The Faculty of Media Engineering and Technology
German University in Cairo
and
The Faculty of Computer Science
Duale Hochschule Baden-Württemberg Stuttgart

An Open Source Question Answering System Trained on Wikipedia Dumps Adapted to a Spoken Dialogue System

Bachelor Thesis

Author: Youmna Heikal
Supervisor: Prof. Dr. David Suendermann

Abstract

In this thesis the open source open domain question answering system Open Ephyra, which was produced by one of the creators of IBM's Watson has been used and adapted to a Spoken Dialogue System (SDS). In the SDS a user can call into the system and ask a certain question. Open Ephyra is responsible for finding an answer and passing it to the component in the SDS that speaks the answer to the user. Open Ephyra retrieves information from the web or from local text corpora. The web has been used to retrieve information for the SDS. On the other hand, Open Ephyra has also been trained on the English Wikipedia Dump as a local text corpus. Open Ephyra has been linked to other components of the SDS through java servlets, where the communication is accomplished via HTTP GET requests.

Contents

Chapter 1

Introduction

1.1 QA and Spoken Dialogue Systems

Question Answering is a discipline in computer science which depends on document retrieval and natural language processing.

Question Answering Systems tend to only provide the information a user is asking for. Documents with possible answers are first retrieved from available knowledge sources. Afterwards, specific and exact answers are extracted.

Answers may be retrieved from different types of knowledge sources, some may be redundant or unstructured, while others may be semi structured or structured. Documents of the world wide web are an example of unstructured data. They can be used to answer questions in any domain. On the other hand, structured local text corpora can be used, but this often limits questions to specific domains. Common question types are:

1. Factoids : Questions that seek concise answers that reference to facts or evidence (e.g. Who discovered gravity?).
2. Definitional : Questions that seek the definition or relevant information on a certain topic (e.g What is Oxygen ?).
3. List : Questions with more than one factoid answer (e.g. Who are members of the European Parliament?).

Annually the Text Retrieval Conference (TREC) which is a series of workshops that focus on different information retrieval research areas, evaluates tasks which include question answering. Participating in the conference requires participants to run evaluations on a specified corpus which falls into a certain topic (e.g. AQUAINT newswire corpus). During the conference the results of all participants are compared and the best results are determined.

One system that depends on question answering is IBM's Watson. In 2011, it competed in the quiz show Jeopardy against two former winners. In Jeopardy, wide varieties of topics are covered by the quiz materials. The contestants are given clues in answer form and they should phrase their response in question form. Watson had four terabytes of disk storage, where it had access to 200 million pages of structured and unstructured data including the whole text of Wikipedia, without having access to the internet. Watson out performed its human opponents and won the first price.

Spoken Dialogue Systems interact with users via speech. The input speech is processed to determine the information the user requested. Afterwards, the system responds by retrieving the correct information from available knowledge bases. The response must be specific, because the user will not be interested to hear unwanted extra information.

Therefore, Spoken Dialogue systems often use Question Answering Systems as their information retrieval or answer extraction component.

1.2 State of the Art

The JAVELIN open-domain question answering system [NFM+] uses semantic parsing techniques to provide answers for the TREC 14 evaluation relational questions. Later, it was extended to answer questions in restricted domains [NMF+05].

The factoid question answering system QASR [SHTT06] extracts relevant predicate arguments from a question and sentences in the corpus using ASSERT [SSP] [PWH+04] as a semantic role labeling tool. Semantic role labeling is the task of detecting the semantic arguments that come with a certain verb. For example, given the sentence "John sent the mail to Sarah." the semantic role labeling task would be to detect that "sent" is the verb, "John" is the sender and Sarah is the receiver.

The question answering system from the National University of Singapore [RSK05] which also uses ASSERT to answer factoid and list questions in TREC 14, where relevant documents are extracted from Answers.com, if no documents were found it extracts from Google.

In thesis the open-source open-domain question answering system Open Ephyra [Sch01] has been used, which was introduced by one of Watson's creators. It is an integration between several techniques for question analysis, query generation and answer extraction. Individual approaches often suffer from low precision. Thus the system's design was based on using the advantages of previous individual approaches to build a big system and increase the overall performance.

1.3 Open Ephyra QA System

Open Ephyra is a combination of components for *question analysis*, *query generation*, *search*, *answer extraction* and *answer selection* as shown in Figure 1.1.

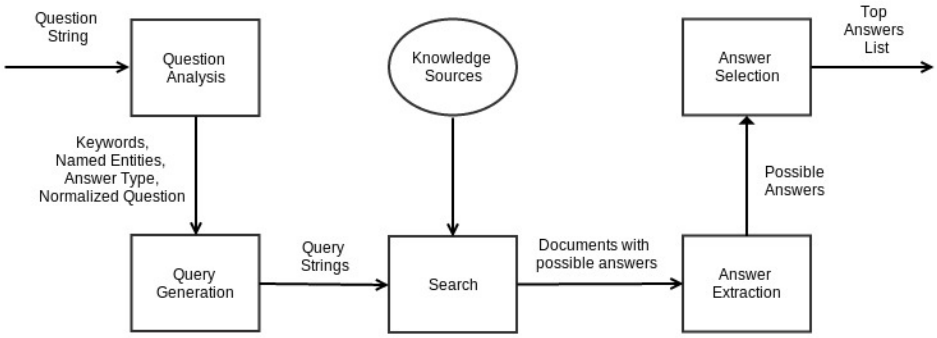

Figure 1.1: Open Ephyra's QA phases.

First, question strings are analyzed and normalized to be prepared for query generation. Afterwards, queries are generated to search the available knowledge source(s) and extract sentences with possible answers. In the answer extraction phase, a list of possible answers is produced from the sentences extracted during the search. Finally, in the answer selection phase, the list produced from answer extraction passes through filters to remove unwanted answers and produce a final list of the top results.

Open Ephyra's architecture allows the system to be extended by adding external tools or components. The system available without external additions is called the Baseline System, which uses the web as a knowledge source.

There are three answer extraction techniques available: answer type analysis, pattern learning and matching and semantic role labeling. The semantic role labeling technique requires the useage of the external semantic role labeling tool ASSERT [SSP] [PWH+04]. Therefore, the baseline system only uses the answer type analysis and pattern learning and matching techniques in answer extraction. In this thesis, the Baseline system has been used due to time constraints.

1.3.1 Question Analysis

First, the question string is transformed into a normalized form. This is achieved by:

- Removing punctuation, quotation marks and repeated white spaces.
- Expanding short form verbs (e.g. Where's is expanded to Where is).
- Substituting auxiliary verbs by simple verbs (e.g. did and die become died)

After normalizing the question string, it is analyzed to extract keywords, named entities and the expected type of the answer.

1.3.2 Query Generation and Search

Queries are generated using the information retrieved during the question analysis phase. They are expanded to increase the number of relevant documents retrieved using WordNet [Fel10], which is an electronic lexical database for English.

WordNet is used to extract synonyms for the question keywords. The keywords and their synonyms are combined to form simple boolean queries, which are known as "Bag of Words" queries.

For instance, the following is an expanded query *taken from the system* for the question "When did Albert Einstein die?":

("Albert Einstein" OR Einstein) (died OR "pass awayed" OR conked OR "kick the bucketted" OR deceased OR "cash in one's chipsed" OR "pop offed" OR perished OR choked OR exited OR "drop deaded")

Another complex type of queries is also used. It tends to produce possible structures of the answer sentence from the question string, which are later used as query strings. For instance, the system uses "Albert Einstein died in" and "Albert Einstein died on" as queries for the question "When did Albert Einstein die?".

Afterwards, the queries are used to search the available knowledge source(s), where sentences with possible answers are extracted.

1.3.3 Answer Extraction

In this section, the three answers extraction techniques and their additional steps in the questions analysis and query generation phases will be illustrated.

Answer Type Analysis

In this technique, the question string is compared to a list of question patterns to determine the expected answer type. Table 1.1 shows a sample of the question patterns and their corresponding Named Entities (NE). Question Patterns are regular expressions that match question strings to named entity types questions are asking for. Question patterns and their corresponding answer types were produced by analysing previous TREC questions.

NE type	Type Pattern
Date	When
Date (Season)	(what|which) (.*)?(season|time of (the)?year)
Date (Weekday)	(what|which) (.*)?(day of (the)?week|weekday)
Location	where
Location (City)	(what|which|name) (.*)?(city|metropolis|town|village)
Location (Country)	(what|which|name) (.*)?(colony|country|nation)
Size	how (big|large)
Size (Area)	(how large in|how many) (acre|square (foot|.*meter))
Size (Length)	how (deep|far|high|long|tall|wide)
Size (Length)	(how large in|how many) (foot|inch|.*meter|mile|yard)
Size (Volume)	(how large in|how many) (gallon|liter|ounce)

Table 1.1: NEs and their question patterns [Sch01].

After determining the expected answer type, an Answer Type Filter is used to extract possible answers with the expected type. This technique can only answer questions with known answer types. Therefore, it is used in addition to the pattern learning and matching technique, which can extract answers without knowing their type.

Pattern Learning and Matching

In this technique, two types of textual patterns are used. The first type is used to analyse and interpret question strings. The second type is used for answer extraction. The question analysis patterns are produced manually, while the answer extraction patterns are produced by the system.

This technique depends on the concept that a question asks for a *property* of a *target* in a *context*. For instance the question "What was the name of Tom Cruise in Mission Impossible?" asks for the name (property) of Tom Cruise (target) in Mission Impossible (context).

The question analysis patterns are used to extract the *property*, *target* and the *context* from the question string. Those three components are the question interpretation. Afterwards, a Question Interpretation Generator takes the question interpretation and produces queries containing the *target* and the *context*.

Queries are used to extract sentences from the knowledge source(s) with possible answers. Afterwards, the answer extraction patterns are used to extract possible answers from the sentences resulting from the search. The answers extracted are the sentences that contain the *target* and the *property* of the question.

As mentioned previously, answer extraction patterns are produced by the system. Questions and answers from old TREC evaluations are used as training data. The question interpretation and its answer are used to from query strings. Those queries are used

to search the knowledge base for sentences containing the *target* and the answer. Afterwards, Open Ephyra produces patterns from the sentences extracted. These patterns link the *target* and the *property* of a question.

An answer pattern consists of *target* and *property* tags. A pattern also has words that define the position of the *target* and the *property* in the sentence. The following is an example from [Sch01] for an answer pattern that links the day of death (property) to a person (target):

<Target> , who died in Salzburg in <Property> and

Patterns produced are specific and need to be generalized. Therefore, they are transformed into regular expressions, where they are filtered to only contain entities related to the *property*. The generalized form of the previous example looks as follows:

<Target> [^<]*died [^<]*(<Location>)?[^<]*<Property_Date>

Semantic Role Labeling

This technique uses the external semantic role labeling tool ASSERT [SSP] [PWH+04] for semantic parsing. Semantic parsers are often error prone in parsing questions, because they are trained on sentences. Therefore, question strings are first transformed into sentences.

A placeholder replaces the interrogative pronoun, which has to be of the same type, because it is used later for answer extraction. For instance, the question "When did Albert Einstein die?" is transformed into the sentence "Albert Einstein died on 2/2/2222".

The sentence produced is parsed to determine the semantic roles. For instance, given a certain action, semantic roles can determine what is the action, who did it, where, when and why did it happen. Afterwards, the placeholder is dropped and its semantic role is said to be missing.

Queries are generated and possible answers are extracted. The next step is to parse search results, but semantic parsing is time consuming, therefore the search results need to be narrowed down before parsing.

First, the length of a sentence needs to be within a certain range, otherwise the sentence is dropped. Second, a sentence is required to have a predicate with the missing semantic role in the question. Afterwards, if the answer type is known, the sentence is required to have a named entity of that type. Finally, the terms in the sentence need to be similar to the terms in the question. The concept of term similarity is introduced in section (3.3.2) in [Sch01].

If the answer type is known, answers are determined by extracting entities of that type. If not, answers are arguments with the missing semantic role in the question.

1.3.4 Answer Selection

In the Answer Selection phase, filters run on the answers list produced during the answer extraction phase to produce a list of top answers. They discard the following answers:

- Function words (e.g he, him, the, a).
- Repeated answers.
- Answers containing information in the question, this occurs when the question contains a named entity of the expected answer type.
- Answers containing an interrogative pronoun.
- Wrong structured sentences (e.g wrong bracketing).
- If an answer is a subset of another answer, the shorter answer is dropped.

Semantic parsers often do not recognize argument boundaries correctly. Therefore, the extras accompanied with the arguments are removed, which are:

- White spaces and characters (Except units symbols).
- Articles.
- Prepositions.

Chapter 2

Implementation

2.1 Open Ephyra

Open Ephyra can be downloaded from the following link:

`http://sourceforge.net/projects/openephyra/`

It can be easily set up on a machine, the only system requirements are a Java runtime environment (version 1.5 or later) and about 1 GB of free RAM [Opea]. In this thesis Open Ephyra has been used on a Linux operating system.

2.1.1 How to run Open Ephyra?

Go to *scripts/* in the Open Ephyra folder and run one of the following:

1. `OpenEphyra.sh`: to run Open Ephyra using the web as a knowledge source.
2. `OpenEphyraCorpus.sh /path/to/indri/index/` : to run Open Ephyra using a local text corpus as a knowledge source.

After running one of scripts above, Open Ephyra starts the following:

- Creating the tokenizer, sentence detector, POS tagger, chunker, syntactic parser using [Opeb].
- Creating the stemmer using [Sno].
- Creating the Stanford Parser using [Gro].
- Creating the WordNet dictionary.
- Loading query reformulators, question patterns and answer patterns.

Afterwards, an interactive shell starts and waits for questions to be entered. To exit the interactive shell type *exit*.

2.1.2 External Libraries

Open Ephyra uses several external libraries, they are located in *lib/* in the Open Ephyra folder. To add an external library, add it to *lib/* or one of its sub-folders. Moreover, the library's path needs to be added to the java class path, so java can see it during the building process. That is done by adding the library's path to the variable `CLASSPATH` in the script before running it and also adding it to the file `build.xml`.

2.1.3 Azure Bing Search

Unfortunately, the available version of Open Ephyra uses old versions of search APIs, which are now deprecated. Therefore, the system has been adapted to work with the new Microsoft Azure Bing search API as it offers 5,000 free queries per month, other APIs require purchase . The API returns the search results in the form of XML or json. Therefore, the output from the API had to be parsed, as Open Ephyra only needs the URLs and the descriptions of the documents retrieved. Thus, the library [azu] has been used.

The steps required to adapt Open Ephyra to Azure Bing search are the following:

1. Download [azu].
2. Add the downloaded jar file to *lib/search/* in the Open Ephyra folder.
3. Add the library to the java `CLASSPATH` and `build.xml` as mentioned above.
4. Sign in to `http://datamarket.azure.com/`.
5. Search for "Bing Search API" and sign up for the free package.
6. Go to My Account and save your Primary Account Key for a later step.
7. Replace the method `doSearch` in `info.ephyra.search.searchers.BingKM.java` in Open Ephyra with the following:

```
@Override
    protected Result[] doSearch(){
        AzureSearchWebQuery aq = new AzureSearchWebQuery();
        aq.setAppid("put your Primary Account Key");
        aq.setQuery(query.getQueryString());
        aq.doQuery();

        AzureSearchResultSet<AzureSearchWebResult> results  = aq
        .getQueryResult();
        ArrayList<String> snippets = new ArrayList<String>();
        ArrayList<String> urls = new ArrayList<String>();
        for (AzureSearchWebResult result : results) {
                snippets.add(result.getDescription());
                urls.add(result.getUrl());
        }
                return getResults(Collections.toStringArray
                (snippets),
                Collections.toStringArray(urls), true);
    }
```

8. Add the following imports to the class mentioned in the previous step:

```
import net.billylieurance.azuresearch.AzureSearchWebQuery;
import net.billylieurance.azuresearch.AzureSearchResultSet;
import net.billylieurance.azuresearch.AzureSearchWebResult;
```

9. Go to the Open Ephyra folder and run **ant -f build.xml** to rebuild the project.

Note: Due to incompatibility issues, some changes have been made to the library [azu]. All the occurrences of the method `.getTextConent()` have been changed to `.getFirstChild().getNodeValue()`. This has been done by downloading the source code, applying the changes and re-extracting the library as a *jar* file.

2.2 English Wikipedia Dump

Wikipedia offers all its databases for user download, they contain all articles in different languages. These databases are called Wikipedia Dumps. Users can customize the download by choosing the database of a desired language.

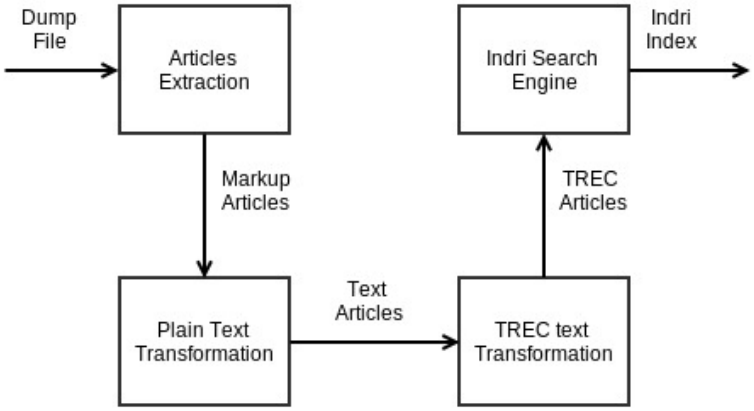

Figure 2.1: Wikipedia Dump transformation steps.

Dumps can be downloaded in an XML or an SQL format. In this thesis the XML English Wikipedia Dump has been used to train Open Ephyra. First, articles are indexed using the Indri search engine from the Leumer project [Leu]. Afterwards, the indices are given as an input to the system, which adds Wikipedia's articles as the knowledge source.

The English Wikipedia Dump can be downloaded from the following link:

`http://en.wikipedia.org/wiki/Wikipedia:Database_download`

Before indexing the dump using Indri, it needs to be in the correct format first. Figure 2.1 illustrates the dump's transformation steps.

The dump is a single XML file containing all English articles, where each article is an XML element called "page". Each page has several elements, which represent Wikipedia's information about the article. A sample page element that represents the article "Freescale DragonBall" is shown below:

```
<page>
    <title>Freescale DragonBall</title>
    <ns>0</ns>
    <id>8664</id>
    <revision>
      <id>539349743</id>
      <parentid>536640995</parentid>
      <timestamp>2013-02-21T02:56:34Z</timestamp>
      <contributor>
        <username>Legobot</username>
        <id>7304691</id>
      </contributor>
      <minor />
      <comment>Bot: Migrating langlinks to [[WP:Wikidata]] -
            [[d:q511698]]
      </comment>
      <text xml:space="preserve">{{Redirect|Dragonball|the
            Japanese manga and media franchise|Dragon Ball}}
            [[Image:Motorola DragonBallEZ XC60EZ328PU16V top
            .jpg|thumb|Motorola DragonBall EZ Microprocessor]]

            [[Motorola]]/[[Freescale Semiconductor]]'s
            '''DragonBall''', or '''MC68328''', is a [[microcon
            troller]] design based on the famous [[Motorola
            68k|68000]] core, but implemented as an all-in-one
            low-power solution for [[handheld computer]] use.
            It was designed by Motorola in [[Hong Kong]] and
            released in 1995.&lt;ref&gt;{{cite web | url=http
            ://pdadb.net/index.php?m=cpu&amp;id=c68328&amp;c
            =motorola_dragonball_mc68328 | title=Motorola
            DragonBall MC68328 CISC SoC | accessdate=2010-01-27
            | publisher=PDAdb.net}}&lt;/ref&gt;

                                              .
                                              .
                                              .


            == References ==
            &lt;references/&gt;

            == External links ==
            * [http://www.motorola.com/mediacenter/news/detail
            .jsp?globalObjectId=3248_2672_23 History of
            Motorola DragonBall]

            {{Motorola processors}}

            [[Category:68k microprocessors]]
            [[Category:Science and technology in Hong Kong]]
            [[Category:1995 introductions]]


            {{microcompu-stub}}
      </text>
      <sha1>0y96yzlhvbqvv3lme5oz46rgxdnh78u</sha1>
      <model>wikitext</model>
      <format>text/x-wiki</format>
    </revision>
  </page>
```

The actual text of an article is located between the <text>..</text> tags. Articles are written in Wiki markup, which is a language similar to HTML used to write pages in Wiki websites [WIK].

The actual text of an article is the only useful information needed to train Open Ephyra, thus it needs to be extracted. This has been achieved by looping through the Wiki dump file and reading it line by line, because the size of the file is too large to read any other way. A new file has been created for each article containing what is between *title* and *text* tags. The titles have been used to distribute the articles alphabetically on directories according to the first two letters in the article's title.

Furthermore, some articles are just redirects. For instance, check what is between the <text>..</text> tags in the following page element:

```
<page>
    <title>AccessibleComputing</title>
    <ns>0</ns>
    <id>10</id>
    <redirect title="Computer accessibility" />
    <revision>
      <id>381202555</id>
      <parentid>381200179</parentid>
      <timestamp>2010-08-26T22:38:36Z</timestamp>
      <contributor>
        <username>OlEnglish</username>
        <id>7181920</id>
      </contributor>
      <minor />
      <comment>[[Help:Reverting|Reverted]] edits by [[Special
        :Contributions/76.28.186.133|76.28.186.133]] ([[User
        talk:76.28.186.133|talk]]) to last version by Gurch
      </comment>
      <text xml:space="preserve">\#REDIRECT [[Computer
            accessibility]] {{R from CamelCase}}
      </text>
      <sha1>lo15ponaybcg2sf49sstw9gdjmdetnk</sha1>
      <model>wikitext</model>
      <format>text/x-wiki</format>
    </revision>
</page>
```

This redirects the user to the article "Computer accessibility" if the article "AccessibleComputing" was requested . Such articles are dropped, as they do not contain useful information for the training.

The python script `extractWikiArticles.py` in */home/tjr/wikipedia/extraction/* on tjr10 extracts individual articles from the dump. Another approach for extraction was to extract articles alphabetically, where scripts for all letters run at the same time. Thus, an article for each letter is extracted at a time instead of a single article at a time.

This can achieved by running the following scripts:

1. `create_directories.py` in */home/tjr/wikipedia/extraction/* on tjr10.
2. All scripts in */home/tjr/wikipedia/extraction/alphabet − extraction/* on tjr10 at the same time.

The extracted articles can be found in */import/scratch/tjr/qa/wikipedia/*.

After extracting the articles in separate files, they need to be parsed into plain text, because the Wiki markup contains links, images, references and other markup elements.

Although Indri can index several file types, Open Ephyra has been adapted to work on TREC text. Therefore, the articles have been converted into the TREC text format before building the Indri index. The TREC text format used looks as follows:

```
<DOC>
<DOCNO> AA-5557738 </DOCNO>
<TEXT>
<p> paragraph 1 </p>
<p> paragraph 2 </p>
.
.
</TEXT>
</DOC>
```

The python script `wikiDumpToTREC.py` in */home/tjr/wikipedia/extraction/TREC/* on tjr10 was used to convert the articles to TREC text. The script has to be run using python 2.7 (e.g. **python 2.7 wikiDumpToTREC.py**) to be compatible with the methods used .

2.3 Indri Search Engine

The following are the steps required to adapt Open Ephyra to use the Indri search engine for local document retrieval.

2.3.1 Set up Open Ephyra to use Indri

Download Indri from the following link:

`http://sourceforge.net/projects/lemur/`

Go to the Indri folder downloaded and run the following commands:

- **./configure --enable-java --with-javahome=<Java home directory>**
  --with-swig=<location of swig, often /usr/bin/swig>
- **make**
- **make install**

Copy the shared library `libindri_jni.so` from *swig/obj/java/* in the Indri folder to *lib/search/* in the Open Ephyra folder (see section 2.1).

If the Indri version `indri.version` in *lib/search/* does not match the version downloaded, overwrite the Java library `indri.jar` in *lib/search/* in the Open Ephyra folder with the one in *swig/src/java/* in the Indri folder .

2.3.2 Prepare for indexing

The next step is to prepare the local document collection for indexing. Ephyra retrieves individual paragraphs by default rather than whole documents. Therefore, all paragraphs of the document collection need to be surrounded by ... tags. Information on how to retrieve documents and not paragraphs will be mentioned in a later step.

2.3.3 Build Index

To build an Indri index for a local document collection, go to *buildindex/* in the indri folder and run the command **IndriBuildIndex**. This command takes parameters either from a file with an XML format or from the command line using dotted path notation. In this thesis an XML file was used to set the parameters.

The parameters file `parameters.xml` in */home/tjr/indri−5.4/buildindex/* on tjr10 was used to index the Wikipedia articles. The definitions of the parameters used are as follows [Ind]:

- corpus: a complex element which contains parameters of a certain corpus and it can be specified multiple times (e.g. if multiple corpora are used). The corpus parameters used are:
  - path: the path to a file or a directory of documents to be indexed.
  - class: the type of the file or documents in a directory. Indri can index the following types:
    - html – web page data.
    - trecweb – TREC web format (e.g. terabyte track).
    - trectext – TREC format (e.g. TREC-3 onward).
    - trecalt – TREC format (e.g. TREC-3 onward, with only the TEXT field included).
    - doc – Microsoft Word format (windows platform only).
    - ppt – Microsoft Powerpoint format (windows platform only).
    - pdf – Adobe PDF format.
    - txt – Plain text format.
- memory: an integer value to specify the memory usage of the indexing process. It is in bytes but a scaling factor can be specified by adding a suffix. A suffix is case sensitive and values available are K (1000) , M (1000000) and G (1000000000).
- index: specifies the directory where Indri will build the index. Make sure that the specified directory does not exist because Indri will override it.

- metadata: a complex element that specifies the metadata fields to index, where metadata is data about data (e.g. title, headline). The parameter used is *field* which makes the specified field available for retrieval as metadata.

- field: a complex element that states the fields to index as data. For instance, *p* which indexes what is between .. tags. The field used is *name* which is the name of the field.

- stemmer: a complex element that specifies the stemming algorithm to use, the available options are *Porter* or *Krovetz*. This parameter is optional with a default set to no stemming.

After creating the parameters file run the following command to start building the index:

- **IndriBuildIndex parameters.xml**

Finally, add the following lines to the method `initFactoidCorpus` in the main class `info.ephyra.trec.OpenEphyraCorpus.java` in *src/* in the Open Ephyra folder to add the Indri knowledge miner:

```
String [] indri_index = "/path/to/indri/index/";
Search.addKnowledgeMiner(new IndriKM(indri_index, false));
```

To retrieve whole documents instead of paragraphs add the following lines instead:

```
String [] indri_index = "/path/to/indri/index/";
Search.addKnowledgeMiner(new IndriDocumentKM(indri_index, false));
```

When Open Ephyra is run on a local corpus, it is set by default to also search the web. If this is not desired, comment the occurrences of the method `initFactoidWeb` in the method `askFactoid` or comment the initialization of Bing from the method `initFactoidWeb`.

In this thesis, an index was built for the English Wikipedia articles, it is located in */import/scratch/tjr/qa/TREC − index/*, which can be accessed from tjr2, tjr10 and tjr23.

2.4 Open Ephyra and Servlets

To use Open Ephyra as the question answering component of a spoken dialogue system, it had to be deployed on a server, because the communication between the spoken dialogue system and Open Ephyra is accomplished using HTTP GET request. Therefore, a java servlet has been created where Open Ephyra was called through it. The servlet created was run on a Tomcat 7 server.

Running Open Ephyra with a script from the command line is different from running it through a servlet. If it is run from a directory other than *scripts/* the configuration and properties files fail to load. Moreover, Tomcat requires directories to be organized in a specific way to run an application successfully. Therefore, Open Ephyra had to be adapted to run on Tomcat.

2.4.1 Open Ephyra on Tomcat

A servlet is a class in the java programming language which can run a java application on a web server, it can be considered as Java Applet which runs on a server instead of a web browser. In this thesis Java Servlets have been used to deploy the system on a Tomcat server, as Open Ephyra is programmed in java. The online version of Open Ephyra was used as the question answering component of a Spoken Dialogue System, where users can call in, ask a question and hear the answer.

The steps to run Open Ephyra through a java servlet are as follows:

1. Login as root using **sudo -s**.
2. Follow [Jav] to create a java Servlet, where `CATALINA_HOME` in ubuntu is */var/lib/tomcat7/*.
3. Copy the contents of the Open Ephyra directory except *src/* and *lib/* into $WEB-INF/$ in the servlet created.
4. Copy the contents of *src/* in the Open Ephyra folder to $WEB-INF/src/$.
5. Copy all jar files in *lib/* in the Open Ephyra folder into *lib/* in the servlet created.
6. Add *res/* and *conf/* with their full paths to `shared.loader` in `catalina.properties` located in $CATALINA\_HOME/conf/$.
7. Increase the java heap in tomcat by adding the following lines to `catalina.sh` in */usr/share/tomcat7/bin/*:

   JAVA_OPTS= "-Xms128m -Xmx3g"
   CATALINA_OPTS="-Xms128m -Xmx3g"

8. Call the method `askFactoid` in one of the main classes from the servlet.

Note: The java method `System.getProperty` does not work with tomcat, therefore the paths to the properties files were hard coded to the system.

These changes have been made to the following classes:

1. `info.ephyra.questionanalysis.atype.FocusFinder.java`
2. `info.ephyra.questionanalysis.atype.WordNetAnswerTypeMapping.java`
3. `info.ephyra.questionanalysis.atype.classifier.RuleBasedQuestionClassifier.java`
4. `info.ephyra.questionanalysis.atype.classifier.TrainedQuestionClassifier.java`

Moreover, some variables contain paths relative to the Open Ephyra directory, because the system was designed to run from it. Therefore, they have been modified to contain the full paths.

The following shows the variables changed and their classes:

1. **defaultSerializedClassifier** in `info.ephyra.nlp.StanfordNeTagger.java`
2. **parser** in `info.ephyra.nlp.StanfordParser.java`
3. **NORMALIZER** in `info.ephyra.OpenEphyra.java`

The servlet `hello` in */var/lib/tomcat7/webapps/* on tjr2 was created to run Open Ephyra online. It can be called as a web application, or internally through the spoken dialogue system as shown:

- http://it-tjr2.dhbw-stuttgart.de:8080/hello/form_input.html
- http://it-tjr2.dhbw-stuttgart.de:8080/hello/echo?question=When+did+Albert+Einstein+die%3F

Chapter 3

Experminets and Results

3.1 Test Set

The TREC 11 test set has been used to re-evaluate the system, it is a set of 500 factoid questions and answers. An answer is a pattern represented in the form of a regular expression, which can match a certain answer written in different forms. A question can have several patterns, meaning that the it might have more than one correct answer. The test data are available in *res/testdata/trec/* under the Open Ephyra directory.

During the evaluation, an answer produced by the system is considered correct if it matches the regular expression corresponding to the question. Some of the questions in the set do not have corresponding answers in the answers file, by excluding these questions we only get 444 questions with known answers.

3.2 How to run the evaluation?

To run the evaluation using a local text corpus, go to *scripts/* and run the following:

EphyraTREC8To11.sh trec11 uniqueID /path/to/indri/index

- EphyraTREC8To11.sh: a script that can run TREC evaluations from 8 to 11.
- trec11: an input that specifies the usage of the TREC 11 test set, it can also be trec8, trec9 or trec10.
- uniqueID: an input which will be used in naming the output file of the evaluation, it can be anything desired.

As mentioned previously, in TREC evaluations the participants are given text corpora to train their systems for the evaluation. Therefore, the evaluation in Open Ephyra was designed to run on a local corpus. Thus some changes need to be made to measure the performance using the Bing search API, these changes can be done as follows:

1. Go to `info.ephyra.trec.EphyraTREC8To11.java`.
2. Change the class to extend `OpenEphyra` instead of `OpenEphyraCorpus`.

To run the evaluation, go to *scripts/* and run the following:

EphyraTREC8To11.sh trec11 uniqueID

NOTE: Do not forget to change the class to extend `OpenEphyraCorpus` again if an evaluation on a local corpus will be run.

After running the evaluation, a file will be created in *logs/* with the name `trec11_uniqeID`. It contains information about each question and a list of top results produced by the system with their scores and whether they are correct or not.

3.3 Results

Open Ephyra has been previously evaluated on the TREC11 test set using the AQUAINT corpus and only considering the 444 question with known answers, it had an accuracy of 49% (218 correct out of 444) [SCCN+11]. In the previous evaluation the baseline system has been used with the semantic role labeling technique for answer extraction (i.e. the three answer extraction techniques have been used).

In this thesis, an evaluation has been run using the Web (Bing search API) as a knowledge source. Another evaluation has been run using the local English Wikipedia Dump corpus. In both evaluations the baseline system has been used without the semantic role labeling answer extraction technique.

Unlike the Bing Search API evaluation, the evaluation using the English Wikipedia Dump had some problems. Some questions caused the evaluation script to stop, but restarting the evaluation using those questions was successful. Other questions caused different problems, but due to time constraints the reasons behind those problems have not been investigated.

Therefore, some questions have been excluded from the evaluation. The test set consists of 500 questions with questions IDs from 1394 to 1893. Table 3.1 shows the IDs of the excluded questions and the reason behind their exclusion.

To rerun the evaluation starting from a certain question. The files containing the questions and patterns need to be changed to start with the question desired.

Question ID	Exclusion Reason
1456	Always Stops the evaluation script.
1580	Ran for more than a day.
1798	Filters ran for hours, which is not normal.

Table 3.1: Excluded Questions.

Note: Changing the pattern file is very important. The patterns file **must** start with the answer pattern of the first question in the questions file.

In order not to mess up the original scripts and the questions and patterns files, copies have been used to complete the evaluation. The script `EphyraTREC8To11-cont.sh` under */import/scratch/tjr/qa/OpenEphyra/scripts/* has been used. And the questions and pattern files `trec11questions-cont` and `trec11patterns-cont` respectively under */import/scratch/tjr/qa/OpenEphyra/res/testdata/trec/* on tjr23.

The exclusion of certain questions and rerunning others, caused the test set to be divided, leading to multiple results files located under */import/scratch/tjr/qa/OpenEphyra/log/*. The test set's division is shown in table 3.2.

File	Questions ID's	Questions Answered	Correct
20augAll	1394-1435	42	8
4sep	1530-1579	50	8
8sep	1582	1	0
9sep	1583-1773	191	36
20sep	1774-1797	24	1
22sep	1799-1848	50	6
26sep	1849-1893	45	6
30sep	1436-1455	20	4
2oct	1457-1504	48	11
4oct	1506-1529	24	4
Total	-	495	84

Table 3.2: TREC11 Division and Results (Wikipedia).

Note: The question with ID 1581 has not been included in the evaluation, because the evaluation could not start with it after excluding question 1580, as it has no known answer. Therefore, the questions file could not begin with a question that the patterns file does not also begin with.

Let n be the number of questions in a test set and c the number of questions answered correctly. An answer is considered correct if the first answer candidate in the top ranked list returned by the system was correct.

The accuracy is defined as:

$$Accuracy = \frac{c}{n}$$

As mentioned previously, only 444 questions have known answers. By excluding the questions in table 3.1 from questions with known answers, we get 440 questions. These are the only questions that will be considered in the evaluations (i.e. n=440).

The Bing Search API evaluation has been run on tjr10, while the Wiki Dump evaluation has been run on tjr23. The results of both evaluations in addition to the AQUAINT evaluation using $n = 440$ instead of $n = 444$ are shown in Table 3.3.

Evaluation	Correct Answers	Accuracy
AQUAINT	218	49.54%
Bing Search API	205	46.59%
English Wiki Dump	84	19%

Table 3.3: Evaluations Results.

In the Bing search API evaluation, the results show that the performance was 2.95% less than the previous evaluation of the system using the AQUAINT corpus. The statistical significance between the two evaluations has been calculated, where it resulted in a p-value of 0.38.

This shows that the difference between the results of the two evaluations is not statistically significant. Thus, it cannot be concluded that the system's performance is worse than the previous evaluation.

On the other hand, the system using the English Wikipedia dump had an accuracy of 19%. The statistical significance between both the evaluation using AQUAINT and the evaluation using the Bing API against the English Wiki dump resulted in a p-values of 0. Which shows that the system's performance is statistically significant than the original system and concludes that the system's performance is worse using the English Wikipedia Dump.

This decrease in performance may be as a result of only taking a part of the Web (Wikipedia) instead of using all of it. It might also be due to not using the Semantic Role Labeling answer extraction technique. Moreover, as the Wikipedia articles still contain some Wiki Markup, these extra information may result in a lower performance.

Chapter 4

Future Work

Since the URLs and the descriptions of the articles retrieved by the API are the only information Open Ephyra needs. The web itself can be searched instead, where required information about the articles can be retrieved by parsing the html page of the results (see Figure 4.1).

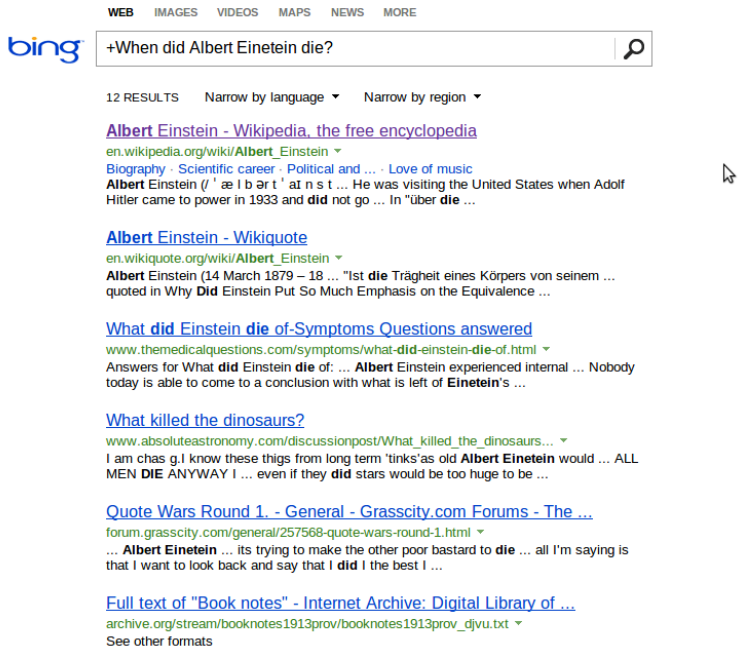

Figure 4.1: Bing search results for "When did Albert Einstein die?".

In the answer extraction phase, combining the three answer extraction techniques results in a better performance [Sch01]. Therefore, the system can be adapted to work with ASSERT. This will add the semantic role labeling technique to the baseline system. This can be achieved by downloading ASSERT from [SSP] and following the tutorial [Sch] to adapt it with Open Ephyra.

The script used to remove the Wikipedia markup does not handle complex markup cases. Thereofre, external tools as [ZMG08] [JWP] or [GWT] can be used to remove the extra markup.

Finally, The reasons behind the problems in the English Wikipedia Dump evaluation should be investigated.

Bibliography

[azu] Java implementation of bing search api hosted in windows azure. https://code.google.com/p/azure-bing-search-java/.

[Fel10] Christiane Fellbaum. *WordNet*. Springer, 2010.

[Gro] The Stanford Natural Language Processing Group. The stanford parser. http://nlp.stanford.edu/software/lex-parser.shtml.

[GWT] The gwtwiki java wikipedia api (bliki engine). https://code.google.com/p/gwtwiki/.

[Ind] Tutorial for index building by indri from the leumer project. http://www.lemurproject.org/lemur/indexing.php#IndriBuildIndex.

[Jav] Java servlets tutorial. http://www.ntu.edu.sg/home/ehchua/programming/java/JavaServlets.html.

[JWP] The jwpl java-based wikipedia library. https://code.google.com/p/jwpl/.

[Leu] Lemur toolkit for language modeling and information retrieval . http://www.lemurproject.org/.

[NFM+] Eric Nyberg, Robert Frederking, Teruko Mitamura, Matthew Bilotti, Kerry Hannan, Laurie Hiyakumoto, Jeongwoo Ko, Frank Lin, Lucian Lita, Vasco Pedro, and Andrew Schlaikjer. Javelin i and ii systems at trec 2005.

[NMF+05] Eric Nyberg, Teruko Mitamura, Robert Frederking, Vasco Pedro, Matthew W. Bilotti, Andrew Schlaikjer, and Kerry Hannan. Extending the javelin qa system with domain semantics. In *Proceedings of the Question Answering in Restricted Domains Workshop at AAAI 2005*, 2005.

[Opea] Open ephyra open source qa system. http://www.ephyra.info/.

[Opeb] Opennlp java api for various natural language processing tasks. http://opennlp.sourceforge.net/.

[PWH+04] Sameer S Pradhan, Wayne Ward, Kadri Hacioglu, James H Martin, and Daniel Jurafsky. Shallow semantic parsing using support vector machines. In *HLT-NAACL*, pages 233–240, 2004.

[RSK05] Y.F. Tan H. Cui T.-S. Chua R. Sun, J. Jiang and M.-Y. Kan. Using syntactic and semantic relation analysis in question answering. In *Proceedings of the Fourteenth Text REtrieval Conference*, 2005.

[SCCN+11] Nico Schlaefer, Jennifer Chu-Carroll, Eric Nyberg, James Fan, Wlodek Zadrozny, and David Ferrucci. Statistical source expansion for question answering. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 345–354, New York, NY, USA, 2011. ACM.

[Sch] N. Schlafer. Using the Semantic Answer Extractor Tutorial. https://mu.lti.cs.cmu.edu/trac/Ephyra/wiki/Docs/Tutorials/UsingSemanticAnswerExtractor.

[Sch01] Nico Schlafer. Deploying semantic resources for open domain question answering. Master's thesis, Universitat Karlsruhe, May 2001.

[SHTT06] Svetlana Stenchikova, Dilek Hakkani-Tür, and Gökhan Tür. Qasr: question answering using semantic roles for speech interface. In *INTERSPEECH*, 2006.

[Sno] The snowballstemmer from weka. http://weka.wikispaces.com/Stemmers.

[SSP] Kadri Hacioglu James H. Martin Daniel Jurafsky Sameer S. Pradhan, Wayne Ward. ASSERT. http://cemantix.org/software/assert.html.

[WIK] Wikipedia markup language. http://en.wikipedia.org/wiki/Help:Wiki_markup.

[ZMG08] Torsten Zesch, Christof Müller, and Iryna Gurevych. Extracting lexical semantic knowledge from wikipedia and wiktionary. In *LREC*, volume 8, pages 1646–1652, 2008.