7]. Both papers provide a broad overview of the DBpedia background.

3.2.5 RDF Data Storage

Major RDF data storage solutions including Jena [164], Virtuoso [52], Sesame [27], Oracle [37] and 3store [69] use relational databases to store the data. The general idea is to create one giant triples table with three columns (corresponding to RDF triples: subject, predicate, object) containing one row for each RDF statement.

Instead of storing whole identifiers and literals directly in the triples table, Oracle, Virtuoso and Sesame replace strings with integer identifiers, so that the data is normalised in two tables. Whereas Jena takes the trade off – space for speed – and keeps strings directly in the triples table.

According to DBpedia benchmark [113] Virtuoso outperforms other triple stores in terms of performance of SPARQL querying.

[^http://www.wikipedia.org]
3.3. RELATED WORK IN INFORMATION FILTERING ON THE WEB

3.2.6 Summary

Four our use case it is necessary to crawl rather constrained predefined data sources then
crawling the entire web of data like in [70, 46, 121, 34]. We decided to implement our own
focused crawler but for semantic web resources [A.9]. We also perform data consolidation
and matching of same entities during the crawling process like [34, 70] but we do not
rely on error prone inverse functional properties. We take rather a light weight approach
exploiting owl:sameAs mapping in crawled data sources. We provide more details about
our concrete implementation in Section 4.2.

As RDF data storage in our experiments described in Chapter 4, we use Virtuoso based
on its good results in DBpedia benchmark [113].

3.3 Related Work in Information Filtering on the
Web

So far, we have introduced state of the art methods for data acquisition from the web. Now
we will briefly cover the information filtering part of the problem. In following sections
we focus more on induction of new knowledge based on the data acquired from the web.
There are two main approaches to filtering of content on the web: Collaborative Filtering
and Content Based Filtering.

Collaborative filtering is a general concept not necessarily specific for the news domain.
It draws on the idea, that people who are similar in their subjective evaluation of past
items are likely to agree again in the future.

Collaborative filtering has its advantages as a universal method – to a certain extent
domain independent. However, there are its drawbacks as well.

We need to keep track of many users with overlapping interests. If the system is not
used by enough users, or some types of the content are rare and interesting for only a
small group of people, the recommendation does not work very well (cold start problem).
If there is no one else, who could recommend reading an article, the article does not get
recommended. This is an issue by new articles, too. This problem is often referred to as
the first-rater problem [64].

Additionally, in the case of news, the data set changes very often. But for the collab-
orative filtering, it takes at least several hours to learn the recommendations from other
readers [102]. Thus the recommended news are often outdated. This is not the case by
other domains like retail products, where data sets are usually more stable.

Also by collaborative filtering the mechanism of recommendation is not explanatory
and remains hidden from users and a user can not do much to influence it directly.

Another practical problem of collaborative filtering in conjunction with news recommen-
dation was observed by Google News [102]. The collaborative filtering method sometimes
does not account for the individual variability between users. The authors observed that
entertainment news stories are constantly recommended to most of the users, even for those
users, who have never clicked on entertainment stories. They explain this observation with
The general popularity of entertainment news stories. Thus, there are always enough clicks on entertainment stories from user's neighbours to make the recommendation.

The other possibility of information filtering, generally, is a content based filtering. No user overlap is needed, as the system analyses item properties. Thus even a system with only one user would work.

While filtering products for example, this could be straightforward. Each product has some properties and we can try to figure out a user personal taste [51]. Given a user who likes black notebooks with large screen and enough disk space, we can recommend products with these properties. The main task is to model an actual user taste.

In news first of all, it is not clear, what are the item properties. We are provided with an unstructured text with only little explicit semantic information obtained for example from an RSS feed (e.g. a publication date, what is the title, who is the author and what is the main text).

But when we talk about the content based recommending by news articles, we do not have many attributes which would serve us to check against a particular user profile. We can compare articles based on the text properties using information retrieval techniques. Our challenge is to go beyond this approach and enable the use of semantic concepts in the form of named entities and their relations.

### 3.3.1 Content Based Recommendation Techniques

In news recommendation systems instead of talking about queries and documents an existing known article may be used as a query in order to find most similar (related) articles [21].

Decision tree learners such as ID3 [133] build a decision tree by recursively partitioning training data (text documents), into subgroups until those subgroups contain only instances of a single class. Unfortunately text classification tasks frequently involve a large number of relevant features. Therefore, a decision tree's tendency to base classifications on as few tests as possible can lead to poor performance on text classification. However, it was shown that if there is a small number of structured attributes, the decision trees can work well [88].

Nearest neighbour methods or clustering [73] can be applied on plain texts, too. In the case of texts, cosine similarity is usually used as a distance measure together with vector space model [166].

Another widely used machine learning approach for content based filtering of textual documents is Naïve Bayes. Naïve Bayes is well-performing text classification algorithm and is frequently used in text filtering tasks [18, 31].

Association rules mining can be used for text classification as was shown in [3]. Authors build on the idea that frequent itemsets associated with text categories represent the discriminate features among the documents in the collection.

---

[7] Really Simple Syndication - a family of web feed formats for publishing frequently changing works such as news or blogs.
3.4 Related Work in Named Entity Extraction

One of the main tasks of this doctoral thesis is to examine the possibilities of named entity recognition and linking for enhanced information filtering. Named entities are the missing piece connecting unstructured texts to structured knowledge bases. Recognized named entities linked to a backing knowledge base induce new knowledge about the texts and their semantic content. In this section we walk through various approaches to named entity recognition and disambiguation which opened the door for further research in more specific entity linking.

The named entity recognition and disambiguation process consists generally of the following steps:

- **Named Entity Recognition** – Identification of words or phrases in unstructured texts that represent a named entity. Later we call these phrases also surface forms.

- **Candidate Generation** – Finding possible word senses or identifiers of concrete candidate entities that can occur under the recognized surface form.

- **Disambiguation** – Selecting the most appropriate meaning (concrete category or identifier from a knowledge base) within a given context. When a concrete identifier from a knowledge base is selected, we talk rather about named entity linking.

Named entity recognition is one of the fundamental tasks in information extraction. Pioneering task of named entity recognition was introduced during the 6th Message Understanding Conference (MUC) [153]. The task was to develop a tool that would recognize named entities of following types.

**Entity names (ENAMEX):**

- Organisations (e.g. names of companies, political parties)
- Persons
- Locations (e.g. states, cities, lakes, rivers, mountains)

**Temporal expressions (TIMEX):**
CHAPTER 3. BACKGROUND AND STATE-OF-THE-ART

- Dates
- Times

Number expressions (NUMEX):

- Currency values
- Percentages

The task of named entity recognition was solved also in years 2002 and 2003 as part of the Conference on Computational Natural Language Learning. During this conference the task of language independent named entity recognition was defined [56, 157].

There are three more recent workshops focused on named entity recognition and named entity linking. By named entity linking the task is to find more specific identifier usually URI from some general knowledge base (e.g. Wikipedia or Freebase).

The most popular workshop is Knowledge Base Population (KBP) Workshop by Text Analysis Conference (TAC) [154]. The workshop organized by US National Institute of Standards and Technology (NIST) has multiple tracks, which include also the task of entity linking. The document collection comprises a combination of newswire articles and posts to blogs, newsgroups, and discussion fora. Given a query that consists of a document with a specified name mention of an entity, the task is to determine the correct node in the reference knowledge base (based on Wikipedia dump from 2008) for the entity, adding a new node for the entity if it is not already in the reference knowledge base. Entities can be of type person, organization, or geopolitical entity.

More recent dataset based on Freebase is used in Entity Recognition and Disambiguation Challenge (ERD) by Special Interest Group On Information Retrieval (SIGIR) Conference [30]. Here the task is similar to TAC-KBP entity linking track, but includes also short queries (typically queries submitted by search engine users). Also the evaluation queries do not include the entity names or their offsets in the text. Therefore evaluated systems can not exploit this information and have to correctly identify also the occurrence of the named entity. This distinction is important. In our experiments in Section 6.6 we use the TAC 2013 dataset, however we test both cases. Either we omit the information about entity names an their offsets, or we include them in the evaluation process.


3.4.1 Statistical Approaches Grounded in Computational Linguistics

Early studies were mostly based on hand crafted rules, but most recent ones use supervised machine learning as a way to automatically induce rule-based systems or sequence labeling
3.4. RELATED WORK IN NAMED ENTITY EXTRACTION

algorithms starting from a collection of training examples. However, when training examples are not available, even recent approaches stick with some kind of hand crafted rules often backed by a knowledge base [147]. Statistical approaches to named entity recognition can be divided into three groups: Supervised Learning Approaches, Semi-Supervised Learning Approaches and Unsupervised Learning Approaches.

3.4.1.1 Supervised Learning

The idea of supervised learning is to study the features of positive and negative examples of named entities over a large collection of annotated documents and design (learn) rules that capture instances of a given type. Supervised machine learning techniques include Conditional Random Fields [93, 104, 59], Hidden Markov Models [17], Support Vector Machines [6], Decision Trees [146] and Maximum Entropy Models [24].

Supervised learning techniques vary also in the range of features used to determine a concrete entity type. The LEXAS system [118] uses a wide range of features that can be used to train the disambiguation algorithm. These include Part of Speech (POS) tags of surrounding words, POS tag of the disambiguated word, surrounding words in their basic form, collocations (words or phrases often co-occurring with the given sense), verb-object syntactic relations. LEXAS determines the correct meaning of the word by looking for the nearest meaning in terms of the features. In [129], bigrams occurring nearby the disambiguated word are used as features. Weka [165] implementations of the C4.5 decision tree learner, the decision stump and the Naive Bayesian classifier are used.

3.4.1.2 Semi-Supervised Learning

As opposed to supervised learning methods, semi-supervised methods require only a limited set of examples or initial seeds in order to start the learning process. For example, the system may ask for a limited number of names of sought entities. They are then located in a text and the system tries to identify some contextual features characteristic for all the located entities. The results are then used to locate additional entities found in similar contexts. The learning process is then repeated.

In [114] a named entity extractor exploits the HTML markup of web pages in order to locate named entities. It is reported to outperform baseline supervised approaches but it is still not competitive with more complex supervised systems.

Semi-supervised learning is used in [25] to extract names of books and their authors. At the beginning example pairs of author name -- book name are given. They are used to learn patterns that model the context of these pairs. A limited class of regular expressions is used for the patterns. Such derived patterns are then used to extract new names.

Collins and Singer [38] use unlabeled data directly through co-training. They rely upon POS-tagging and parsing to identify training examples and patterns. Patterns are kept in pairs spelling, context where spelling refers to the proper name and context refers to the noun phrase in its neighbourhood. The training starts with a limited group of spelling rules. They are used to identify candidates in a text and classify them. The most frequent
CHAPTER 3. BACKGROUND AND STATE-OF-THE-ART

candidate contexts are used to derive contextual rules that can in turn be used to identify further spelling rules.

In [136], the algorithm starts with a set of seed entity examples of a given type. At the heart of the approach, there is a mutual bootstrapping technique that learns extraction patterns from the seed words and then exploits the learned extraction patterns to identify more words that belong to the semantic category. More fine-grained context representation is introduced in [39] where elementary syntactic relations [12] are used.

A web scale fact extraction is performed in [128]. The recall of fact extraction is increased by pattern generalization – words from the same class are replaced by the same placeholder. The authors report a precision of about 88% by 1 million extracted facts from 100 million web documents.

Ensembles are used in [130]. Combination of distributional [128] and pattern-based [125] algorithms is re-implemented. A gradient boosted decision tree is used to learn a regression function over the feature space for ranking the candidate entities. Another example of web scale named entity recognition is given in [163]. A wide variety of entity types is recognized. Training data is automatically generated from lists on web pages (tables and enumerations) and again by deriving patterns (templates). However, templates are used as a filter, rather than as an extraction mechanism.

In [63] a similar task of word sense disambiguation is supported by semantic resources obtained from large corpora where terms are mapped to domains. This domain model is constructed in the completely unsupervised way using clustering based on Latent Semantic Analysis. The authors report that such a domain model contributes to better results even with limited amount of training data that are often difficult to gather.

3.4.1.3 Unsupervised Learning

An example of unsupervised named entity recognition using WordNet is given in [2]. The aim is to assign a known concept from WordNet to an unknown concept in a text. It is achieved by analysing words that often co-occur with each known concept. Certain language patterns (e.g. such as, like, or other) are exploited in [54]. The Google search engine is used to locate additional hypernyms. The sets of hypernyms are then clustered in an attempt to find general types of named entities. An observation that a named entity is likely to appear synchronously in several news articles, whereas a common noun is less likely is exploited in [148]. Authors report they successfully obtained rare Named Entities with 90% accuracy just by comparing time series distributions of a word in two newspapers.

KnowItAll [53] uses the redundancy of the web to perform a bootstrapped information extraction process. As one of the features that serve as an input for Naïve Bayesian Classifier a pointwise mutual information (PMI) [159] is used. The PMI is counted between each extracted instance and multiple, automatically generated discriminator phrases associated with the class.
3.4. RELATED WORK IN NAMED ENTITY EXTRACTION

3.4.2 Knowledge Based Approaches

Apart from statistical approaches to named entity recognition, the recognition and disambiguation may be supported by a knowledge base. The knowledge base serves on one hand as the white list of names that are located in a text. On the other hand, many services supported by a knowledge base assign concrete identifiers to recognized entities and thus can be mapped to additional information describing the recognized entities. In this case we talk about named entity linking [134]. Many general purpose named entity recognition tools use DBpedia [20] as their knowledge base (e.g. DBpedia Spotlight [107], Wikify [109]) or map recognized named entities directly to Wikipedia articles [28].

Sometimes, limiting the recognition to only a constrained domain may improve the results for domain specific application. In [66], the authors deal with texts written in informal English by restricting the named entity recognition to the music domain. MusicBrainz [155] is used as a backing knowledge base. In [74], the authors use a specific ontology for person names disambiguation. They disambiguate names of researchers in posts from DBWorld [13], using DBLP [98] as a knowledge base. Person names disambiguation is examined also in [140]. Here, a social graph mined from social networks is used as a knowledge base. An example of named entity recognition in the geospatial domain is given in [162] that uses data from GNIS [61] and Geonet [62] combined with Wordnet [110] as a knowledge base.

One of the most popular knowledge bases remains Wikipedia. It was used also in [109, 107, 111]. A big advantage of Wikipedia is that links created in articles by Wikipedia contributors can be used as manual annotations. Each link to a Wikipedia article represents a mention of an entity identified by the target article. We elaborate this topic in more detail in Section 5.2.

One important feature of an entity is its commonness [106] (i.e. prior probability of a particular sense of a given surface form). In the case of Wikipedia, this is usually measured as the count of incoming links having a given anchor text (i.e. surface form) leading to a corresponding Wikipedia article. At least, when we do not have access to any context of the entity (e.g. when we just see USA), the most common meaning of that shortcut is probably the most meaningful match. In [150], the authors claim that disambiguation based purely on the commonness of meanings outperforms some of the state of the art methods dealing with the context of entities. We confirm these results with our observations (see Section 6.4) and elaborate in more detail the types of errors resulting from usage of the most common sense.

The most popular or most common meaning is not always the best match and the proper model of an entity context is very important. We can divide the approaches used for named entity disambiguation into two groups: either textual features of a context are compared in order to disambiguate a meaning, or structural relations between entities mentioned in a text are considered.
3.4.2.1 Textual Disambiguation

One of the earliest works focusing on entity linking to Wikipedia uses textual representation of the context. The context derived from Wikipedia articles is compared with the textual context of the entity in the disambiguated text. Context vectors are compared in order to determine the best candidate.

Textual representation of an entity context is used also in. Links in Wikipedia articles are used as annotations and their surroundings (words within a fixed size window around the annotation) are collected and indexed. They are then compared against the context of a disambiguated entity in new texts. When the context of an entity is not sufficiently big, the taxonomy of Wikipedia categories is taken into account for the disambiguation. For comparison of textual context vectors, the cosine similarity and TF-IDF weight are used.

Wikify and Spotlight use the textual representation of entities described in Wikipedia articles, too. Wikify attempts to identify the most likely meaning for a word in a given context based on a measure of contextual overlap between the dictionary definitions of the ambiguous word – here approximated with the corresponding Wikipedia pages, and the context where the ambiguous word occurs (the current paragraph is used as a representation of the context). The approach is inspired by.

Spotlight represents the context of an entity in a knowledge base by the set of its mentions in individual paragraphs in Wikipedia articles. DBpedia resource occurrences are modeled in a Vector Space Model where each DBpedia resource is a point in a multidimensional space of words. The representation of a DBpedia resource thus forms a meta document containing the aggregation of all paragraphs mentioning that concept in Wikipedia.

The meta document context representation of each candidate entity for an ambiguous surface form is compared to the target paragraph (containing disambiguated entity). The closest candidate in terms of cosine similarity in the vector space model is selected. For weighting individual terms, the TF-ICF weight is introduced. The TF-ICF measure is an adaptation of the TF-IDF measure. The only difference is that the IDF part is counted among concrete selected candidates and not over the entire knowledge base. Thus, the discriminator terms specific for the specific candidate selection are weighted higher.

In more recent work, a weakly semi-supervised hierarchical topic model is used for named entity disambiguation. It leverages Wikipedia annotations to appropriately bias the assignment of entity labels to annotated words (and un-annotated words co-occurring with them). In other words the frequency of occurrence of the specific form of the word in annotations of particular entities in Wikipedia is taken into account, when selecting the correct entity. The Wikipedia category hierarchy is leveraged to capture entity context and co-occurrence patterns in a single unified disambiguation framework. In our work we proposed and evaluated the approach that uses sentence structure among the whole Wikipedia corpus to represent an entity context (see Section 5.6.2).
3.4. RELATED WORK IN NAMED ENTITY EXTRACTION

3.4.2.2 Structural Disambiguation

In [111], the structure of links to Wikipedia articles corresponding to disambiguated entities is analysed. Each entity is represented by a Wikipedia article. The most similar entities to entities which are not ambiguous in the texts get higher score. The similarity [106] between two entities represented by Wikipedia articles depends on the number of Wikipedia articles that link to both of them. The score computed this way is then combined with an overall entity commonness for a particular surface form using a C4.5 classifier.

A very similar approach to word sense disambiguation was proposed in [117]. WordNet [110] is used as the knowledge base. The disambiguation starts with non-ambiguous words in the text and searches for senses that are connected to these non-ambiguous words. The grammar for this kind of disambiguation is proposed.

A more general approach to structural disambiguation of word senses is introduced in [108]. Distance between candidate labels or senses is counted and a graph is constructed consisting of labels as vertices and distances as weights of edges. The Random Walk adaptation in the form of PageRank algorithm is used to determine scores for individual labels. For each word, its label with the best score is selected. Various representation of distance measures are proposed. For the evaluation, the definition overlap of individual label definitions in a dictionary is used. This sense similarity measure is inspired by the definition of the Lesk algorithm [97]. Word senses and definitions are obtained from the WordNet sense inventory [110].

The work presented in [111] was further improved in [92]. An annotation is scored based on two types of features: one set is local to the occurrence of the surface form of mentioned entity and the other set of features is global to the text fragment. The annotation process is modelled as a search for the mapping that maximizes the sum of the local and global scores of the selected annotations. Experiments over a manually annotated dataset showed that the approach presented in [92] yields a precision comparable to [111] but outperforms it in terms of recall.

Systems introduced in [29] and [126] show a more general direction close to the semi-supervised learning approaches. They are able to learn automatically, how to identify even new entities (not present in a backing knowledge base yet) and extract relations between them.

3.4.3 Exploitation of Named Entities

Traditional information retrieval measures work pretty well when we aim at identification of documents with similar content or retrieving documents containing a predefined set of keywords. However, in [89] authors argue that named entities mentioned in the documents constitute important part of their semantics. In [119] named entities are used to detect the novelty of text documents.

With the emergence of huge freely available knowledge bases such as Wikipedia or DBpedia [20], or social network data, the effort of named entity recognition approaches focuses on linking terms in texts to concrete entities in these knowledge bases. We thus
have the possibility to gain additional information about an entity from these knowledge bases and represent the semantics hidden in named entities.

A practical example of exploitation of named entities in BBC is described in [91]. Named entity linking is used to enrich BBC content with structured information from Linked Data resources like DBpedia [20, 7] or MusicBrainz [155].

KIM platform [131] introduces rather a structured approach to information retrieval and discovery of related information. Named entities identified in texts are used to provide more precise search results. Thanks to the mapping of discovered entities to a backing ontology, the system is able to respond even to structured queries, not only keyword based queries.

The idea of exploiting semantic annotations for better retrieval was proposed already in earlier works [85, 127]. In [33] a probabilistic modelling framework is proposed that combines both human-defined concepts and data-driven topics and is used to model the content of documents. Wikipedia concept relatedness information is combined with a domain ontology to produce semantic content classifiers for content filtering in [103]. In [50, 49] automatic information extraction and fuzzy inductive logic programming is used for automatic classification of reports about accidents and news about acquisitions in Czech and English language.

The importance of named entities and their linking from texts shows also the Knowledge Graph effort and related work at Google [47, 68, 122].

In multimedia context, named entity recognition helps in retrieval of additional related content and locating related videos [13].

3.4.4 Public Named Entity Recognition APIs

There are several publicly available APIs performing named entity recognition and linking. The comparison of existing tools is provided in [137]. The most popular ones include:

- Alchemy API [8]
- DBpedia Spotlight [9]
- OpenCalais [10]
- Zemanta

According to the evaluation performed in [107] the best results in entity linking were achieved using DBpedia Spotlight. Very good results of DBpedia Spotlight by named entity linking are reported also in a more recent study [137]. That’s why we decided to use DBpedia Spotlight as a reference tool in our evaluation (Section 6.6).

---

8 http://www.alchemyapi.com
9 https://github.com/dbpedia-spotlight/dbpedia-spotlight/wiki
10 http://www.opencalais.com
3.4. RELATED WORK IN NAMED ENTITY EXTRACTION

3.4.5 Summary

We were considering to use one of the ready made tools as a black box for our information filtering system. But given the feedback to our vision paper [A.5] we decided to dig deeper in the problem of named entity linking and examine the problem on a lower level. Also many of the tools set quite restrictive limits on the count of queries one can post to them. Last but not least for further usage of the results it is critical to have a low level control over the entity linking process, configuration and confidence scoring. This turned out to be right decision also because some of the tools that we were considering discontinued meanwhile the maintenance of named entity recognition services.

We proposed a co-occurrence based approach to disambiguation [A.2] similar to the approach described in [111]. Contrary to this approach, we look for the most probable combination of all entities identified in an article (we don’t compare their individual similarity). Thus we do not need to start from non ambiguous entities. Even if all entities in an article are ambiguous, our method is able to disambiguate them. Additionally, we analyse co-occurrences of entities in same paragraphs in Wikipedia articles, which provides a more fine grained context representation than comparison of whole Wikipedia articles.

In [A.3] we were evaluating three different context representations:

- Structural context representation based on the approach described in [A.2].
- Sentence structure context representation.
- Bag of words context representation based on the approach described in [107].

Structural context representation turned out to be the most effective. In [A.1] we added also multilingual support. Our approach is summarized in Chapter 5 and evaluated in Chapter 6 where we extend the structural context representation and provide extensive statistics about the structure of links in Wikipedia.

In Section 5.7.1 we extend the commonness measure [106, 150]. Apart from the in-link count we take into account also the specific anchor text of the in-links. In Section 6.4 we list typical errors produced by this method.

In Section 6.6.2 we evaluate weaknesses of Spotlight [107] in disambiguation of rare entities and compare its results with structural context representation.

Pretty much state of the art tool for named entity recognition nowadays is Stanford Named Entity Recognizer [60]. It can be used as a Java Library. We use it as well for named entities recognition. The recognized entities are then disambiguated with our own algorithms.

Dictionary based approach is some times used by tools that work with large knowledge bases with a comprehensive set of possible surface forms like DBpedia Spotlight [107]. Here often dictionary-matching AhoCorasick [1] algorithm is used. It locates elements of a finite set of strings within an input text. It matches all patterns simultaneously and thus is relatively fast. The complexity of the algorithm is linear in the length of the patterns plus the length of the searched text plus the number of output matches.
Chapter 4

Building the Knowledge Base

In this chapter we cover approaches to building a semantic web knowledge base that we initially examined. In Section 4.1 we describe the process of extracting knowledge from Wikipedia used in DBpedia \cite{20,7} that we applied in Czech environment. In Section 4.2 and Section 4.3 we show our approach to general semantic web content crawling.

A well managed knowledge base is crucial for correct entity linking. It also provides a source of additional structured information for textual documents enrichment. We cover the process of porting global DBpedia project to Czech Republic. Czech DBpedia is not directly connected to our approach to named entity recognition and disambiguation. We mostly evaluate our algorithms on English texts and use the English version of DBpedia, because of the bigger extent of English DBpedia. But the description of creation of Czech DBpedia provides a good illustration of the process of creating similar knowledge bases based on Wikipedia in general. Still there are advantages of having a local clone of DBpedia:

- Similarly to local Wikipedias, local DBpedias usually provide more comprehensive information about specific local entities, like geographical data (smaller Czech cities, mountains, rivers, lakes), data about important persons (Czech politicians, movie directors, writers, etc.).

- As such, this dataset may serve as a base for various mashup application or automatic data processing tools. Apart from the range of locally specific data, the language of entries and their direct connection to Czech Wikipedia pages might be an advantage, too.

- Thanks to tens of manually created mapping rules, Czech DBpedia is a good ontology mapping dictionary as well. It may serve as a central hub providing a set of common identifiers together with basic properties of identified entities.

- All machine readable data on DBpedia have a direct connection to corresponding Wikipedia articles, where the information is presented in an unstructured way, usually as a plain text. Thus Czech DBpedia might serve as a testing dataset for various natural language processing and information retrieval approaches focusing on Czech language.
CHAPTER 4. BUILDING THE KNOWLEDGE BASE

The porting of DBpedia to Czech Republic was initiated and lead by the author of this doctoral thesis. Thus we include a short report here for completeness.

Further we elaborate possibilities of collecting additional structured data from general resources on the web. The possibilities of scalable crawling are described in Section 4.2. This work was conducted in cooperation with Ondřej Klimpera as part of his master thesis proposed and supervised by the author of this doctoral thesis. More details on the crawling topic can be found in this thesis [90], a short report is provided in [A.9].

4.1 Building Czech DBpedia

In order to obtain raw data from Wikipedia, we use the DBpedia Extraction Framework [143]. This is a module based framework maintained by the international DBpedia team. Wikipedia provides freely accessible dumps of the whole encyclopaedia database [160]. The framework thus downloads recent dumps of all Wikipedia pages covering all topics described on Wikipedia. The pages are downloaded in the source format marked by Wiki markup [95]. These source files are parsed. Data are extracted from parsed pages using various extractors. An extractor is a mapping from a page node to a graph of statements about it.

Various information can be obtained from Wikipedia pages. It is quite easy to get names of entities and extract their connections by analysis of links between corresponding Wikipedia articles. However, the core of the extraction process is the retrieval of information contained in so called infoboxes. An example of such an infobox in a Wikipedia article is shown in Figure 4.1.

Figure 4.1: Infobox example taken from Czech Wikipedia. On the right side, there is a source code of this Wikipedia infobox written using Wiki markup.

In the case of the Czech DBpedia, we use following extractors provided by the extraction
4.1. BUILDING CZECH DBPEDIA

framework:

- Label Extractor (extracts labels of entities from titles of corresponding articles).
- Geo Extractor (extracts geographic coordinates).
- Page Links Extractor (extracts internal links between DBpedia instances from the internal page links between Wikipedia articles).
- Wiki Page Extractor (extracts links to corresponding articles on Wikipedia).
- Infobox Extractor (extracts all properties from all infoboxes).
- Mapping Extractor (extracts structured data based on hand-generated mappings of Wikipedia infoboxes to the DBpedia ontology).

It is important to note the difference between Infobox Extractor and Mapping Extractor. Consider the source code from Figure 4.1. The Infobox Extractor extracts this information as it is written in the source code. Property names are not cleaned, there is no consistent ontology for the infobox dataset. Thus generating RDF triples like:

\[
\text{<http://cs.dbpedia.org/resource/Albert_Einstein>}
\text{<http://cs.dbpedia.org/property/misto_narozeni>}
\text{<http://cs.dbpedia.org/resource/Ulm>}
\]

Unfortunately, infoboxes on Wikipedia are inconsistent and it is usual that same property is described in many different ways. Someone can call the same property misto_narozeni (placeOfBirth), whereas someone else might use puvod (origin), or narozeni_misto (birthPlace).

The answer to these difficulties is the Mapping Extractor which uses hand written rules that map different patterns used in Wikipedia infoboxes to a consistent ontology of DBpedia.

For our dataset we manually created rules covering more than 40 types of entities (e.g. cities, politicians, actors, writers). Apart from server administration of the Czech version this is our main contribution to Czech DBpedia. Totally, there are currently about 789 Infobox templates on Czech Wikipedia that can be potentially mapped to DBpedia ontology. Mappings can be edited via the Mappings Wiki [42]. As the mapping effort is community driven, everyone can join and help creating and maintaining mapping rules. In order to create new mapping rules, the user has to have basic knowledge of the Wikipedia syntax.

Patronage over the project of Czech DBpedia has recently been taken by University of Economics[1] and the development and maintenance continues in cooperation with doc. Ing. Vojtěch Svatěk, Dr. and Ing. Václav Zeman.

---

4.2 Semantic Web Data Sources Crawling

DBpedia is one of the biggest and most general Linked Data resources on the web. However, it is not the only one. The rapid growth of Linked Data resources on the web poses new challenges to data consumption from semantic web. The Linked Data cloud can be viewed as one big interconnected graph of linked data sources. Data can be accessed via SPARQL endpoints, in the form of RDF data files or they can be embedded in ordinary web pages - for example in the form of RDFa, microdata or microformats.

Semantic web resources, similarly to ordinary web resources (i.e. web pages) are completely decentralized and interlinked. However, there is no implicit hub that would provide access to all available resources on one place. The distributed nature of data sources offers many advantages. Anyone can contribute to the global knowledge base without a limitation. The whole infrastructure is very variable and flexible. On the other side, the distributed nature of these data sources brings complications for data consumers.

Often individual data sources do not offer a SPARQL endpoint to query contained data and thus crawling data dumps or even individual data records is the only way, how to work with provided data. Also when we want to work with data from multiple resources, we have to integrate them.

We propose a new approach to semantic data crawling. We developed a semantic web crawling framework that supports all major ways to obtain data from semantic web (querying SPARQL endpoints, loading RDF dumps, crawling individual RDF resources and extraction of semantic data annotated directly in ordinary web pages).

Our framework is designed as a web service that accepts individual crawling requests (tasks), processes them and returns crawling results in desired format. In order to support the scalability needed for crawling web resources the whole service runs on a Hadoop computer cluster using MapReduce algorithm to distribute the workload.

This is a shift from current approaches where either individuals have to run a crawler on their own infrastructure, or they have to rely on a centralised crawler of a semantic web search engine, where the possibility of influencing individual crawling tasks is limited. Our crawler provides on-demand crawling capabilities.

4.2.1 Robots.txt and Load Balancing

The Robots Exclusion Protocol published inside robots.txt files is a valuable resource of information for crawlers (an example of such file is displayed in Figure 4.2). In order not to crawl sensitive information and not to overload crawled servers it is important to respect disallowed pages and information about crawling delays between requests to one host.

The requirements obtained from robots.txt need to be applied across whole computer cluster, the crawler is running on. We maintain an API providing site information to each

---

2The documentation and source codes are available on [http://research.i-lasek.cz/projects/semantic-web-crawler](http://research.i-lasek.cz/projects/semantic-web-crawler)
# Hash symbols introduces a comment

# Standard part of robots.txt

User-Agent: *
Disallow: /private/

# Crawl delay set to 10 seconds for all crawlers
Crawl-Delay: 10

# Non standard part of robots.txt
User-Agent: googlebot
Disallow: /not/for/google/

# Page specifically set as allowed for crawling robots
Allow: /google/info.html

# Specific location of host sitemap resource
Sitemap: http://example.com/sitemap.xml

Figure 4.2: Robots.txt example.

node in the cluster. Robots.txt data itself are stored using caching framework EHCache\(^3\). Each crawling node in the cluster asks the API to provide information for given URL, which is split into domain and its path part. Domain is then looked up in the cache. If a match is found, its path is verified against disallowed patterns. Additionally, if given URL is allowed to be crawled, next free time slot for crawling is computed according to Crawl-delay record. API then sends a response to crawling node, which waits until the allowed time comes or leaves the resource uncrawled. Figure 4.3 illustrates this situation.

The disadvantage is a rapid slowdown of crawling. When the crawler deals with hundreds of resources from the same host, the crawling delay creates a queue of waiting nodes to take their turn. To minimize probability that more nodes need data from the same host, the URL input list for each crawling depth is shuffled to mix site’s resources as much as possible.

### 4.2.2 Semantic Sitemaps

Another valuable source of crawling information are sitemap XML files \(^4\) (an example sitemap is shown in Figure 4.4). Our crawler is able to extract three types of information from them: URL resources, sitemap index files and semantic data dump locations. Found URLs are just added to crawling seed-list. Sitemap index files provide information where to find more XML data on the same host. Data dump locations are gold-mines for the

\(^3\)EHCache is a Java open-source, standards-based framework used for large data caching. For more information see \(\text{http://ehcache.org}\)

\(^4\)
crawler. They contain sets of RDF triples, which were already collected from the host by its owner.

When requested to crawl a sitemap, the crawler guarantees crawling all its subresources independently on selected crawling depth. This applies to cases where the sitemap is marked in a user’s request.

Otherwise, the user can explicitly set if crawler should process such resources during crawling ordinary resources in breadth. The reason why this is optional comes from the fact that sometimes a user might want to crawl many different resources and do not want to stick with large data dumps that are a kind of special type of resources.

4.2.3 SPARQL Endpoints

Another way how to make semantic data accessible is to provide a SPARQL endpoint. Servers like DBpedia \[7\] have an endpoint API consuming SPARQL queries. We designed the crawler so that it is able to send HTTP GET requests to given endpoints, which is the most common way of their access. Results are then retrieved and used for further processing, i.e. further crawling of contained resources, if requested – according to a requested crawling depth. A user has to provide just a SPARQL endpoint URL and list of param-value pairs shaping the request URL together with a SPARQL query.

The concrete SPARQL query has to be designed by the user. Sometimes it is quite difficult to propose a SPARQL query to discover some useful data with limited knowledge of the target data source. There exist initiatives that aim to add schemata to Linked Data cloud to facilitate easier querying \[79\]. Currently there is no standard for this but it could help the users in the future.
4.2. SEMANTIC WEB DATA SOURCES CRAWLING

Figure 4.4: Sitemap.xml example with Sitemap Semantic extension data. Comments denote the data source locations processed by our crawler.

4.2.4 Project Architecture and Implementation

Our crawler uses MapReduce model implementation provided by Apache Hadoop framework\(^4\). See Section 3.1.3 for detailed description. Crawling of web resources is done by launching series of dependent MapReduce jobs, where each one has a designated task to be performed. Details about particular jobs are described in Section 4.2.6.

The crawler is created as a self-sufficient Java application able of running on a computer cluster or in a stand-alone mode. There are several components which shape a Semantic web crawler.

The application building blocks are a graph database to store found owl:sameAs state-

\(^4\)Apache Hadoop project homepage: \(\text{http://hadoop.apache.org}\)
ments, Hadoop as a MapReduce implementation dedicated to perform crawling and a light-weight web server used for user-computer communication.

There is no need to install any additional software. All components are Java managed. Although application owner has to go through some configuration, which is necessary and deals with a configuration of the cluster architecture and available storage locations.

### 4.2.5 Merging Same Entities

We use the Neo4j\(^5\) graph database to store information about same resources. It is an open-source, pure Java graph database implementation. This concept was chosen according to main data load presented by stored \texttt{owl:sameAs} statements, which naturally leads to a richer graph structure.

Apart from pure crawling, the crawler also collects and groups same resources using different URIs in subgraphs of equal entities. There is always chosen (randomly, but can be changed via the API) one URI, which is set as a reference identifier (e.g. DBpedia URI). The crawler then replaces every part of crawled RDF triples (subject, predicate or object), from final set of crawled data with this reference value.

The objective is to group all information about the same entity under one identifier so that the end user does not have to join these information manually every time some resources are crawled. Finally the other URIs from the same group are added to resulting data with \texttt{owl:sameAs} to preserve also the original identifiers.

### 4.2.6 Hadoop Implementation

Next used component is the Hadoop framework. As other application components this is also an open-source project dedicated to implement distributed data processing using MapReduce programming model.

Crawling web resources consists of several MapReduce jobs. It starts with user’s request with a list of various types of resources (URLs, Sitemaps and SPARQL queries to selected SPARQL endpoints). The crawler creates two job feeders. One is responsible for creating an initial seed-list \texttt{SeedListJobFeeder}, second one deals with crawling web resources \texttt{CrawlingJobFeeder}.

#### 4.2.6.1 Seed List Creation

To create a seed list, it has to be checked if there are some robots.txt and sitemap resources. If so, a MapReduce jobs are created to crawl them and discover new locations, which are later added to the seed-list. Processing robots.txt resources is quite easy. The disallowed patterns are stored. Sitemap locations are collected and processed. Crawling sitemaps is a little bit more difficult, as each sitemap can contain references to other sitemaps on the server. The crawler has to crawl them in depth and each level requires a new MapReduce job.

\(^5\)Neo4j project homepage: http://neo4j.org
4.2. SEMANTIC WEB DATA SOURCES CRAWLING

4.2.6.2 Crawling Jobs

Master node can now create a seed list by adding all found and user provided resources. The result is just a plain text file with one URL per line. This output is given to CrawlingJobFeeder which starts the 0 depth crawling job (NDepthCrawlingJob). The simplified draft of NDepthCrawling job MapReduce implementation is shown in Figure 4.5. Physically reduce() function has multiple outputs where triples and owl:sameAs statements are written separately. In addition, each triple part (subject, predicate or object) is written to the output as a newly found resource for the next depth level of crawling.

```java
class NDepthCrawlingJobMapReduce {

  map(Key key, Value value, Collector<Key, Value> collector) {
    timestamp = getNextFreeCrawlingTime(url);
    do {
      wait;
    } while (currentTime < timestamp);
    extractTriples(url, collector); // Key is a triple itself
  }

  reduce(Key key, Value value, Collector<Key, Value> collector) {
    if (isOwlSameAsStatement(key)) {
      collectOwlSameAsStatement(getOwlSameAsFor(key))
    }
    collectTriple(key)
  }
}
```

Figure 4.5: Pseudo-code Map() and Reduce() for crawling.

After this job is finished, and the final crawling depth wasn’t reached, there has to be created a list of URLs for next crawling depth. This handles NDepthUrlList, which compares previously crawled URLs and the newly found and filters out those resources that have not been processed yet.

Finally in MergeTriplesJob collected triples are checked against the database of owl:sameAs statements and different URIs representing the same entity are replaced with one common URI.

4.2.6.3 MapReduce Job Launching

When a user submits a request for crawling to the API, it is added to the queue of waiting jobs. These requests are processed in separate threads managed by a thread pool of fixed size. There is a MapReduceJobRunner that creates mentioned job feeders and handles their submission to the cluster. This class also handles job status and progress monitoring and provides its information to the user. The whole process is shown in Figure 4.6.
4.2.7 Data Extraction

We use Any23 framework\(^6\) for data extraction from resources in various data formats and from ordinary web pages. Any23 is a universal and very useful framework, but some of its parts needed more research and extracting mechanism speed up. Especially dealing with HTML5 Microdata encountered a large application slowdown. As a rough patch we added a caching mechanism to make Microdata extraction faster.

4.2.8 REST API and Web Server

RESTful API is implemented using Jersey framework\(^7\). It includes Grizzly server which listens on a selected port. Supported communication data formats are XML and JSON.

4.3 Testing on a Hadoop Cluster

We tested our crawler on a small scale cluster. The cluster structure is shown in Figure 4.7. Where each node had its own local 20 GB storage. Node one has in addition a NFS mounted storage with 1TB available disk space that were used to store job results, log files, and database file structure.

We ran a test downloading data from [http://dbpedia.org](http://dbpedia.org) and [http://data.nytimes.com](http://data.nytimes.com) semantic dump files. The crawling results contained over 30 million triples in more than 4.2GB of data. Test ran on 2, 4 and 7 cluster nodes. We observed how long did it take the crawler to download and extract data and time to merge crawled triples in final data set.

Figure 4.8 shows running times (in minutes) for the crawling task. We measured the overall processing time (All Jobs) and also the time spent on two main subtasks: crawling phase (NDepthCrawlingJob) and merging results into the final dataset (TriplesMergeJob). The 7* denotes another run on 7 cluster nodes after some tune up of Hadoop configuration.

The improvement in performance caused by employing more processing nodes was not as big as we expected. One of the main causes was the effect of load balancing policy, while we crawled a limited count of data sources and it was necessary not to overload target servers.

\(^6\)Any23 project homepage: [http://any23.org](http://any23.org)
\(^7\)Jersey – a lightweight framework for API services [http://jersey.java.net](http://jersey.java.net)
The other problem was an architectural issue in the merging phase, where we employed a shared storage for keeping information about the same entities marked with `owl:sameAs`. This shared storage turned out to be a significant bottleneck.

Next, we were interested in processing individual resources, especially, how many triples were extracted in one second, how long did it take to retrieve the resource from web and how long did the crawler have to wait according to crawling delay read from robots.txt. Figure 4.9(a) shows percentage share of processes used for extracting triples from one URL. Average size of a resource was 270kB containing approximately 1600 triples. In average, there were about 10 triples per millisecond extracted on one node.

On the other hand processing data dump files with average size over 1GB containing over 9 million triples has a different task percentage distribution. Here the most time is spent on extracting data. However the extraction is faster as the overhead of opening and closing input is saved. There were extracted about 19 triples per millisecond on one node. The time consumption distribution of individual tasks is shown in Figure 4.9(b).

We found out that downloading times per kB differ dramatically when we compared data dumps and downloading individual resources. Data dumps are often served from a different location as a static content and thus the downloading is much faster. Contrary, individual resources are often served dynamically and thus response times are much longer. It might take up to several seconds to establish a connection.

### 4.4 Summary

Our experiments show that there is only a little gain in crawling several selected linked data resources separately. The scalability of the crawling process is often limited by crawling constraints when querying for these resources. The parallel processing brings then suboptimal results, because the crawler has to wait not to overload the target machines anyway.

---

*Data dumps are often compressed. This number corresponds to uncompressed file size.*
Figure 4.8: Dependency of time to finish a crawling task on a number of cluster nodes. Time is shown in minutes [90].

Also the quality of datasets crawled in this way is low. Data are often messy, inconsistent and incomplete.

The main Linked Data resources provide the option to download whole data dumps in a compressed format. Collecting selected data dumps is more effective with this regard. Loading of data is much faster. But even in the case of dumps quite extensive clean-up is necessary. We elaborate more on this topic in Section 5.4.3.

On the other hand our focused crawler still provides a unified framework to load data from various sources and can be used to keep our knowledge base fresh by re-crawling and seamless integration of several selected data sources building our knowledge base.

The massive scalability enabled by Hadoop could be a huge benefit in the future when Linked Data resources consolidate and their performance (currently limited by high crawling delays) grows.
4.4. SUMMARY

(a) Crawling individual URLs

(b) Crawling data dumps

Figure 4.9: Time consumption of URL processing tasks [90].
Chapter 5

Entity Linking with Wikipedia

Based on the results of general linked data resources crawling, we decided to take a step back and stick only with Wikipedia as a single knowledge base for our experiments. Even with only a single set of dumps the data processing is quite complex and data need proper cleaning before they can be used. We touch more on this topic in Section 5.4.

In this chapter, we describe our approach to named entity recognition and linking to Wikipedia. In Section 5.1 we formally define the problem of named entity recognition and disambiguation. In Section 5.2 we define structures in Wikipedia, then in Section 5.3 we formulate our working hypothesis. Section 5.4 describes structure of indexes obtained from Wikipedia datasets to enable efficient entity linking.

On top of these indexes we implemented various named entity disambiguation approaches using various representations of the named entity context (Section 5.6) and propose enhanced indexes to facilitate more extensive link analysis (Section 5.7). Finally we show the results of concrete implementation of our algorithms in Section 5.8.

Please note that in following chapters terms entity and link in Wikipedia are used interchangeably (if not specified otherwise), because all the presented approaches work with entities linked to Wikipedia and articles in Wikipedia represent the corresponding entities.

5.1 Problem Description

Formally, the problem consists of extracting automatically a set of surface forms $S_t$ of possible named entities (see the definition in Section 3.1.4.4) from a given text $t$ (see Figure 5.1). For each surface form $s \in S_t$, we try to find its correct identifier in a knowledge base. In other words, we try to find a specific entity $e_s$ that is represented by a certain surface form $s \in S_t$ in the given text $t$. Thus we aim at disambiguating a specific meaning of an entity, not only its type. This means for example distinguishing New York – city in United States and New York – a settlement in England with respect to the given text $t$. Both are locations, but on completely different places. The set of all entities $E^{KB}$ represents the whole knowledge base $KB$. 
CHAPTER 5. ENTITY LINKING WITH WIKIPEDIA

Figure 5.1: Named entities recognition and linking process. Surface forms $S_t$ denoting potential named entities are identified in a text $t$. Surface forms can be marked manually or using an algorithm such as Stanford Named Entity Recognizer. For all surface forms possible candidate meanings $C_t$ are generated (we describe the candidate generation process in Section 5.4.2) and from the candidates correct entities are finally selected $E_t$ (see Section 5.6 and Section 5.7).

Challenging are surface forms that can have several candidate entities. We denote the set of candidate entities in the text $t$ as a set $C_t$. The set of candidates for a certain surface form $s$ is $C_s$. Thus $C_t^{KB} \subset C_t \subset E_{KB}$ where $C_t^{KB} = \bigcup_{s \in t} C_s$.

The main task is to select a correct candidate entity $\hat{e}_t \in C_s$ for each surface form $s$ in a text $t$. In this chapter, we describe and compare various measures of the candidate relevance for a specific article or even a specific sentence.

In the following text we restrict ourselves to a single knowledge base derived from Wikipedia. Entities thus coincide with Wikipedia articles. Advantage is that those texts and links are edited by human editors. In the following text we describe how we derive the knowledge base from freely available raw dumps of all Wikipedia articles.

5.2 Data Structures Based on Wikipedia

When a part of a text $s$ (called surface form) in an article in Wikipedia is linked to an entity (another article in Wikipedia), we talk about an entity mention or entity occurrence $m$. We denote the entity mention as the triple $m = (e, p, s)$, which means that the entity $e$ appeared in a paragraph $p$ in Wikipedia under a surface form $s$. The paragraph itself is a part of a Wikipedia article $a$. All entities that are mentioned somewhere in Wikipedia form a set $E$. All Wikipedia articles form a set $A$. Then we can say that $E \subset A$, because identifiers of Wikipedia entities coincide with the links to corresponding articles. But not all entities are also mentioned in some other article on Wikipedia. There are entities that are described in an article but not linked from any other article in Wikipedia.

We use paragraphs as the basic text unit, because they pose a semi-structurally defined part of the text. This is an advantage compared to fixed size text windows for example (eg. 100 words around an entity occurrence). Also paragraph is a reasonably long part of the text and the count of paragraphs in Wikipedia is also still manageable. We were experimenting also with the context represented by sentences (see Section 5.6.2), which is much more computationally demanding, but the results were not that good (see Section 6.2).

Figure 5.2 shows an example of links marked with Wikipedia markup in Wikipedia.
5.2. DATA STRUCTURES BASED ON WIKIPEDIA

dump files. It represents a single paragraph \( p \), which contains 4 entities mentions. For example entity \( e \) ”American Revolutionary War” is mentioned under the surface form \( s \) ”American Revolution”. Thus it forms a mention from Equation 5.1 where the number 67834#2 is an identifier of the paragraph shown in Figure 5.2.

\[
m = (\text{American Revolutionary War}, 67834\#2, \text{American Revolution}) \quad (5.1)
\]

We define a set of mentions of an entity in Equation 5.2. Where \( W \) represents Wikipedia as a knowledge base.

\[
N_e = \left\{ p : \exists s, (e, p, s) \in W \setminus \{e\} \right\} \quad (5.2)
\]

Number of mentions of an entity is then counted according to Equation 5.3.

\[
n_e = |N_e| \quad (5.3)
\]

Analogically we define a set of mentions of an entity under a given surface form 5.4.

\[
N_s^e = \left\{ p : (e, p, s) \in W \setminus \{e\} \right\} \quad (5.4)
\]

Then we can count the number of mentions of an entity under a given surface form 5.5.

\[
n_s^e = |N_s^e| \quad (5.5)
\]

We define a set of candidates \( C_s \) for a surface form \( s \) according to Equation 5.6.

\[
C_s = \left\{ e : \exists p, (e, p, s) \in W \setminus \{e\} \right\} \quad (5.6)
\]

The count of candidates is then shown in Equation 5.7.

\[
n_s = |C_s| \quad (5.7)
\]
Additionally we assume that the candidate set is an ordered set (Equation 5.8).

\[ C_s = \{e_1^s, e_2^s, \ldots, e_n^s\} \] (5.8)

Where for \( n_{e_i}^s \) applies Equation 5.9.

\[ n_{e_i}^s \geq n_{e_{i+1}}^s, \quad i \geq 1, \quad i \leq n_s - 1, \quad n_{e_n}^s > 0 \] (5.9)

A surface form \( s_n \) is ambiguous when it has at least two candidates. In our knowledge base it means that it was mentioned with at least two different entities. All ambiguous surface forms in Wikipedia form a set \( S_a \) (Equation 5.10).

\[ S_a = \{s : n_s \geq 2\} \] (5.10)

Further in our metrics we often count entity co-occurrences. A co-occurrence of two entities \( e_i \) and \( e_j \) means that two entities occurred in the same paragraph. Number of co-occurrences \( n_{e_i,e_j} \) is expressed in Equation 5.11.

\[ n_{e_i,e_j} = |\{p : (e_i, p, s_i) \in W \setminus \{e_i\}, (e_j, p, s_j) \in W \setminus \{e_j\}\}| \] (5.11)

### 5.3 Working Hypothesis

Our working hypothesis is that big \( n_{e_i,e_j} \) indicates that \( \hat{e}_i = e_i \) and \( \hat{e}_j = e_j \) for an arbitrary text \( t \). In other words that a high number of co-occurrences of two entities can help us to select the correct candidate. Formally this hypothesis is expressed in Equation 5.12.

\[ \forall t, \forall s_1, s_2 \in S_t, \hat{e}_{s_1}^t \in C_{s_1}, \hat{e}_{s_2}^t \in C_{s_2}, \forall e_i \in C_{s_1}, \forall e_j \in C_{s_2} : n_{e_i,e_j} \geq n_{e_i,e_j} \] (5.12)

### 5.4 Indexing Wikipedia for Disambiguation

In order to improve our disambiguation algorithms we decided to examine deeper Wikipedia datasets. We focused on the link structure, ambiguity of individual entities contained in Wikipedia and variability of surface forms.

Also our first experiments (see Section 6.2) showed that more thorough dataset cleaning is needed. In following Section 5.4.1, Section 5.4.2 and Section 5.4.3 we describe how we cleaned rough Wikipedia datasets and indexed them for further processing.

We will show that after cleaning there is about \( 10^8 \) entities mentions in Wikipedia that we work with. This can build up to \( 10^{16} \) co-occurrences. In Section 5.4.5 and Section 5.4.6 we describe how we deal with this massive amount of data and the way how we shifted from traditional relational database processing to custom inverted indexes and in-memory processing. We show how our sorted indexes can reduce the processing time of dumps from the order of days to the order of minutes.
5.4. INDEXING WIKIPEDIA FOR DISAMBIGUATION

5.4.1 Basic Indexes

In order to support querying on top of data extracted from Wikipedia, we form several indexes that are used for further processing. They are stored in a simple form of CSV files. In this form they can be simply loaded into a relational database or any other type of data store.

The main indexed structures that we maintain are:

- **Entity index** - list of all entities (i.e. links) found in paragraphs of Wikipedia articles.

- **Surface form index** – list of all surface forms (i.e. anchor texts of links) found in paragraphs of Wikipedia articles.

- **Paragraph index** – list of all paragraphs of all Wikipedia articles. The list contains internal Wikipedia identifier of the article, the order of the paragraph in the article and the plain paragraph text.

- **Entities mentions index** – the most important index that contains all entities mentions in Wikipedia $M^W$ (Equation 5.13). This index is used to count statistics about structure of the links in Wikipedia, to extract candidates and to form in-memory indexes described in Section 5.4.2.

$$M^W = \{m = (e, p, s), m \in W \setminus \{e\}\} \quad (5.13)$$

5.4.2 In-memory Indexes

We maintain 2 additional in-memory indexes to support disambiguation queries. Here we describe their structure. Both indexes are based on an in-memory key value store. More technical details are provided in Section 5.4.5.

The **candidate index** contains all surface forms $s$ found in Wikipedia articles together with their candidates $C_s$ (this corresponds to links $c_s$ and their anchor texts $s$ extracted from Wikipedia). Structure of the index is denoted in Figure 5.3. For the notation please refer to the Section 5.1. Together with each candidate, we keep an information, how many times it occurred under the given surface form.

The **co-occurrence index** enables disambiguation based on the number of co-occurrences of two entities in the same paragraph. The key in this case is the concatenation of identifiers of two co-occurring entities (separated by a separator to form a unique identifier of the entity couple). The value is then the absolute number of co-occurrences. The structure is shown in Figure 5.4.
Figure 5.3: Structure of the candidate index for $s_1, s_2 \in S^W$. Each record contains sorted set of candidates (see Equation 5.6) together with counts of their occurrences under a given surface form (see Equation 5.5).

Figure 5.4: Structure of the co-occurrence index.

5.4.3 Indexing Pipeline

In order to index Wikipedia we developed an indexing pipeline (Figure 5.5). Here we describe individual steps that we perform in order to query Wikipedia datasets.

**Step 1: Wikipedia Dump File Loader.** First we load Wikipedia dump files. They are freely available for download in Wikipedia dump repositories.\(^1\)

**Step 2: Wikipedia Parser.** The source files for all Wikipedia articles are provided in the Wiki markup.\(^2\) In order to parse these files, we use Java Wikipedia API (Bliki engine).\(^2\) We customized this parser in order to extract among all links (entities mentions) and their context.

**Basic Indexes.** Wikipedia Parser produces basic indexes described in Section 5.4.1. These indexes are not directly queried. They serve as an intermediate step for construction of the in-memory indexes.

\(^1\)http://dumps.wikimedia.org/enwiki
\(^2\)Our parser is available for download from http://research.i-lasek.cz/projects/wikipedia-indexer
5.4. INDEXING WIKIPEDIA FOR DISAMBIGUATION

![Wikipedia indexing pipeline diagram](image)

**Figure 5.5:** Wikipedia indexing pipeline.

**Step 3: Cleaner.** During our first experiments (see Section 6.2) it turned out that rough data extracted from Wikipedia is too messy. The dump files contain many irrelevant pages that do not represent entities we are interested in. For example dump files include pages representing user profiles, categories and other similar service or meta data pages.

We decided to clean them up. Usually the service pages can be recognized by the title. For example they often contain the colon in their titles and in URLs. Categories are denoted as `Category:Category name` and the link then looks like this `http://en.wikipedia.org/wiki/Category:Category_name`. Analogically user pages can be recognized by the `User:` prefix. So our first try was to remove all pages with this kind of special title and to skip all links to them.

Table 5.1: Results of cleaning of service pages – size of basic indexes.

<table>
<thead>
<tr>
<th>Index type</th>
<th>Count of items</th>
<th>Index file</th>
<th>Index file size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities (E_0)</td>
<td>10 389 236</td>
<td>entities.csv</td>
<td>503 MB</td>
</tr>
<tr>
<td>Entities mentions (M_0)</td>
<td>100 623 276</td>
<td>entities_mentions.csv</td>
<td>6.8 GB</td>
</tr>
<tr>
<td>Paragraphs</td>
<td>26 267 287</td>
<td>paragraphs.csv</td>
<td>8.6 GB</td>
</tr>
<tr>
<td>Surface forms</td>
<td>12 231 856</td>
<td>surface_forms.csv</td>
<td>231 MB</td>
</tr>
</tbody>
</table>

In the Table 5.1, we show the resulting index sizes after removing the service pages. In the tables we work only with entities \(E_0\) that are extracted from links on Wikipedia. In other words with entities that are mentioned in some paragraph in Wikipedia (Equation 5.14).

\[
E_0 = \{ e : n_e > 0 \} \quad (5.14)
\]

Therefore the number of entities should be lower than the total number of articles in Wikipedia, because not all entities are linked from some Wikipedia article. The cleaning
removed 3,380,698 entities from entity index. But still the number of articles is much higher than the 4.5 million English articles promoted on Wikipedia homepage.

We thus continued in the cleaning and omitted disambiguation pages (dataset reduction can be seen in the Table 5.2) and further removed links to anchors within article texts that are in our case redundant with links to full articles (see Table 5.3).

Table 5.2: Results of removing disambiguation pages – size of basic indexes.

<table>
<thead>
<tr>
<th>Index type</th>
<th>Count of items</th>
<th>Index file</th>
<th>Index file size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities ($E_1$)</td>
<td>10,115,707</td>
<td>entities.csv</td>
<td>488 MB</td>
</tr>
<tr>
<td>Entities mentions ($M_1$)</td>
<td>99,567,171</td>
<td>entities_mentions.csv</td>
<td>6.7 GB</td>
</tr>
</tbody>
</table>

But still after the links filtering we were far from the real count of entities on Wikipedia. In our indexes remained many incorrect links probably due to typing errors, links to dead pages, redirects etc. So we decided to take another approach. Wikipedia provides also SQL dumps containing among others meta data tables describing properties of individual pages within Wikipedia.

The central storage is the page table, which lists real pages in Wikipedia together with their properties such as information about redirects etc. We imported these SQL dumps and examined them. The count of all records in the page table was in case of our dump (from 4.6.2013) 7,707,681. This is still higher number than the real number of English articles stated on Wikipedia homepage. However, titles in the page table repeat. They differ in the namespace where they appear (see the comparison in Figure 5.6).

Finally when we subtract the number of redirects (see Figure 5.7), we will get to more reasonable numbers. Redirects are essentially redundant, they state for another possible name of an entity.

With the help of the page table, we went through all the links that we extracted from Wikipedia dumps and kept only the ones that we found also in the page table. The results of the final cleaning step are shown in Table 5.4.

Table 5.3: Results of removing links to parts of articles – size of basic indexes.

<table>
<thead>
<tr>
<th>Index type</th>
<th>Count of items</th>
<th>Index file</th>
<th>Index file size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities ($E_2$)</td>
<td>9,709,063</td>
<td>entities.csv</td>
<td>461 MB</td>
</tr>
<tr>
<td>Entities mentions ($M_2$)</td>
<td>98,791,825</td>
<td>entities_mentions.csv</td>
<td>6.7 GB</td>
</tr>
</tbody>
</table>

Step 4: Sorter. The most essential extracted dataset for indexing is the Entities mentions dataset which contains all mentions $m = (e, p, s)$. We use the entity mention index $M_3$ that resulted from the final cleaning. For easier processing during indexing we create three copies of the dataset sorted by entities, paragraphs and surface forms. They are stored as text files on the disk.

---

3http://www.wikipedia.org
4Anchor marks a part of a web page. Links to an anchor link to the corresponding part of the web page (e.g. a section, a certain headline) and not to the whole page.
Step 5: In-memory Indexer. We further use the sorted datasets to produce in-memory indexes discussed in Section 5.4.2. The indexes are then used to produce Wikipedia statistics discussed in Section 6.1.

5.4.4 Observation

After the final cleaning we got to about 85% reduction of the count of entities whereas the reduction of entities mentions was only about 30%. This is interesting. The total count of real entities in our dataset was according to Wikipedia statistic about 4.2 million. This means that only about one third of entities in Wikipedia is linked from some other article in Wikipedia. Further these experiments show that about 30% of links in Wikipedia are

<table>
<thead>
<tr>
<th>Index type</th>
<th>Count of items</th>
<th>Index file</th>
<th>Index file size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities ($E_3$)</td>
<td>1,565,389</td>
<td>entities.csv</td>
<td>77 MB</td>
</tr>
<tr>
<td>Entities mentions ($M_3$)</td>
<td>70,537,227</td>
<td>entities_mentions.csv</td>
<td>4.6 GB</td>
</tr>
</tbody>
</table>
either links to some sort of service pages or just messy links. These counts also show that
the difference between 4.2 million and 1.6 million (i.e. 2.6 million) entities (or topics)
on Wikipedia are rare entities that are not linked from other articles. This might pose
important implications for disambiguation and specificity of entities.

We discuss further aspects of the structure of Wiki links and their use in disambiguation
process in Chapter 6.

5.4.5 Used Technologies

We started the development using Java. The first implementation of the disambiguation
service as well as indexing pipeline was done purely in Java. However, due to performance
issues (detailed configuration of our server is provided in Chapter 6), we later switched to
NodeJS [120].

From the development view, JavaScript (a scripting language) powered by NodeJS
enables faster changes during implementation and tuning of the indexing pipeline. Also
from the performance point of view it improved execution times of many particular steps
of the indexing pipeline. It enables asynchronous execution so that many indexing tasks
can perform computations in parallel while waiting for slower operations such as disk IO
or database queries.

Also databases are usually performing quite well while serving multiple requests. So
processing database queries in parallel brought also huge time savings and better CPU
utilization.

Figure 5.8 shows the overall architecture of the indexing pipeline. We store Wikipedia
dumps on disk. Part of the dumps (i.e. page or redirect table from Section 5.4.3) is stored
in a MySQL database. The indexing pipeline is completely implemented in NodeJS as
well as the benchmark builder. In-memory indexes are stored in Redis in-memory key
value store [135]. Surface form is used as a key by candidate index, combination of two
co-occurring entities identifiers is used as a key in the co-occurrence index.

The disambiguation service is also implemented using NodeJS. It calls third party
component Stanford NER [60] for recognition of the entities. The entities are then disam-
biguated – links to Wikipedia are assigned to them. Finally we implemented an evaluator
based on highly ambiguous benchmarks that we produce.

5.4.6 Performance Consideration

The initial version of the indexing pipeline was working with MySQL database and many
computations were performed directly in the database. However with the size of several
tens of millions of records more complicated queries involving joins, and more extensive
aggregations were running too long or sometimes were not feasible.

We were considering the usage of Hadoop in order to process some of the bigger dumps
and datasets produced from Wikipedia. But based on our experience with usage of Hadoop
for crawling described in Section 4.2 we decided to avoid it. The parallel processing on a
cluster would make the indexing pipeline much more difficult to extend and debug. Often
5.4. INDEXING WIKIPEDIA FOR DISAMBIGUATION

We found a solution with in-memory computing.

We filled all entities mentions in memory. With this change the aggregation computations were running orders of magnitude faster than with the database. Consider the SQL table that contains all entities mentions $m = (e, p, s) \in M_3$ (see Table 5.5 for an example).

Table 5.5: Entity mentions SQL table structure.

<table>
<thead>
<tr>
<th>entity</th>
<th>paragraph</th>
<th>surface_form</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$p_1$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$p_2$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

When we want to find counts of all entities co-occurrences we can use the query from Figure 5.9. The problem by this query is its size in combination with the join. Even after all the cleaning the table still contains about 70 million records (Table 5.4). When we join it with itself, we get $490 \cdot 10^{12}$ records that need to be grouped and counted.
CHAPTER 5. ENTITY LINKING WITH WIKIPEDIA

SELECT em1.entity, em2.entity, COUNT(*)
FROM entities_mentions em1
JOIN entities_mentions em2
ON (em1.paragraph = em2.paragraph AND em1.entity != em2.entity)
GROUP BY em1.entity, em2.entity

Figure 5.9: SQL query to retrieve counts of entity co-occurrences.

Of course we could use another approach and query for co-occurrences of each entity. The cleaned entity set contains about 1.5 million entities. That means 1.5 million queries. If we suppose that a single query takes about 0.5 second to process, the whole processing would take about 208 hours (more than 8 days).

With in-memory storage of co-occurrences counts of entities this is a matter of a single walk through the dataset and can be finished in less than one hour (Table 5.6). The Algorithm 3 shows how we count the co-occurrences.

Table 5.6: Processing times of co-occurrence computation.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Processing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive database cross join</td>
<td>∞</td>
</tr>
<tr>
<td>Sequential database querying</td>
<td>8 days</td>
</tr>
<tr>
<td>In-memory processing</td>
<td>&lt; 1 hour</td>
</tr>
</tbody>
</table>

5.4.6.1 Prerequisites

Our approach supposes that the dataset of entities mentions is sorted by paragraphs. This can be easily achieved using bash sort command [149]. It runs quite fast and 4.6GB dataset is sorted in less than one hour.

Our algorithm stores the co-occurrence scores in memory in a hash map so the querying and updating of co-occurrence scores is fast. The structure of the hash map is shown in Table 5.7. Co-occurring entities identifier serves as a key. The memory consumption can be reduced by compressing textual entity identifiers or replacing them with numerical identifiers.

Table 5.7: In-memory co-occurrence storage.

<table>
<thead>
<tr>
<th>Co-occurring entities identifier</th>
<th>Count of co-occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1, e_2$</td>
<td>$n_{e_1,e_2}$</td>
</tr>
<tr>
<td>$e_3, e_4$</td>
<td>$n_{e_3,e_4}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Algorithm 3 Counting of co-occurrences of entities.

\[\text{entityCoOccurrences} \leftarrow \text{initEntityCoOccurrences}()\]
\[\text{entitiesInSameParagraph} \leftarrow \emptyset\]
\[\text{previousParagraph} \leftarrow p_1\]
\[\text{for all } M_i = (e_i, p_i, s_i) \in M = (e, p, s) \text{ do}\]
\[\text{if previousParagraph} = p_i \text{ then}\]
\[\text{add}(e_i, \text{entitiesInSameParagraph})\]
\[\text{else}\]
\[\text{for all } e_j \in \text{entitiesInSameParagraph} \text{ do}\]
\[\text{for all } e_k \in \text{entitiesInSameParagraph} \text{ do}\]
\[\text{if } e_j \neq e_k \text{ then}\]
\[\text{addOrIncrementCoOccurrences(entityCoOcccurrences, } e_j, e_k)\]
\[\text{end if}\]
\[\text{end for}\]
\[\text{end for}\]
\[\text{clear(entitiesInSameParagraph)}\]
\[\text{end if}\]
\[\text{end for}\]
\[\text{return entityCoOccurrences}\]

5.5 Recognition of Named Entities Surface Forms

Remember the difference between TAC 2013 [154] and SIGIR 2014 [30] entity linking evaluation datasets stressed in Section 3.4. By SIGIR the surface forms are not provided as the input. By our evaluation with TAC dataset we work with provided annotations of surface forms. For our other experiments we suppose the real world use case where we usually don’t work with pre-annotated texts. We need a mapping from a plain text to the set of surface forms (Equation 5.15).

\[t \xrightarrow{\text{alg}} S_t\] \hspace{1cm} (5.15)

For identification of named entity surface forms in test set of texts in English, we use Stanford Named Entity Recognizer [60], which is based on the state of the art Conditional Random Fields model [105]. Identified entity surface forms are searched in the knowledge base. The performance of Stanford NER on non-English texts is evaluated in Section 6.3.1.

If there is only one candidate in the knowledge base (Figure 5.3), we are done – the entity is identified. If there are more candidates, we need to proceed with the next step – disambiguation.
CHAPTER 5. ENTITY LINKING WITH WIKIPEDIA

5.6 Various Context Representations for Disambiguation

In this section we describe our first attempts to the named entity disambiguation using various ways of context representation. The experimental results are presented in Section 6.2. In following sections we describe our approach to context representation as a bag of words (Section 5.6.1), as a structure of the text on the level of individual sentences (Section 5.6.2) and our disambiguation method based on entities co-occurrences (Section 5.6.3).

5.6.1 Bag of Words Context Representation – Algorithm 4

For the bag of words model, we partly adapt the approach of [107]. We implemented the described approach on top of our indexes described in Section 5.4.

Having a set of candidates \( C_s \), we need to find the correct one. For each candidate, we compare, how close are texts of paragraphs, the candidate was mentioned in (in the knowledge base) to the currently considered text.

Algorithm 4 Disambiguation using bag of words context representation.

\[
R \leftarrow \text{initDisambiguationResult}()
S_t \leftarrow \text{recognizeSurfaceFormsIn}(t)
\text{for all } s \in S_t \text{ do}
\quad P_s \leftarrow \text{initParagraphIndex}()
\quad C_s \leftarrow \text{getCandidatesFromCandidateIndexFor}(s)
\quad \text{for all } c_s \in C_s \text{ do}
\quad\quad P_{c_s} \leftarrow \text{getParagraphsFromParagraphIndexFor}(c_s)
\quad\quad P_s \leftarrow \text{addRecord}(c_s, P_{c_s}, P_s)
\quad\text{end for}
\quad e_s \leftarrow \text{findBestMatchFor}(t, P_{c_s})
\quad R \leftarrow \text{addToResult}(s, e_s, R)
\text{end for}
\text{return } R
\]

It proved to be valuable for this task, to introduce another index containing all entities. For each entity, we keep track of all concatenated paragraphs, the entity was mentioned in. This serves as one big document representing the bag of words occurring usually with the considered entity. Structure of the index is shown in Figure 5.10. We do not keep a record of a surface form that was used in the specific context. Simply any paragraph that contained the entity under any of its surface form contributes to its context.

We then compare each such textual candidate representation with a target article and select the one that matches the best. For text comparison the vector space model and cosine similarity is used. For weighting individual terms tf-icf weight [107] is used.

Tf-icf measure is an adaptation of the tf-idf [139] measure. The only difference is that the idf part is counted only among selected candidates and not over the whole knowl-
edge base. Thus the discriminator terms specific for the concrete candidate selection are weighted higher. The basic principle is described in pseudocode in Algorithm 4.

For indexing paragraph texts as a bag of words representation and their comparison described in Section 5.6.1, we use a Lucene index.

5.6.2 Sentence Structure Context Representation – Algorithm 5

Another kind of representation that we distinguish is the structure of a sentence where an entity is mentioned. By each such sentence, we identify subject, verb and object triples. If there are multiple subjects or objects, we record them all.

For each entity in our knowledge base, we record the words or phrases that occurred on the position of subject, verb and object in the same sentence as the considered entity. Our hypothesis is that a particular entity co-occurs with certain set of words which can distinguish it from other entities with the same surface form. For each recognized surface form $s$, we store all subject, verb, object triples $r_s$ from the containing sentence forming a set $R_s$.

An approach to extraction of subject verb object triples was described in [141]. Parse trees produced by syntactical parsers for English are examined and based on predefined rules, desired triples are extracted. However, according to the performance, we use a heuristic that involves only faster shallow parsing and does not need to construct the whole parse tree.

From each sentence, we extract first noun phrase as a subject, first verb phrase as a verb and first noun phrase after the verb as an object. If there are more parts of a sentence separated by a comma that do not contain a verb, additional noun phrases are extracted as subjects or objects.

For each candidate entity $c_s \in C_s$ we count the sentence co-occurrence score according to the Equation 5.16.

$$s_{cs} = \sum_{r_s \in R_s} \sum_{r_{cs} \in R_{cs}} ms_{r_s,r_{cs}} + mv_{r_s,c_s} + mo_{r_s,c_s}$$

In Equation 5.16 the subject match score $ms_{r_s,r_{cs}}$ is 1, if the subjects of triples $r_s$ and $r_{cs}$ match, 0 otherwise. Object match score $mo_{r_s,r_{cs}}$ and verb match score $mv_{r_s,r_{cs}}$ are computed analogically. The final sentence co-occurrence score $s_{cs}$ of a candidate $c_s \in C_s$
for a surface form $s$ is summed over all identified triples $R_s$ in the sentence containing the
surface form $s$ and all identified triples $R_c$ in sentences where the candidate $c_s$ occurred
in the knowledge base.

The analysis of a new text containing a surface form $s$ with a set of candidate entities
$C_s$ is described in Algorithm 5.

**Algorithm 5** Disambiguation using sentence structure context representation.

```
v_s ← sentenceContainingSurfaceForm()
R_s ← extractSubjectVerbObjectTriplesFrom(v_s)
C_s ← getCandidatesFromCandidateIndexFor(s)
scs_{best} = 0
c_{best} ← ∅
for all $c_s ∈ C_s$ do
    scs_{c_s,s} = ∑_{r_s ∈ R_s} ∑_{r_c ∈ R_c} m_{s,r_s,c_s} + m_{v,r_s,c_s} + m_{o,r_s,c_s}
    if scs_{best} < scs_{c_s,s} then
        scs_{best} = scs_{c_s,s}
        c_{best} ← c_s
    end if
end for
return c_{best}
```

5.6.2.1 Subject Verb Object Extraction

We completed the task of tokenization, chunking and part of speech tagging with Apache
OpenNLP toolkit\(^5\) and its built in models. However, we are not really satisfied with the
results of chunking. Quite long chunks were often produced, which makes the chunks too
specific for later matching.

The results of subject-verb-object identification on Wikipedia articles are stored in a
relational database.

Because of a high specificity of some chunks the exact matching of identified subject,
verbs and objects was not feasible. Therefore we decided to compare identified phrases
against substrings of triples stored in the database.

5.6.3 Structural Context Representation – Algorithm 6

By structural representation of the context, we aim at capturing relations between entities
rather than their textual representation. An example of such a structural representation is
described in [111], where each entity (represented as a Wikipedia article) is characterized
by a structure of incoming links instead of its textual description.

\(^5\)Apache OpenNLP toolkit – project homepage: http://opennlp.apache.org
Contrary to the approach presented in [111] our structure based model does not compare similarities of individual entities. We are searching for the best combination of candidates for individual surface forms in the analysed text. The whole text represents the context.

Consider for example the following sentence: Michael Bloomberg is the mayor of New York. Simple observation shows that the entity Michael Bloomberg (mayor of New York) co-occurs in the same paragraph in our knowledge base together with the correct entity New York City in United States much more often (88 times) than with the New York in England (0 times).

Because generating all candidate combinations is a very demanding task, we developed a heuristic that quantifies an impact of co-occurrences in the same paragraph.

We construct an incidence matrix $I$ of the size $|C_a| \times |C_a|$. The matrix represents a weighted graph of connections of all candidates in a text $t$. Weights are the co-occurrence measures and are assigned according to Equation 5.17.

$$I_{i,j} = \begin{cases} 
0 & \text{if } s = r \\
0 & \text{if } i = j \\
n_{e_i,e_j} & \text{if } i \neq j \land e_i \in C_s \land e_j \in C_r \land s \neq r \land s, r \in t 
\end{cases} \quad (5.17)$$

The weight $n_{e_i,e_j}$ is counted according to Equation 5.11. So the weight $n_{e_i,e_j}$ (count of paragraphs, where $e_i$ and $e_j$ were mentioned together) is counted only in the case that the candidates represent a different entity $i \neq j$ and belong to a different surface form $s \neq r$. Otherwise it is 0.

Then we compute a score $m_i$ for each candidate as a sum of a line of the matrix representing the candidate (Equation 5.18).

$$m_i = \sum_{j=1}^{|C_i|} I_{i,j} \quad (5.18)$$

The whole disambiguation process of all surface forms in the text $t$ is shown in Algorithm 6. It returns a map of best candidates $B$. The map contains for each surface form the best candidate selected based on its score $m_i$.

### 5.6.4 Summary

First we ran our disambiguation algorithms described in Section 5.6 on smaller datasets (see Section 6.2). By deep examination of the results, we came to several findings:

- Our indexes need to be more thoroughly cleaned.
- Solely the context is not enough for proper disambiguation. Sometimes the context does not provide enough information to disambiguate a particular entity.
Algorithm 6 Disambiguation using basic structural context representation.

\[ S_t \leftarrow extractSurfaceFormsFromArticle(t) \]
\[ C_t \leftarrow getCandidatesForSurfaceForms(S_t) \]
\[ I \leftarrow 0_{|C_t| \times |C_t|} \]
for all \( e_i \in C_t \) do
  for all \( e_j \in C_t \) do
    if \( i \neq j \) and \( e_i \in C_s \) and \( e_j \in C_r \) and \( s \neq r \) then
      \[ I_{i,j} \leftarrow n_{e_i,e_j} \]
    end if
  end for
end for
\[ B \leftarrow initBestCandidatesMap() \]
for all \( s \in S_t \) do
  \[ b \leftarrow 0 \]
  for all \( e_i \in C_s \) do
    \[ m_i = \sum_{j=1}^{|C_t|} I_{i,j} \]
    if \( b < m_i \) then
      \[ b \leftarrow m_i \]
      \[ B \leftarrow putBestCandidateForSurfaceForm(e_i, s, B) \]
    end if
  end for
end for
return \( B \)
Sentence structure context showed to be practically unfeasible in bigger extent than the experiments we ran. It would demand to annotate all sentences in Wikipedia at least with a shallow parser. Moreover the results were not very compelling. So we decided to discontinue further explorations in this direction.

Also the proposed Structural context representation described in Section 5.6.3 was very computationally demanding.

5.7 Co-occurrence Based Algorithms

Based on the results summarized in Section 5.6.4 we proposed several changes described in this section:

- We incorporated the cleaner in our indexing pipeline (Section 5.4.3).
- We decided to include also entity popularity (Section 5.7.1).
- We proposed optimized data structures for effective dealing with the Structural context representation (Section 5.7.2).
- We proposed a new variant of the Structural context representation where we replaced the usage of sum function with maximum (Section 5.7.3).

The experimental results are presented in Section 6.6.

5.7.1 Default and Normalized Most Popular Meaning Disambiguation – Algorithm 7

Based on our findings, we integrated the cleaner in our indexing pipeline. As a baseline method for further examinations, we proposed the Most Popular Meaning disambiguation. The approach is described in Algorithm 7. It simply selects the candidate with the highest \( n_s \) score (see Equation 5.5).

We work with two types of indexes for the Most Popular Meaning algorithm. In the default case we search for exact match of the surface form in the index. The second type uses normalized surface forms instead of the exact match (i.e. it matches for example london and London to the same surface form). For the normalization we use basic information retrieval techniques (see Section 3.1.4.1). We reduce the number of trailing spaces, cast surface forms to lower case, strip diacritics and clean several special characters. However, the order of words is preserved.

Contrary to commonly used popularity measure of entities that count the in-links leading to an entity in Wikipedia [57, 99] we take into account also the concrete surface form.
CHAPTER 5. ENTITY LINKING WITH WIKIPEDIA

Algorithm 7 Disambiguation using sentence structure context representation.

\[
C_s \leftarrow \text{getCandidatesForSurfaceForm}(s) \\
n_{\text{best}} \leftarrow 0 \\
c_{\text{best}} \leftarrow \emptyset \\
\text{for all } e \in C_s \text{ do} \\
\quad \text{if } n_e > n_{\text{best}} \text{ then} \\
\quad \quad c_{\text{best}} \leftarrow e \\
\quad \quad n_{\text{best}} \leftarrow n_e \\
\quad \text{end if} \\
\text{end for} \\
\text{return } c_{\text{best}}
\]

5.7.2 Sum Co-occurrence Disambiguation with Enhanced Co-occurrence Index – Algorithm 8

When we look at the basic structural context representation and the Algorithm 6, we quickly find out that the computational complexity is \(O(|C_t|^2 + |S_t| \times |C_t|^2)\). What is more important in this case, the algorithm executes \(O(|C_t|^2)\) queries on the co-occurrence index. In reality by smaller news articles (for example classical web news) this is to some extent feasible. However, for longer articles with hundreds of surface forms this is too much. Some surface forms can have up to thousand candidates. Thus we end up with millions of queries for co-occurrences to evaluate a single article. Even when we use very fast Redis in-memory store, for production use or even more extensive evaluation this is unacceptable.

So we came up with the enhanced co-occurrence index with different structure. Compare the former index in Figure 5.4 with the new one in Figure 5.11.

![Figure 5.11: Structure of the enhanced co-occurrence index.](image)

Instead of keeping a record for all co-occurring entities pairs, we keep only a single record for each entity. The record then contains the whole list of co-occurring entities together with the counts of co-occurrences with the indexed entity.

The index itself does not improve the performance. We need to adjust the algorithm as well. The Algorithm 8 shows the new approach to disambiguation. For each co-occurring entity \(e_o\) we define the sum \(k_{e_o}\) of all co-occurrences of the entity \(e_o\) with all candidates.
5.7. CO-OCCURRENCE BASED ALGORITHMS

\( f \in C_t \) for surface forms \( s_t \in S_t \) in the disambiguated text \( t \) (Equation 5.19).

\[
k_{e_o} = \sum_{f \in C_t} n_{e_o,f}
\]  

(5.19)

Then we define the set \( S_{e_o} \) of surface forms that have candidates co-occurring with entity \( e_o \) (Equation 5.20).

\[
S_{e_o} = \{ s : \exists p, \exists e_j, \exists s_i, (e_o, p, s_i) \in M, (e_j, p, s) \in M \}
\]  

(5.20)

We define a tuple \( g_{e_o} \) according to Equation 5.21.

\[
g_{e_o} = (k_{e_o}, S_{e_o})
\]  

(5.21)

We define a set \( R_t \) of all entities having a co-occurrence with an entity candidate for some surface form in text \( t \) (Equation 5.22).

\[
R_t = \{ e : \exists s \in S_t, \exists f \in C_s, n_{e,f} > 0 \}
\]  

(5.22)

Finally, all the tuples \( g_{e_o} \) form a set \( G \) (Equation 5.23).

\[
G = \{ g_{e_o} : e_o \in R_t \}
\]  

(5.23)

During the disambiguation, we then look up all tuples \( G_{c_s} \) for each candidate \( c_s \in C_s \) for each surface form \( s \). If the corresponding tuple exists, it means that the candidate co-occurred in Wikipedia with some other candidate from our article. We then check if it is not a candidate of the same surface form. If not, the co-occurrence score of the candidate is increased.

Finally for each surface form, we return the candidate with the best co-occurrence score.

Algorithm 8 Sum co-occurrence disambiguation with enhanced co-occurrence index.

\[
\begin{align*}
S_t & \gets extractSurfaceFormsFromArticle(t) \\
C_t & \gets getCandidatesForSurfaceForms(S_t) \\
G & \gets initCoOccurrenceMap() \\
& \text{for all } c_s \in C_t \text{ do} \\
& \quad G \gets getAndStoreCoOccurrencesInG(c_s, G) \\
& \text{end for} \\
T & \gets initCandidatesScores() \\
& \text{for all } c_s \in C_t \text{ do} \\
& \quad \text{if } G_{c_s} \in G \text{ and } (s \not\in S_{c_s} \text{ or } |S_{c_s}| > 1) \text{ then} \\
& \quad \quad T_{c_s} \gets incrementCandidateScoreBy(k_{c_s}) \\
& \quad \text{end if} \\
& \text{end for} \\
\text{return } selectCandidatesWithBestScoresFrom(T)
\end{align*}
\]
With this kind of disambiguation we need only $O(|C_t|)$ queries on the index. This is much more scalable than the previous case. Additionally the algorithm can be further optimized by skipping obscure candidates $c_s$ that have low $n_s^e$ score for surface forms with more candidates.

5.7.3 Maximum Co-occurrences Disambiguation – Algorithm 9

Sum as a function for candidates disambiguation has a significant drawback. It flattens the influence of a particular candidate pair co-occurrence. Generally very popular entities that occur often with many others can contribute to the scores of unpopular ones and sometimes push the wrong candidate. Maximum is more selective in choosing only one best candidate pair for each candidate.

For maximum co-occurrence score we again work with the incidence matrix $I$ (Equation 5.17). But in order to score candidates, we select only the maximum co-occurrence count (Equation 5.24).

$$m_i = \max_{1 \leq j \leq |C_t|} I_{i,j}$$

For the final disambiguation algorithm (Algorithm 9) using maximum, we have to adjust a little bit the tuple with information about co-occurring candidates. The tuple $g_{e_o,s}$ in Equation 5.25 contains for each co-occurring entity $e_o$ the candidate $c^s_{\text{max}}$ with the maximum co-occurrence score $m_{e_o}$. An item for a given surface form $s$ and entity $e_o$ is in $G$ only if the co-occurrence count $n_{e_o,c^s}$ is the highest co-occurrence count among all candidates $C_t$.

Example of a concrete implementation of the co-occurrence sum and max algorithms in NodeJS is provided in Appendix A

$$G = \{ g_{e_o,s} = (m_{e_o}, c^s_{\text{max}}) \}$$

5.7.4 Summary

With performance optimizations described in Section 5.7 we were able to decrease the average processing time of TAC queries with sum and maximum disambiguation algorithms to about 15 seconds. Shorter texts take up to 5 seconds for disambiguation. We evaluated our methods with general TAC 2013 benchmark (see Section 6.6.1 for the results) and then we tested specifically contextual awareness on our own highly ambiguous benchmark (see Section 6.6.2 for the results).

5.8 SemiTags – Our Entity Linking Tool

SemiTags is our online tool supporting named entity recognition and disambiguation. It provides a graphical user interface, where users can try the service by themselves (Fig-
Algorithm 9 Maximum co-occurrences disambiguation.

\begin{algorithm}
\begin{algorithmic}
\State \( S_t \leftarrow extractSurfaceFormsFromArticle(t) \)
\State \( C_t \leftarrow getCandidatesForSurfaceForms(S_t) \)
\State \( G \leftarrow initCoOccurrenceMap() \)
\ForAll {\( c_s \in C_t \)}
\State \( G \leftarrow getAndStoreCoOccurrencesInG(c_s, s, G) \)
\EndFor
\State \( T \leftarrow initCandidatesScores() \)
\ForAll {\( c_s_1 \in C_t \)}
\If {\( G_{c_s_1, s_2} \in G \)}
\If {\( s_1 \neq s_2 \)}
\If {\( T_{c_s_1} < m_{c_s_1, s_2} \)}
\State \( T_{c_s} \leftarrow m_{c_s_1, s_2} \)
\EndIf
\If {\( T_{c_s_1}^{\text{max}} < m_{c_s_1, s_2} \)}
\State \( T_{c_s_1}^{\text{max}} \leftarrow m_{c_s_1, s_2} \)
\EndIf
\EndIf
\EndIf
\EndFor
\Return \( selectCandidatesWithBestScoresFrom(T) \)
\end{algorithmic}
\end{algorithm}

Users can paste a text to the web form, select the disambiguation method and submit it for recognition. The tool then returns the set of discovered named entities and their identifiers (links to Wikipedia articles describing them) for those that it was able to disambiguate.

Apart from the graphical user interface, we provide also a RESTful web service that enables other programs to exploit results of the recognition and disambiguation. In order to get the results it is necessary to send a POST request with parameters text (the disambiguated text) and preprocess (the method used for disambiguation – default, norm, sum max for exact match and normalized most frequent sense, sum and maximum disambiguation respectively). An example response in JSON format is shown in the Figure 5.13.

Apart from the offset of an entity occurrence in the text, we return its URI in Wikipedia (we trim the http://wikipedia.org/wiki/ prefix, because it is the same for all entities.

The type is one of the four general types provided by Stanford Named Entity Recognizer (person, location, organization and miscellaneous) and serves for a very general categorization of the entity. It is useful only for entities that we were not able to find a link for. When a link is available, it provides much more precise categorization, because the link can be looked up in DBpedia that provides more detailed categorization.

The confidence of the linked entity \( \hat{e}_{i,s} \) for a surface form \( s \) is counted as the ratio of the highest co-occurrence score of the winning candidate to the sum of co-occurrence scores of
Figure 5.12: SemiTags – Graphical user interface.

the rest of candidates for given surface form (Equation 5.26).

\[
\text{confidence}_s = \frac{\max \{ \text{score}_{c_s} \}}{\sum_{c_s \in C_s} \text{score}_{c_s}}
\]  

(5.26)

5.8.1 SemiTags architecture

In Figure 5.14, we show a simplified architecture of our tool. Third party Satnford named entity recognizer is marked with grey color. We wrapped it in our own component that provides the recognition as a RESTful service. Therefore it can be easily replaced with a component providing some other type of named entity recognition. It is completely independent on the rest of the system.

The recognition service is used by the disambiguation service. The disambiguation service reads recognized surface forms and assigns concrete identifiers to them (see Sections 5.6.3, 5.7.1 and 5.7.3 for description of different disambiguation strategies that are used).

The disambiguation service provides again RESTful interface, so any client can connect to it and use it automatically. We provide a basic web user interface for the REST service. Our automatic evaluators also connect to the REST service.
Figure 5.13: Example of SemiTags web service JSON response.

```json
[
  // ...
  {
    "score": 1,
    "start": "42",
    "link": "Mahmoud_Abbas",
    "type": "PERSON",
    "name": "Mahmoud Abbas"
  },
  {
    "score": 0.9574468085106383,
    "start": "63",
    "link": "Qatar",
    "type": "LOCATION",
    "name": "Qatar"
  }
  // ...
]
```

Figure 5.14: SemiTags architecture.
Chapter 6
Evaluation

In this chapter we present the results of our examination of Wikipedia data dumps. We evaluate our disambiguation approaches on three types of benchmarks stressing different types of disambiguation scenarios. All the experiments were gathered using dumps capturing Wikipedia snapshots from 4/6/2013.

Testing server configuration:

- CPU: Intel(R) Xeon(R) CPU E5520 @ 2.27GHz
- Memory: 56GB RAM
- Storage: NAS 6.8TB

6.1 Wikipedia Statistics

Our disambiguation algorithms (and many others – see [109, 107, 111] for examples) use Wikipedia as the source knowledge base and exploit structure of links in Wikipedia articles. So we thought a good idea is to take a look at the Wikipedia dataset itself and count some basic statistics about the data distribution.

Figure 6.1 shows the distribution of counts of entities per paragraph. Y-axis (Equation 6.2) denotes how many paragraphs contain given count of entities (on X-axis – Equation 6.1). It shows a classical long tail distribution. 95% of paragraphs contain up to 10 entities. The maximum count of entities per paragraph that we discovered is 7380\(^1\). It turned out to be a Wikiproject Estonia watchlist – a service page that we filtered out during our cleaning steps (see Section 5.4.3).

\[
n_p = |\{e: \exists s, (e, p, s) \in W \setminus \{e\}\}| \quad (6.1)
\]

\[
y = |\{p: x = n_p\}| \quad (6.2)
\]

\(^1\)http://en.wikipedia.org/wiki/Wikipedia:WikiProject_Estonia/publicwatchlist

79
First ten results are shown in Table 6.1. We can see that about 72% of total count of paragraphs containing at least one link to an entity can be used for co-occurrence disambiguation.

Table 6.1: Count of entities per paragraph – from one to ten entities per paragraph.

<table>
<thead>
<tr>
<th>Count of Entities</th>
<th>Count of Paragraphs</th>
<th>Total Paragraphs</th>
<th>Count Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,690,349</td>
<td>16,191,123</td>
<td>28%</td>
</tr>
<tr>
<td>2</td>
<td>3,377,214</td>
<td></td>
<td>20%</td>
</tr>
<tr>
<td>3</td>
<td>2,589,410</td>
<td></td>
<td>15%</td>
</tr>
<tr>
<td>4</td>
<td>1,861,603</td>
<td></td>
<td>11%</td>
</tr>
<tr>
<td>5</td>
<td>1,273,312</td>
<td></td>
<td>7%</td>
</tr>
<tr>
<td>6</td>
<td>875,813</td>
<td></td>
<td>5%</td>
</tr>
<tr>
<td>7</td>
<td>602,902</td>
<td></td>
<td>4%</td>
</tr>
<tr>
<td>8</td>
<td>417,517</td>
<td></td>
<td>2%</td>
</tr>
<tr>
<td>9</td>
<td>293,882</td>
<td></td>
<td>2%</td>
</tr>
<tr>
<td>10</td>
<td>209,121</td>
<td></td>
<td>1%</td>
</tr>
<tr>
<td>Sum</td>
<td>16,191,123</td>
<td></td>
<td>95%</td>
</tr>
</tbody>
</table>

In a similar way we explored the distribution of entities per surface forms. This is even more important distribution for disambiguation purposes, because it shows how ambiguous actually are surface forms in Wikipedia (Equation 6.3, see Equation 5.7 for definition of \( n_s \)). The distribution (Figure 6.2) is even less uniform and was a bit surprising for us. It showed that about 92% of all 4.4 million examined surface forms was not ambiguous and linked to only one entity.

\[
y = |\{s : x = n_s\}|
\]

(6.3)

It should be noted that we examined the surface forms in their exact form that was used in Wikipedia. In other words the surface forms were compared for exact string match when
6.1. WIKIPEDIA STATISTICS

Figure 6.2: Count of entities per surface form (Equation 6.3) – default exact match of surface forms. Graph is constructed according to Equation 6.3.

deciding about distinct surface forms. To see, how big influence did this fact had on our evaluation, we walked through the whole dataset once again and stored normalized forms of the surface forms. We omitted special characters, reduced multiple spaces, lowered the cases and stripped diacritics. With this normalized dataset, we measured the distribution once again.

The results of the evaluation shown in Figure 6.3 are not much different from the previous state. The overall count of distinct surface forms decreased from 4,339,023 to 3,734,848. But the proportion of surface forms with only a single meaning decreased by less than 1% from 92.15% to 91.44%.

Figure 6.3: Count of entities per surface form (Equation 6.3) – normalized surface forms. Graph is constructed according to Equation 6.3.

We summarized first five results of the normalized surface forms set in Table 6.2. We can see that about 99% of surface forms have up to 5 possible candidates.

In Figure 6.4 we show the comparison of the distribution of counts of entities per normalized surface form and per exact matched surface form. As we can see the distribution is pretty similar.

Just for the illustration one of the most ambiguous surface forms that we discovered was ”2006” with 998 candidates. On the other hand the huge proportion of very specific surface forms can be illustrated by the screen shot on Figure 6.5. It shows an article about a street in Ottawa. ASCII art is used in this case to draw the map of the street, which
Table 6.2: Count of entities per normalized surface form – from one to five entities per surface form.

<table>
<thead>
<tr>
<th>Count of Entities</th>
<th>Count of Surface Forms</th>
<th>Total Surface Forms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,415,005</td>
<td>91.44%</td>
</tr>
<tr>
<td>2</td>
<td>193,773</td>
<td>5.19%</td>
</tr>
<tr>
<td>3</td>
<td>52,797</td>
<td>1.41%</td>
</tr>
<tr>
<td>4</td>
<td>23,047</td>
<td>0.62%</td>
</tr>
<tr>
<td>5</td>
<td>12,735</td>
<td>0.34%</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>3,697,357</strong></td>
<td><strong>100%</strong></td>
</tr>
</tbody>
</table>

contains links to description of particular buildings. However the links are not labeled with the names of the buildings, but with equal signs. Different count of equal signs corresponds to different size of buildings and is completely irrelevant to the link target.

By this example, we want to illustrate the creativity of authors of Wikipedia texts and the sort of pitfalls that need to be solved during the data cleaning phase when someone wants to deal with Wikipedia datasets.

Finally we evaluated also how many possible names can have one entity (Equation 6.4). Figure 6.6 shows the distribution (Equation 6.5). It is similar to previous cases. More than 70% of entities have up to 3 possible names. The article with most variants of its name according to link structure is the article describing topic *Case citation*. Many concrete case citations link to this general topic. That is why it has 1663 different surface forms.

$$l_e = |\{s : \exists p, (e, p, s) \in M\}|$$  

$$y = |\{e : x = l_e\}|$$

First ten results are summarized in Table 6.3. Again entities with up to 10 surface forms build the vast majority of 93.65% of entities in Wikipedia.

When disambiguating an entity we generate candidates based on their surface form (see Section 5.1). In Figure 6.7 we can see how is the situation by surface forms that have two or more candidates. On the x-axis, we see in how many percent of occurrences the surface form appeared in the sense of the most popular candidate. In most cases the first candidate takes 50% of occurrences. Higher percentage corresponds to a surface form with one very popular candidate and then several very rare ones. Lower percentage corresponds to surface forms with many different rarely covered candidates. The histogram is assembled according to Equation 6.6.

$$y = \left\{ s : \frac{n^s_{e_1}}{\sum_{i=2} n^s_{e_i}} = x \right\}$$
Interesting fact is that by 97.84% of all surface forms the most popular candidate is a correct guess with more than 50% probability. We evaluated the method taking the most popular meaning of an entity in later experiments.

Also note the difference between histograms shown in Figure 6.2 and Figure 6.3 where we count just the number of possible candidates. Here in Figure 6.7 we count the popularity of a candidate mentioned under the given surface form.

### 6.2 Various Context Representations for Disambiguation

In this section we show evaluation results of algorithms described in Section 5.6. Because of the computational complexity of the algorithms (especially the sentence structure representation), we tested them on a smaller dataset assembled manually by a human annotator.

#### 6.2.1 Testing Dataset Description

For the evaluation purposes, we randomly selected nine news articles from New York Times and Wall Street Journal. We manually annotated these articles. The annotation resulted in 106 distinct entities.
CHAPTER 6. EVALUATION

Figure 6.5: Illustration of inadequate usage of links in Wikipedia.

(a) Normal scale  (b) Log scale

Figure 6.6: Surface forms per entity. Graph is constructed according to Equation 6.5.

6.2.2 Comparison of Various Context Representations

On the dataset consisting of nine news articles with 106 entities, we evaluated all three described methods of named entity disambiguation.

Figure 6.8 shows the comparison of all three described disambiguation methods in terms of precision. We can see that the structural based measure analysing co-occurrences of entities in same paragraphs outperforms both the other textual based measures, achieving the precision of 91.9 %.

The comparison of recall of all the three approaches is presented in Figure 6.9. Here in turn the bag of words approach outperforms the structural based model. However the difference is very small – about 1 %, which is in absolute numbers only one more correctly disambiguated entity.

The model exploiting a structure of sentences did not work well in both cases. This might be partially due to the simplification of identification of subject, verb, object triples or because of sometimes inappropriate results of chunking. However, an interesting fact
that we observed is that the mistakes of the sentence structure model and the bag of words representation are somehow correlated. In most cases, when the bag of words approach disambiguated an entity incorrectly, even the sentence based approach failed. But the sentence based approached failed more often.

We assume that these two approaches are in the final effect quite similar. By recognizing subject, verb and object, we try to identify semantically most important words in a sentence. Whereas the bag of words approach with tf-icf term weights aims to achieve the same by weighting potentially disambiguating terms higher.

Another effect that we observed was that often very general verbs were identified (such as is, was, has etc.). While they can appear in connection with about any word they contribute very little to correct results.

On the other hand the bag of words model and the structural measure taking into account co-occurrences of individual entities are complementing each other. It might be interesting to combine those two approaches. However, we leave this for future work.

6.3 Multilingual support

The structural representation for named entity disambiguation is to some extent language independent. As long as there is a local version of Wikipedia in the desired language, we can represent relationships between entities as their co-occurrences. To demonstrate and evaluate this idea, we created a web service called SemiTags.

6.3.1 Named Entity Recognition in Non-English Texts

For English and German there are available already learned models for named entity recognition using Stanford Named Entity Recognizer [60]. For Dutch we haven’t found such a
Figure 6.7: Candidates percentage histogram counted according to Equation 6.6.

Figure 6.8: Comparison of precision of all three described methods on manually assembled dataset based on news articles from New York Times and Wall Street Journal. Used methods: Bag of Words (Section 5.6.1), Structural Measure (Section 5.6.3), Sentence Structure (Section 5.6.2).

ready-made model. Therefore we had to create one our selves. We were considering two state of the art tools: Stanford Named Entity Recognizer and OpenNLP [10] (Apache open source library supporting various natural language processing tasks including named entity recognition). With both tools it is possible to train models on an arbitrary dataset.

For training, we used the dataset provided for CONLL-2003 shared task [157], which could be used for both tools (after conversion to an appropriate format).

For the comparison, we used 10 manually annotated articles collected from the Dutch TV Show Tussen Kunst & Kitsch web page. Totally we identified 131 named entities in these texts.

Both tools were trained using the same CONLL-2003 datasets. In Figure 6.10, we show results (overall precision and recall) of entity identification in provided texts – in other words the ability of the tool to determine the exact position of the named entity.
Figure 6.9: Comparison of recall of all three described methods on manually assembled dataset based on news articles from New York Times and Wall Street Journal. Used methods: Bag of Words (Section 5.6.1), Structural Measure (Section 5.6.3), Sentence Structure (Section 5.6.2).

Figure 6.10: Precision and recall of entity identification in Dutch Texts using Stanford Named Entity Recognizer (Stanford) and OpenNLP library (ONLP).

Figure 6.11 shows the precision and recall of type determination in both tools. Note that this is the overall precision and recall of the identification and type determination. Thus, when an entity is not identified, it also results in an error in the type determination. Therefore, Figure 6.11 shows the results of the whole recognition process rather than just the type determination.

While results of the entity identification are acceptable, the performance of type determination is relatively poor. However, type determination will be validated in the next step of our disambiguation process using data found in our knowledge base indexed from Wikipedia. The Stanford Named Entity Recognizer outperforms the results of the OpenNLP library significantly. For further experiments we chose the tool provided by Stanford.

In our next experiment, we tested the Stanford Named Entity Recognizer trained on German texts with another manually annotated dataset of German articles. For testing the
CHAPTER 6. EVALUATION

Performance in German, we randomly collected 10 articles from RBB Online web page. In German articles, we identified 121 entities. In Figure 6.12, precision and recall of entity identification in German and Dutch texts is shown.

Finally, Figure 6.13 shows precision and recall of the type determination of identified entities again in German and Dutch texts. The named entity recognition and type disambiguation processes have better results for German texts.

It is necessary to note that Tussen Kunst & Kitsch Web page in Dutch represents a much more difficult domain than German RBB News articles. The training data sets are focused on news domain. Therefore the recognition in this domain provides better results. Texts on Tussen Kunst & Kitsch Web page are also often partially structured (e.g. contain lists) and often do not contain whole sentences. For the recognizer, it is then difficult to determine boundaries of extracted named entities.

http://www.rbb-online.de
6.3. MULTILINGUAL SUPPORT

Figure 6.13: Precision and recall of type determination – Named entity recognition for Dutch and German.

6.3.2 German Disambiguation

Apart from evaluation of state of the art named entity recognition tools provided in Section 6.3.1, we evaluated also the whole recognition and disambiguation process using SemiTags on German texts. The testing set was composed of 10 articles collected from different categories of German RBB Online news web page. The articles contained totally 126 named entities that were manually annotated. As a knowledge base we used the German Wikipedia. In Figure 6.14, we show precision and recall of the results provided by SemiTags.

The precision of the disambiguation is remarkably high – more than 95%. The co-occurrence based disambiguation trades precision for recall and behaves rather conservatively. Given entity is rather not disambiguated at all, which does not mean that it is not present in the backing knowledge base (Wikipedia). It means that it has no connection to other entities identified in the same text.

Figure 6.14: Precision and recall of German disambiguation.
6.4 TAC EntityLinking Task Dataset

For more thorough evaluation, we have chosen the dataset of Entity Linking Task from Text Analysis Conference (TAC) [154]. The dataset is composed of documents from three types of sources: news, web pages and discussion fora.

Together with the LinkedTV team, we took part in the TAC2013 Entity Linking Task [13]. For the track evaluation we used old indexes with non cleaned results which lead to poorer results. So we decided to explore the impact of index cleaning in more detail. Table 6.4 compares the results of structural disambiguation evaluated in the previous Section 6.2.2 with baseline method taking into account only the most popular meaning of a surface form. However the baseline method is applied already on cleaned datasets (see Section 5.4.3 for the description of cleaning steps that we performed).

The most popular meaning method is evaluated in two variants. The first one uses the default surface form index with exact surface form match. The second one uses the index with normalized surface forms.

<table>
<thead>
<tr>
<th>Disambiguation</th>
<th>B³ P</th>
<th>B³ R</th>
<th>B³ F1</th>
<th>B³⁺ P</th>
<th>B³⁺ R</th>
<th>B³⁺ F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>structural.noncleaned</td>
<td>0.912</td>
<td>0.428</td>
<td>0.582</td>
<td>0.558</td>
<td>0.292</td>
<td>0.383</td>
</tr>
<tr>
<td>default.cleaned</td>
<td>0.952</td>
<td>0.429</td>
<td>0.592</td>
<td>0.637</td>
<td>0.318</td>
<td>0.424</td>
</tr>
<tr>
<td>normalized.cleaned</td>
<td>0.952</td>
<td>0.480</td>
<td>0.638</td>
<td>0.681</td>
<td>0.377</td>
<td>0.485</td>
</tr>
</tbody>
</table>

By Entity Linking task B-Cubed measure [9] usually used to evaluate quality of clustering is used in its enhanced form of B-Cubed+ measure that takes into account also correctness of identifiers assigned to linked entities (see Section 3.1.5 for explanation).

6.4.0.1 Observation

We thoroughly examined a sample of 100 queries from the TAC dataset to find out what is the most common cause of an error in disambiguation using the most frequent sense method on normalized datasets. Figure 6.15 shows the results.

The most common source of errors – almost 38% of overall error count – is caused by a missing record in the knowledge base. Very often this is not due to knowledge base incompleteness but rather due to a misspelled name of the surface form or in another type of rare naming that did not appear in Wikipedia texts and therefore is not in our surface form index for generating candidates. This could be improved by index expansion with generated most common misspellings and further more extensive lemmatisation, stemming, synonymy generation and other query expansion information retrieval techniques (see Section 3.1.4). The second most common source of errors was wrong disambiguation where an incorrect link was assigned, because the particular surface form simply was not used in its most common meaning (n²₁ > n²₂ and ñ = e₂). These results can be further improved taking into account the context of the surface form. We will cover more on this topic later.
6.5. CONSTRUCTION OF HIGHLY AMBIGUOUS BENCHMARKS

(a) Overall results

(b) Errors only

Figure 6.15: Most frequent sense disambiguation results manually categorized on a sample of 100 queries.

Another group of errors – false positives – is connected also to the disambiguation process. The problem occurs in situations when in the text appears a shortcut or a name which is the same as some name of an entity in the knowledge base, but it is just a coincidence and the particular surface form should not get any identifier. Often this happens by normal people (who have no records on Wikipedia) that have same name as someone famous. This problem is quite difficult to solve. Some clue can be taken from a low context score.

The rest couple of errors was caused by an inconsistency between the reference knowledge base and the disambiguation knowledge base.

We were quite surprised by the fact that even a basic most frequent sense method did not have that poor results. It correctly classified 71% of the examined queries. Needless to say that some of the queries (especially those coming from discussion fora) are really tricky.

6.5 Construction of Highly Ambiguous Benchmarks

Based on our previous experiments we decided to construct our own benchmark based on Wikipedia that would extract only the tricky cases of non popular meanings of entities. Formally we define \( n^s_e \) as the number of mentions when the surface form \( s \) appeared in the meaning of entity \( e \). Then \( n^s \) is the total number of surface form mentions under any meaning (Equation 6.7).

\[
    n^s = \sum_{e \in E} n^s_e
\]

(6.7)

We also assume the ordering of candidate meanings (entities) of a concrete surface form \( s \) defined in Equation 5.9. Then we define an ordering of surface forms. Surface form \( s \) is
more suitable than $r$ for inclusion in the benchmark, when Equation 6.8 holds.

$$1 \leq \frac{n_{e_1}}{n_{e_2}} \leq \frac{n_{e_1}^r}{n_{e_2}^r} \quad (6.8)$$

So we select only surface forms that have two first candidates with the same or very similar relevance for the surface form (in terms of the most common meaning). With these surface forms, we walk through indexed paragraphs and extract the ones that mention the first or the second candidate entity. These paragraphs are after basic data cleanup included in the benchmark. Each paragraph together with the surface form and link to the entity representing the correct meaning of the surface form build a single query. This way we composed the Soft benchmark that contains 226 queries.

Additionally we built even more tricky benchmark that contains only the third most popular sense $e_3^s$ for each surface form $s$ according to Equation 5.9. We call this benchmark 3dcandidate benchmark and it contains 628 queries.

When building the indexes the benchmark paragraphs were left apart in order to keep the testing set independent of the training set.

### 6.6 Co-Occurrence Based Algorithms Evaluation

In this section we show evaluation results for our algorithms described in Chapter 5. As a baseline we use the default most frequent sense disambiguation. We also compare our results with one of the most popular disambiguation tools DBpedia Spotlight [107]. It has publicly available REST service, so we wrote an evaluator that connects to this web service and evaluates its result on testing datasets.

Please note that this evaluation does not provide official DBpedia Spotlight benchmark results. We include Spotlight just as a reference of a representative general purpose disambiguation tool in order to better compare several levels of context awareness in our disambiguation algorithms.

We use three datasets described in previous sections.

#### 6.6.1 TAC Evaluation – context aware approaches

Table 6.5 shows results of the sum and max co-occurrence disambiguation together with Spotlight results. We included also the baseline default and normalized most frequent sense disambiguation from previous runs.

We achieved the best results with the maximum co-occurrence method. Interesting is that the normalized most frequent sense was the second best method in this case. This corresponds also to the analysis summarized in Figure 6.15, which shows that the disambiguation errors do not build that big portion of errors that one would expect.

Normalized most frequent sense is even better than Spotlight but it is necessary to note that our service enables explicit queries for entities. So even if Stanford NER does not
6.6. **CO-OCCURRENCE BASED ALGORITHMS EVALUATION**

Table 6.5: TAC Dataset evaluation – default, normalized (Section 5.7.1, Algorithm 7) co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9) and Spotlight disambiguation (Section 3.4.2).

<table>
<thead>
<tr>
<th>Disambiguation</th>
<th>$B^3 P$</th>
<th>$B^3 R$</th>
<th>$B^3 F1$</th>
<th>$B^{3+} P$</th>
<th>$B^{3+} R$</th>
<th>$B^{3+} F1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>0.952</td>
<td>0.429</td>
<td>0.592</td>
<td>0.637</td>
<td>0.318</td>
<td>0.424</td>
</tr>
<tr>
<td>normalized</td>
<td>0.952</td>
<td>0.480</td>
<td>0.638</td>
<td>0.681</td>
<td>0.377</td>
<td>0.485</td>
</tr>
<tr>
<td>sum</td>
<td>0.935</td>
<td>0.461</td>
<td>0.618</td>
<td>0.636</td>
<td>0.347</td>
<td>0.449</td>
</tr>
<tr>
<td>max</td>
<td>0.953</td>
<td><strong>0.487</strong></td>
<td><strong>0.644</strong></td>
<td><strong>0.685</strong></td>
<td><strong>0.388</strong></td>
<td><strong>0.495</strong></td>
</tr>
<tr>
<td>spotlight</td>
<td><strong>0.976</strong></td>
<td>0.420</td>
<td>0.587</td>
<td>0.630</td>
<td>0.307</td>
<td>0.413</td>
</tr>
</tbody>
</table>

recognize the entity, we can still post the surface form to our service and it is disambiguated in the context of other recognized entities.

Spotlight does not support this feature. So it is handicapped. In order to avoid this handicap for the next run of evaluation we chose only those queries, that were correctly recognized by Spotlight (so we measured only the disambiguation capabilities). The results are summarized in Table 6.6.

Now Spotlight outperformed our approaches. The second best option was again normalized most frequent sense. However in this case the advantage is on the side of Spotlight, because we select only entities that are disambiguated by Spotlight we favour its knowledge base against our.

Table 6.6: TAC Dataset evaluation – default, normalized (Section 5.7.1, Algorithm 7) co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9) and Spotlight disambiguation (Section 3.4.2). Selected queries recognized by Spotlight.

<table>
<thead>
<tr>
<th>Disambiguation</th>
<th>$B^3 P$</th>
<th>$B^3 R$</th>
<th>$B^3 F1$</th>
<th>$B^{3+} P$</th>
<th>$B^{3+} R$</th>
<th>$B^{3+} F1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>0.942</td>
<td>0.751</td>
<td>0.835</td>
<td>0.629</td>
<td>0.539</td>
<td>0.581</td>
</tr>
<tr>
<td>normalized</td>
<td>0.939</td>
<td>0.826</td>
<td>0.879</td>
<td>0.697</td>
<td>0.640</td>
<td><strong>0.668</strong></td>
</tr>
<tr>
<td>sum</td>
<td>0.924</td>
<td>0.795</td>
<td>0.855</td>
<td>0.611</td>
<td>0.560</td>
<td>0.585</td>
</tr>
<tr>
<td>max</td>
<td>0.937</td>
<td>0.841</td>
<td>0.887</td>
<td>0.687</td>
<td>0.644</td>
<td>0.665</td>
</tr>
<tr>
<td>spotlight</td>
<td>0.918</td>
<td><strong>0.934</strong></td>
<td><strong>0.926</strong></td>
<td><strong>0.729</strong></td>
<td><strong>0.724</strong></td>
<td><strong>0.727</strong></td>
</tr>
</tbody>
</table>

In order to avoid the bias, our third run (Table 6.7) included only queries recognized by both the tools – Spotlight and our Normalized most frequent sense (that means that we have also candidates for the selected entities in our knowledge base). Spotlight outperformed our methods but the difference between Spotlight and normalized most frequent sense was not very big.

Finally we ran the evaluation only with the queries that were not null in the TAC dataset (results in Table 6.8). That means only queries that had links assigned in the original golden standard. Here again maximum co-occurrence measure outperformed the others. It seems that by keeping only queries that contained surface forms recognized by both systems we omitted the more tricky entities that are probably better disambiguated.
Table 6.7: TAC Dataset evaluation – default, normalized (Section 5.7.1, Algorithm 7) co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9) and Spotlight disambiguation (Section 3.4.2). Selected queries linked by both Spotlight and normalized most frequent sense algorithm.

<table>
<thead>
<tr>
<th>Disambiguation</th>
<th>B^3 P</th>
<th>B^3 R</th>
<th>B^3 F1</th>
<th>B^3+ P</th>
<th>B^3+ R</th>
<th>B^3+ F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>0.924</td>
<td>0.840</td>
<td>0.880</td>
<td>0.713</td>
<td>0.657</td>
<td>0.684</td>
</tr>
<tr>
<td>normalized</td>
<td>0.920</td>
<td>0.961</td>
<td>0.940</td>
<td>0.815</td>
<td>0.819</td>
<td><strong>0.817</strong></td>
</tr>
<tr>
<td>sum</td>
<td>0.916</td>
<td>0.880</td>
<td>0.897</td>
<td>0.692</td>
<td>0.676</td>
<td>0.684</td>
</tr>
<tr>
<td>max</td>
<td>0.933</td>
<td>0.932</td>
<td>0.933</td>
<td>0.787</td>
<td>0.780</td>
<td>0.783</td>
</tr>
<tr>
<td>spotlight</td>
<td><strong>0.940</strong></td>
<td><strong>0.978</strong></td>
<td><strong>0.958</strong></td>
<td><strong>0.823</strong></td>
<td><strong>0.832</strong></td>
<td><strong>0.828</strong></td>
</tr>
</tbody>
</table>

taking context into account (as in the case of max co-occurrence disambiguation contrary to normalized most frequent sense).

Table 6.8: TAC Dataset evaluation – default, normalized (Section 5.7.1, Algorithm 7) co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9) and Spotlight disambiguation (Section 3.4.2). Only queries with links in the golden standard.

<table>
<thead>
<tr>
<th>Disambiguation</th>
<th>B^3 P</th>
<th>B^3 R</th>
<th>B^3 F1</th>
<th>B^3+ P</th>
<th>B^3+ R</th>
<th>B^3+ F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>0.979</td>
<td>0.473</td>
<td>0.638</td>
<td>0.451</td>
<td>0.313</td>
<td>0.369</td>
</tr>
<tr>
<td>normalized</td>
<td>0.979</td>
<td>0.563</td>
<td>0.715</td>
<td><strong>0.536</strong></td>
<td>0.422</td>
<td>0.472</td>
</tr>
<tr>
<td>sum</td>
<td>0.938</td>
<td>0.531</td>
<td>0.678</td>
<td>0.463</td>
<td>0.369</td>
<td>0.411</td>
</tr>
<tr>
<td>max</td>
<td>0.960</td>
<td><strong>0.577</strong></td>
<td><strong>0.721</strong></td>
<td><strong>0.536</strong></td>
<td><strong>0.441</strong></td>
<td><strong>0.484</strong></td>
</tr>
<tr>
<td>spotlight</td>
<td><strong>0.990</strong></td>
<td>0.457</td>
<td>0.625</td>
<td>0.408</td>
<td>0.283</td>
<td>0.334</td>
</tr>
</tbody>
</table>

### 6.6.2 Highly Ambiguous Benchmarks Evaluation

With our highly ambiguous benchmarks we were first evaluating the disambiguation on the soft benchmark. Soft benchmark contains only ambiguous surface forms that have at least two candidates. These candidates are very close to each other in terms of most frequent sense. So it contains surface forms like Z1, which can be the name of BMW Z1 (two-seat roadster) or Z1 computer. Both possibilities are according to Wikipedia mentions equally probable. The benchmark then contains mentions of these entities in any of their sense.

In Table 6.9 we show results of the first run of evaluation. We can see that the performance of default and normalized most frequent sense disambiguation are relatively poor corresponding to the fact that some times it hits the correct first candidate in the benchmark by chance. Also the fact that the default (exact match) version performs better than normalized is not surprising. Probably some of the rare candidates have usually slightly different names that were merged with the more general ones by normalization process. On the other hand the improvement by disambiguation approaches taking into account the context is significant. Maximum works better than sum in this case.
This run was again limited to queries that were recognized by Spotlight, too. In order to grant equal chances to Spotlight. Nevertheless, the performance of Spotlight is poor on this benchmark. Comparable to our most frequent sense disambiguation. It seems that in case of Spotlight the popularity plays a significant role in disambiguation. It is interesting to compare it to the result on a real world benchmark such as TAC (Table 6.7) where it gained the highest score very close to the score of our normalized most frequent sense disambiguation.

<table>
<thead>
<tr>
<th>disambiguation</th>
<th>$B^3_P$</th>
<th>$B^3_R$</th>
<th>$B^3_F1$</th>
<th>$B^{3+}_P$</th>
<th>$B^{3+}_R$</th>
<th>$B^{3+}_F1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>0.705</td>
<td>0.955</td>
<td>0.811</td>
<td>0.227</td>
<td>0.352</td>
<td>0.276</td>
</tr>
<tr>
<td>normalized</td>
<td>0.682</td>
<td>0.955</td>
<td>0.795</td>
<td>0.205</td>
<td>0.295</td>
<td>0.242</td>
</tr>
<tr>
<td>sum</td>
<td>0.886</td>
<td>0.955</td>
<td>0.919</td>
<td>0.409</td>
<td>0.432</td>
<td>0.420</td>
</tr>
<tr>
<td>max</td>
<td>0.886</td>
<td>0.977</td>
<td>0.930</td>
<td>0.489</td>
<td>0.523</td>
<td>0.505</td>
</tr>
<tr>
<td>spotlight</td>
<td>0.682</td>
<td>1.000</td>
<td>0.811</td>
<td>0.227</td>
<td>0.295</td>
<td>0.257</td>
</tr>
</tbody>
</table>

Our final evaluation was run using our 3d Candidate Wikipedia benchmark. The results are summarized in Table 6.10. We omitted the most frequent sense methods in this case, because they would get the 0 score anyway. Again we included only queries recognized and disambiguated by Spotlight. The best score was achieved again with maximum co-occurrences scoring. Similarly to the soft benchmark evaluation Spotlight performed worse in this scenario with more obscure entities.

<table>
<thead>
<tr>
<th>disambiguation</th>
<th>$B^3_P$</th>
<th>$B^3_R$</th>
<th>$B^3_F1$</th>
<th>$B^{3+}_P$</th>
<th>$B^{3+}_R$</th>
<th>$B^{3+}_F1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>0.994</td>
<td>1.000</td>
<td>0.997</td>
<td>0.274</td>
<td>0.274</td>
<td>0.274</td>
</tr>
<tr>
<td>max</td>
<td>0.994</td>
<td>1.000</td>
<td>0.997</td>
<td>0.282</td>
<td>0.283</td>
<td>0.282</td>
</tr>
<tr>
<td>spotlight</td>
<td>0.988</td>
<td>1.000</td>
<td>0.994</td>
<td>0.105</td>
<td>0.106</td>
<td>0.105</td>
</tr>
</tbody>
</table>

### 6.7 Summary

Our Wikipedia statistics and the evaluation on the TAC dataset show that there are two main factors influencing the disambiguation correctness:

- Context of the disambiguated entity – When a surface form is ambiguous and it is not used in its most frequent sense, the only clue that we have to guess the correct
CHAPTER 6. EVALUATION

type is its context. Therefore it is crucial to choose correct context representation. We evaluated several different approaches to the context representation. The most promising seems to be the structural context representation exploiting the structure of links in Wikipedia. More specifically the best performance was achieved by selecting candidates having the maximum co-occurrence score.

- Preparation of the lookup index – The first step before the actual disambiguation is candidate generation. The success of the subsequent disambiguation depends completely on the result of the candidate generation. Therefore very important is the way, how we map surface forms to their candidates. Figure 6.15 clearly shows the impact of incomplete or to constrained candidate lookup index. We implemented basic normalization of surface forms in order to increase the recall. However, in the future results could be further improved by more extensive query expansion when querying surface form index.

Usually by disambiguation approaches the stress is given on the proper interpretation of the context and its balancing with general popularity of the entity. Our experiments show that the importance of general popularity of entity is bigger than we expected. Similar results were observed in [150].

Especially the second factor – proper preparation of the lookup index – is often overlooked when talking about disambiguation algorithms. However, for real world texts written by human, it is essential to deal with factors such as misspellings, incorrect names or abbreviations. In fact focus on this part of the disambiguation process – the candidate generation – could have even bigger impact on proper entity linking than tuning of disambiguation approaches.

In Table 6.11 we summarize the results measured as B³⁺ F1 score of all the evaluation runs with different subsets of the TAC dataset.

<table>
<thead>
<tr>
<th>Disambiguation</th>
<th>TAC</th>
<th>TAC-Spot</th>
<th>TAC-Spot+MFS</th>
<th>TAC-links</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>0.424</td>
<td>0.581</td>
<td>0.684</td>
<td>0.369</td>
</tr>
<tr>
<td>normalized</td>
<td>0.485</td>
<td>0.668</td>
<td>0.817</td>
<td>0.472</td>
</tr>
<tr>
<td>sum</td>
<td>0.449</td>
<td>0.585</td>
<td>0.684</td>
<td>0.411</td>
</tr>
<tr>
<td>max</td>
<td>0.495</td>
<td>0.665</td>
<td>0.783</td>
<td>0.484</td>
</tr>
<tr>
<td>spotlight</td>
<td>0.413</td>
<td>0.727</td>
<td>0.828</td>
<td>0.334</td>
</tr>
</tbody>
</table>

Table 6.12 summarizes the results of evaluation with our two highly ambiguous benchmarks. Here our co-occurrence based methods clearly outperform the most frequent sense
6.7. SUMMARY

Table 6.12: Comparison of results on both highly ambiguous benchmarks. Used methods: default, normalized (Section 5.7.1, Algorithm 7) co-occurrence sum (Section 5.7.2, Algorithm 8), max (Section 5.7.3, Algorithm 9).

<table>
<thead>
<tr>
<th>Disambiguation</th>
<th>Soft</th>
<th>3dcandidate</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>0.276</td>
<td>0</td>
</tr>
<tr>
<td>normalized</td>
<td>0.242</td>
<td>0</td>
</tr>
<tr>
<td>sum</td>
<td>0.420</td>
<td>0.274</td>
</tr>
<tr>
<td>max</td>
<td><strong>0.505</strong></td>
<td><strong>0.282</strong></td>
</tr>
<tr>
<td>spotlight</td>
<td>0.257</td>
<td>0.105</td>
</tr>
</tbody>
</table>

methods and even Spotlight that seems to tend to favour most popular entity meanings, too.

It was not our aim to develop an ultimate system that would combine many existing systems in one big ensemble like the complex system introduced in [80]. Rather we aim at uncover the black boxes and focus on basic patterns and rules in Wikipedia entity linking. We believe, these root findings can contribute to improvements of more complex systems.
Chapter 7

Conclusions

7.1 Summary

Figure 7.1 shows the information filtering system schema from Section 1.2. We covered the topic of structured data sources on the web in Chapter 3. We examined linked data sources that are linked to Wikipedia articles. The connection with Wikipedia is a huge benefit for entity linking because Wikipedia entities can be linked to concrete articles on Wikipedia that build the context. The context can be further used for improved entity disambiguation.

We examined possibilities of Linked Data crawling and storage in Chapter 4. We were
inspired by DBpedia project as one of the most comprehensive data stores in Linked Data
cloud. As a side project we launched Czech branch of DBpedia extracting data from Czech
Wikipedia. The project is currently further maintained as part of the LOD2 European
project at University of Economics in Prague and serves Czech Linked Data community.
From our experiments with general Linked Data Crawling we concluded that generally
crawled information is not very useful for our use case as it is too sparse in topics and
entities that are relevant to our use case.

Thus we decided to use only selected data sources that we can clean for our purposes.
We turned back to Wikipedia for two reasons:

- It can be easily linked to DBpedia, which in turn can be linked to many others Linked
  Data resources.
- Entities are bound to their context in terms of surrounding text and link structure
  over the whole Wikipedia.

We further provide important insides in Wikipedia link structure in Chapter 6 that
helps in understanding the ambiguity of named entities surface forms and extent of links
available in Wikipedia. We used these findings to construct two own benchmarks testing
specifically context-aware disambiguation approaches.

In chapter 5 we proposed several approaches to entity linking that can be used for
information enrichment. We focused on context aware techniques. We evaluated the
impact of context ignorance on the results of entity linking in Chapter 6. Our experiments
evaluate different ways of context representation that show consequences of exploitation of
links structure in Wikipedia.

7.2 Contributions of the Thesis

Below, we list the main contributions of this thesis:

1. We proposed a framework for on demand crawling of Linked Data. Using this frame-
work we performed basic evaluation of its scalability and identified some bottlenecks
of many Linked Data resources lying in restrictive crawling delays preventing exten-
sive resource crawling – Section 4.2 and Section 4.3.

2. We launched the project of Czech DBpedia – Section 4.1.

3. We proposed context based approaches to named entity disambiguation exploiting
various context representation models (bag of words, linguistic and structural repre-
sentation) – Section 5.6 and Section 5.7.

4. We compared two state of the art frameworks for named entity recognition on non-
English texts – Section 6.3.
7.3. **ONE MORE THING**

5. We constructed a comprehensive dataset based on all English Wikipedia articles for named entity disambiguation. The dataset reflects link structure and named entity co-occurrences in paragraphs from Wikipedia. These co-occurrences are then used for entity linking based on the context of an entity represented as the group of entities co-occurring in the same paragraph – Section 5.4.

6. We evaluated and compared the individual context representation models on this dataset – Section 6.2, Section 6.4 and Section 6.6.

7. We designed a new approach to effectively deal with data obtained from Wikipedia. We work with about $10^8$ records reflecting entity occurrences in Wikipedia. Instead of dealing with up to $10^{16}$ combinations of records, we reduced the workload dramatically using sorted datasets and in-memory computation – Section 5.4.6 and Section 5.7.

8. We provided deep insides in the structure of links between Wikipedia articles and evaluated link usage patterns – Section 6.1.

9. Publicly available benchmarks testing named entity linking often contain records that do not need very sophisticated disambiguation approaches. Most frequent sense of an entity is often a correct guess. We compiled two new named entity linking benchmarks based on data extracted from Wikipedia that test the ability to disambiguate rather rare meanings of entities, where the real context has to be taken into account – Section 6.5.

### 7.3 One More Thing

The proposal of an information filtering system was covered in Chapter 2 as is noted in Figure 7.1. Apart from this as part of this research the master thesis supervised by the author of this doctoral thesis and developed by David Esner [55] focused on this topic and turned it to a practical application. The thesis implements the framework for news articles filtering based on contained named entities and their relations in DBpedia. Figure 7.2 shows the interface of the application that lists news articles and searches articles with related topics.

In Figure 7.3 the detail of relatedness classification is shown. The project implemented several approaches to the relatedness scoring. We refer interested reader to the master thesis for more details [55]. Originally the recommendation service was connected to Spotlight. However the news filter was designed as a general framework and another named entity disambiguation systems (e.g. ours developed here) can be plugged into it. We plan to switch to our named entity disambiguation system in the future.
7.4 Future Work

The author of this doctoral thesis suggests to explore the following:

- In terms of data acquisition for consecutive information enrichment it would be interesting to incorporate more resources than Wikipedia and DBpedia. A link discovery framework such as Silk [161] could be used to incorporate additional data sources into the knowledge base.

- According to our evaluation results, better performance of the named entity recognition and disambiguation could be achieved by query expansion techniques applied on the candidate index for surface forms. Including most common typing errors and additional normalization would probably improve the recall of our methods. Also we consider using an information retrieval index such as Lucene [4] for surface form lookup.

- The extent of our indexes could be further increased by crawling external sources and searching for links to Wikipedia. In this way we could obtain also Wikipedia links in the context of informal language. However, this kind of links discovery demands already extensive crawling capacities in order to discover enough links to Wikipedia from the free Web.

- Analogically a benchmark composed of ordinary web resources could be constructed
Figure 7.3: Clever News – detail of the relatedness classification of two articles [55].

in a similar way to our highly ambiguous benchmark construction based on Wikipedia. Again this would need enough crawling capacity.
Bibliography


5. Stephen Harris and Nicholas Gibbins. 3store: Efficient bulk rdf storage.


international conference on Systems documentation, SIGDOC ’86, pages 24–26, New York, NY, USA, 1986. ACM.


Publications of the Author

Relevant Journal Papers (Peer Reviewed)


Relevant Conference Papers (Peer Reviewed)


The paper has been cited in:


The paper has been cited in:


Other Relevant Publications (Peer Reviewed)


The paper has been cited in:

Other Conference Papers (Peer Reviewed)


The paper has been cited in:


The paper has been cited in:


Other Publications


Appendix A

Implementation of Co-Occurrence Disambiguation

In this appendix we show two code snippets illustrating the basic principle of co-occurrence disambiguation using sum and max co-occurrence scores.

A.1 Sum Co-Occurrence Disambiguation

```javascript
/**
 * Method returns list of scored candidates. Used scoring function: Sum.
 * @param entities List of surface forms.
 * @param coocCallback
 */
function coOccurrenceSumScore(entities, coocCallback) {
  var entCandidates = {};

  // Find candidates for all passed surface forms
  async.eachSeries(entities, function(entity, callback) {
    redisClient.select(REDIS_DATABASE_NORMALIZED, function() {
      redisClient.hgetall(normalizer.normalize(entity.name[0]),
        function(err, candidates) {
          if (!candidates) {
            candidates = {};
          }
          entCandidates[entity.name[0]] = {};
          entCandidates[entity.name[0]].candidates = candidates;
          entCandidates[entity.name[0]].entity = entity;
          callback();
        });
    }, function(err) {
      console.log("All candidates for all recognized entities: %j",
        entCandidates);
    });
  }, function() {
    coocCallback();
  });
```
// Init candidates list - co-occurrence score is set to 0
for (var i in entCandidates) {
    for (var j in entCandidates[i].candidates) {
        entCandidates[i].candidates[j] = {
            sfScore: parseInt(entCandidates[i].candidates[j]),
            coocScore: 0
        };
    }
}

var coocMap = {};

edisClientCoOc.select(REDIS_DATABASE_COOC, function() {
    // For all surface forms
    async.eachSeries(Object.keys(entCandidates), function(i, callback1) {
        console.log("Scoring surface form cooccurrences " + i);
        // For all candidates of the surface form i
        async.eachSeries(Object.keys(entCandidates[i].candidates), function(candidate1, callback2) {
            if (entCandidates[i].candidates[candidate1].sfScore > 3) {
                // Find candidates in Redis
                redisClientCoOc.hgetall(candidate1, function(err, coocCand) {
                    // Fill the co-occurrence map
                    if (coocCand) {
                        for (var iCooc in coocCand) {
                            if (coocMap[iCooc]) {
                                coocMap[iCooc].cnt += parseInt(coocCand[iCooc]);
                                coocMap[iCooc].sfs[i] = true;
                            } else {
                                coocMap[iCooc] = {
                                    cnt: parseInt(coocCand[iCooc]),
                                    sfs:{}};
                                coocMap[iCooc].sfs[i] = true;
                            }
                        }
                    }
                    callback2();
                }, callback1);
            } else {
                callback2();
            }, callback1);
        }, function (err) {
            // Use the co-occurrence map for scoring candidates
            // For all surface forms
            for (var i in entCandidates) {
            }"
### A.2 Maximum Co-Occurrence Dismabiguation

```javascript
console.log("Scoring surface form " + i);
// For all surface form candidates
for (var candidate1 in entCandidates[i].candidates) {
    // If the candidate is in the co-ocurrence map
    // and co-occured with a candidate
    // for a different surface form,
    // increase the candidate score.
    if (coocMap[candidate1] && (!coocMap[candidate1].sfs[i]
        || (Object.keys(coocMap[candidate1].sfs).length > 1))) {
        entCandidates[i].candidates[candidate1].coocScore += coocMap[candidate1].cnt;
    }
}
coocCallback(err, entCandidates);
```

Listing A.1: Sum Co-Occurrence Dismabiguation

```javascript
/**
 * Method returns list of scored candidates. Used scoring function: Max.
 * @param entities List of surface forms.
 * @param coocCallback
 */
function coOccurrenceMaxScore(entities, coocCallback) {
    var entCandidates = {};

    // Find candidates for all passed surface forms
    async.eachSeries(entities, function (entity, callback) {
        redisClient.select(REDIS_DATABASE_NORMALIZED, function () {
            redisClient.hgetall(normalizer.normalize(entity.name[0]), function(err, candidates) {
                if (!candidates) {
                    candidates = {};
                }
                entCandidates[entity.name[0]] = {};  
                entCandidates[entity.name[0]].candidates = candidates;
                entCandidates[entity.name[0]].entity = entity;
                callback();
            });
        });
    });
} 
```
APPENDIX A. IMPLEMENTATION OF CO-OCCURRENCE DISAMBIGUATION

```javascript
}, function (err) {
    // Init the candidates list - the co-occurrence score is set to 0
    for (var i in entCandidates) {
        for (var j in entCandidates[i].candidates) {
            entCandidates[i].candidates[j] = {
                sfScore: parseInt(entCandidates[i].candidates[j])
            };  
            coocScore: 0
        }
    }

    var coocMap = {};

    redisClientCoOc.select(REDIS_DATABASE_COOC, function () {
        // For all surface forms
        async.eachSeries(Object.keys(entCandidates), function (i, callback1) {
            console.log("Scoring surface form cooccurrences " + i);
            // For all candidates of the surface form i
            async.eachSeries(Object.keys(entCandidates[i].candidates), function (candidate1, callback2) {
                // Omit rare candidates
                if (entCandidates[i].candidates[candidate1].sfScore > 3) {
                    // Find candidates in Redis
                    redisClientCoOc.hgetall(candidate1, function (err, coocCand) {
                        if (coocCand) {
                            // Fill the co-occurrence map
                            for (var iCooc in coocCand) {
                                if (!coocMap[iCooc]) coocMap[iCooc] = {
                                    cnt: parseInt(coocCand[iCooc])
                                };  
                                if (coocMap[iCooc][i]) {
                                    if (parseInt(coocMap[iCooc][i].cnt) <
                                        parseInt(coocCand[iCooc])) {
                                        coocMap[iCooc][i].cnt =
                                            parseInt(coocCand[iCooc])
                                    ;
                                    coocMap[iCooc][i].cooc =
                                        candidate1;
                                }
                            } else {
                                coocMap[iCooc][i] = {
                                    cnt: parseInt(coocCand[iCooc])
                                };  
                                cooc: candidate1
                            }
                        }
                    })
                }
            }, callback1)
        }, callback)
    });
```
A.2. MAXIMUM CO-OCCURRENCE DISMABIGUATION

```javascript
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104

callback2();

if (i != j) {
    // If the current score is bigger
    // than the biggest
    // score for the candidate,
    // set new maximum
    if (entCandidates[i].candidates[candidate1].coocScore < coocMap[candidate1][j].cnt) {
        entCandidates[i].candidates[candidate1].coocScore = coocMap[candidate1][j].cnt;
    }
    // Set the score for both
    // co-occurring candidates
    if (entCandidates[j].candidates[coocMap[candidate1][j].cooc].coocScore < coocMap[candidate1][j].cnt) {
        entCandidates[j].candidates[coocMap[candidate1][j].cooc].coocScore = coocMap[candidate1][j].cnt;
    }
}
```

Listing A.2: Maximum Co-Occurrence Disambiguation