THE EPHYRA QA SYSTEM AT TREC 2006

N. Schlaefer and P. Gieselmann

Interactive Systems Labs
University of Karlsruhe
Germany
{nico,petra}@ira.uka.de

G. Sautter

Institute for Program Structures and
Data Organization
University of Karlsruhe
Germany
sautter@ipd.uka.de

ABSTRACT

The Ephyra QA system has been developed as a flexible open-domain QA framework. This framework allows us to combine several techniques for question analysis and answer extraction and to incorporate multiple knowledge bases to best fit the requirements of the TREC QA track, in which we participated this year for the first time. The techniques used include pattern learning and matching, answer type analysis and redundancy elimination through filters. In this paper, we give an overview of the Ephyra system as used within TREC 2006 and analyze the system's performance in the QA track.

1. INTRODUCTION

Instead of focusing on the optimization of a single approach, we combine several techniques for question analysis and answer extraction to best fit the requirements of the track. Individual techniques usually have weaknesses regarding their precision and the types of questions they cover, thus we integrate multiple approaches to build a strong overall system.

For the factoid and list questions, we combine a simple approach based on answer type analysis with a more sophisticated pattern learning approach. The first approach determines the expected answer type of a question from a hierarchy of named entity (NE) types and chooses an appropriate NE tagger to extract entities of that type. While this approach has a high precision, it fails if the answer type cannot be determined or the answer cannot be tagged. Thus, we complement it with another approach that uses textual patterns to classify and interpret questions and to extract answers from text snippets. The interpretation of a question abstracts from the original question string while preserving its semantics and allows forming a query that is largely formulation-independent. The patterns for answer extraction are automatically learned using a set of question-answer pairs as training data.

Our approach for the 'other' questions is based on a pipeline of answer selection components that gradually eliminate redundant information and text snippets that do not provide sensible information on the target object. We experimented with various features for detecting redundancy and eliminating irrelevant information and multiple arrangements of these features in the pipeline.

The remainder of this paper is organized as follows. Section 2 gives an overview of our QA framework. Section 3 deals with the factoid component, Section 4 describes how we extended it to address the list questions and Section 5 is about the 'other' component. In Section 6 deals with the results of the three TREC runs we submitted and their interpretation. Finally, Section 7 summarizes the key findings and outlines open issues for future work.

2. SYSTEM OVERVIEW

Our experiments for the QA track are based on our QA framework Ephyra, introduced in [1]. Ephyra is organized as a pipeline composed of standardized components for query generation, search and answer selection. The components can be combined and arranged arbitrarily, and different configurations can be used for different question types. This architecture facilitates experimenting with various setups and allows integrating multiple QA techniques and knowledge sources in one system.

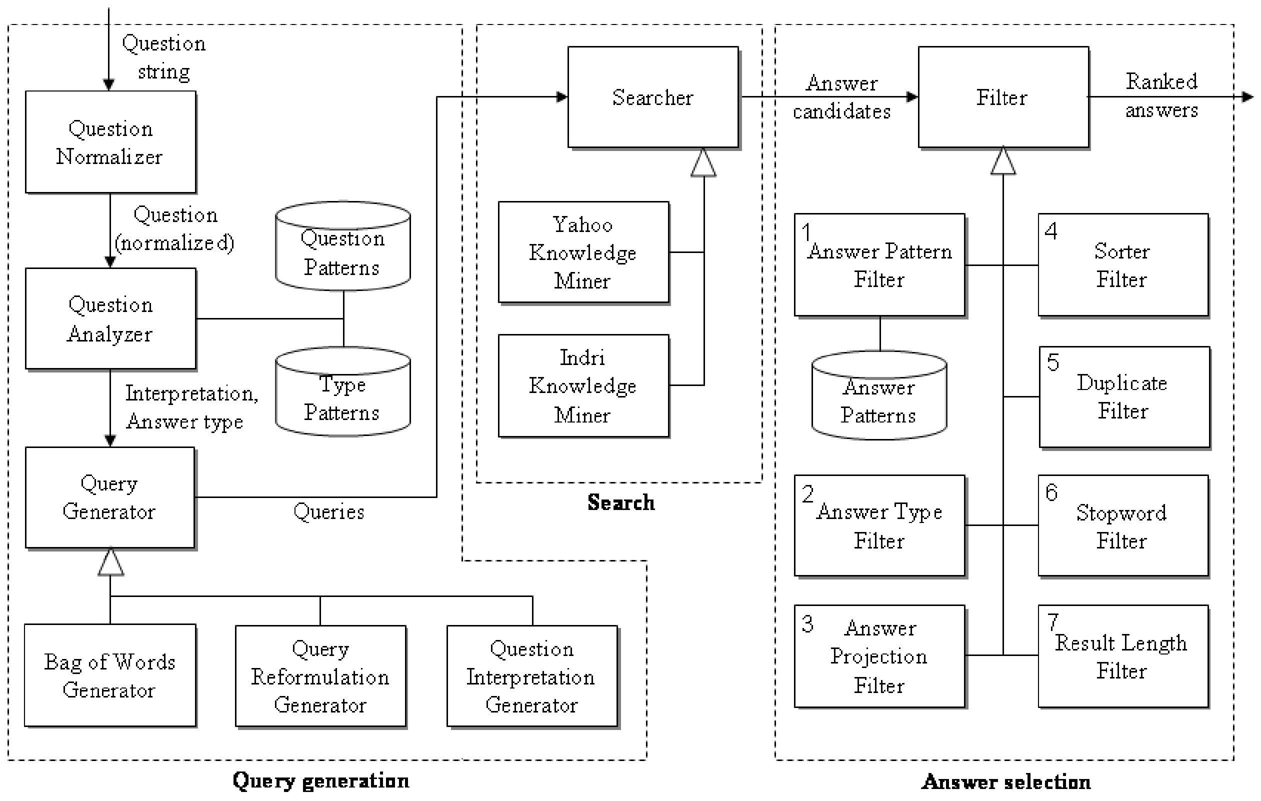

Fig. 1. Pipeline for factoid and list questions.

To incorporate a new technique for question analysis or answer extraction or a new knowledge source, one can simply plug in an additional component. Figure 1 shows the setup of our pipeline for factoid and list questions.

At first the question string is normalized, e.g. we drop unnecessary tokens and stem verbs and nouns. The Question Analyzer then determines the expected answer type of the question and interprets the question to create a more concise representation of the question string. Given this information, a set of query generators transforms the question into queries for document retrieval. We create a ”bag of words”, which is a simple set of keywords, as well as more specific queries such as reformulations of the question string.

Ephyra supports two types of search components to incorporate external information sources. Knowledge miners are used to search an unstructured source using an underlying IR system, e.g. a Web search engine. Knowledge annotators extract answers from semi-structured sources such as a gazetteer or a web service that provides weather information. Although knowledge annotation improves the performance on specific question types, we did not use it for the TREC evaluation.

The search results are passed through a set of filters to create a ranked list of answers. A filter eliminates bad results (concatenations of function words, duplicate answers, etc.), creates new results from existing ones (e.g. by extracting factoid answers from a text snippet) or rearranges the results according to a specific feature such as the similarity to the question string.

3. FACTOID COMPONENT

For the factoid questions, we combine two answer extraction techniques: a simple approach based on answer type analysis (Section 3.4) and a pattern matching approach that uses textual patterns to classify and interpret questions and to extract answers from text snippets (Section 3.5). For either approach, we first extract answers from the Web and then search the AQUAINT corpus for supporting documents. In the following, we discuss the components of our pipeline for factoid questions and describe how results from the two answer extraction approaches are integrated.

3.1. Coreference Resolution

As in TREC 2005, the test set consists of question series where each series asks for information regarding a particular target. In this way, each series is an abstraction of a user session with a QA system so that the questions may depend on the target, preceding questions or their answers in the same series. Therefore, we need to resolve coreferences prior to submitting a factoid or list question to our pipeline.

The goal of the coreference resolution is to resolve referring expressions by the question target to produce an independent, self-contained natural language question for subsequent processing. We resolve personal, possessive and demonstrative pronouns, such as ”he”, ”her”, ”their”, ”this”. Our assumption was that pronouns in a question always refer to the target and not to a previous question or to an answer of a previous question since in the question sets of the last years we only found a handful of pronouns which could be interpreted as referring to a preceding question or answer. This assumption is supported by the fact that also other groups only resolve the pronouns to the target [2, 3]. In addition, to resolve also to previous questions or answers, a deep semantic analysis of the question would be necessary which we do not have at the moment.

We replace an occurrence of a possessive pronoun with ”X’s” where X is the target. All personal pronouns are replaced by the target. For demonstrative pronouns, we replace the whole phrase including the pronoun by the target:

> **Target:** Bushehr Nuclear Facility
> **Question:** In what country is this facility located?
> **After Resolution:** In what country is Bushehr Nuclear Facility located?

In addition, we check whether ”her” is used as possessive pronoun or as personal pronoun by using part of speech information from the OpenNLP tagger [4] on the succeeding words: If the succeeding word is a noun, it is unlikely that ”her” is used as a personal pronoun. In this way, we can resolve questions such as ”Who were her family members?” to ”Who were Carolyn Bessette Kennedy’s family members?”.

3.2. Question Normalization

The question normalizer generates two representations of a factoid question, one used for question analysis with textual patterns, one to generate query strings for document retrieval. For either representation, we first drop unnecessary punctuation marks and phrases such as ”so-called” or ”approximately”. To facilitate pattern matching, all verbs are replaced by their infinitives and all nouns by their singular forms. For query generation, we modify verb constructions with auxiliaries that differ in questions and corresponding answers, e.g. ”did ... occur” is replaced by ”occurred”.

3.3. Query Generation and Search

We extract the keywords from the second representation to build a simple ”bag of words” query (Bag of Words Generator). In addition, we generate a number of reformulations of the question string that anticipate what a responsive sentence may look like (Question Reformulation Generator). For instance, the question ”Where was Mozart born?” is transformed into the query string ”Mozart was born in”. The Question Interpretation Generator creates a query string from a concise representation of the question, which we call its interpretation (see Section 3.4).

Our factoid pipeline comprises two knowledge miners (KMs) that retrieve text snippets given these query strings. The Yahoo KM returns the snippets fetched by the Yahoo API, while the Indri KM uses the Indri IR system [5] to search the AQUAINT document collection. We configured Indri to retrieve individual paragraphs rather than documents. Before indexing the corpus, we ensured that all paragraphs are properly tagged. The question reformulations are only used for the Web search since they require a redundant data source.

3.4. Answer Extraction by Answer Types

We analyzed the TREC questions from previous years and devised a hierarchy of named entity (NE) types that factoid questions frequently ask for. Our hierarchy comprises about 70 NE types such as *Date*, *Location*, *Color*, *Person*, *Size* or *Number* on the top level and *Length*, *Area* or *Volume* as subtypes of *Size*. For each of the types, we specified one or more type patterns, i.e. regular expressions that match questions asking for that particular type. Table 1 shows an extract of our NE hier-

NE type	Type pattern
Date	when
Date → Weekday	(what|which|name) (.*)?(day of (the)?week|weekday)
Location	where
Location → Country	(what|which|name) (.*)?(colony|country|nation)
Size	how (big|large)
Size → Length	how (deep|far|high|long|tall|wide)
Size → Length	(how large in|how many) (foot|inch|.*meter|mile|yard)

Table 1. NE hierarchy and type patterns.

archy. In the question analysis phase, we try to match the question string with each of the type patterns to determine potential answer types. If a question matches more than one pattern, we use a couple of tie-braking rules to select the most likely NE type, e.g. we prefer longer and thus more specific patterns over shorter ones.

During answer extraction, the Answer Type Filter applies a NE tagger which is appropriate for the expected type to the text snippets retrieved by Yahoo or Indri. For NEs of type *Person*, *Organization* or *Location*, we use the taggers from the OpenNLP toolkit [4]. For the other types, we devised our own taggers which are either rule-based (e.g. *Number*, *Acronym*) or list-based (e.g. *Language*, *Color*). The filter first tries to extract entities of the most specific type and if no entities could be tagged, it moves upwards in the hierarchy. For instance, when looking for a country name, it first tries the *Country* tagger, followed by the tagger for *Location*. The extracted NEs are normalized (i.e. tokenized and stemmed) to identify similar NEs. One representative is chosen for each cluster of similar entities, and it is assigned a score equal to the number of entities in the cluster.

3.5. Answer Extraction by Pattern Matching

Our pattern matching approach uses textual patterns to classify and interpret questions and to extract answers from text snippets. The interpretation of a question abstracts from the original question string while preserving its semantics and allows forming a query that is largely formulation-independent. The patterns for answer extraction are learned automatically using question-answer pairs as training data. In [1], we discussed our pattern matching approach in detail. This subsection gives an overview of the basic ideas and describes recent enhancements to improve the recall of answer extraction.

We assume that a question is fully specified by three components: A question asks for a *property* of a *target* in a specific *context*. For instance, the question ”What is the job of Mel Gibson in Conspiracy Theory?” asks for the *profession* (property) of *Mel Gibson* (target) in the movie *Conspiracy Theory* (context). In the question analysis phase, a set of manually defined question patterns is applied to the question string to extract these components. The Question Interpretation Generator then transforms the interpretation into a query string, which is submitted to the search component to retrieve relevant text snippets.

A second set of patterns is used to extract answers from text snippets comprising both the target object and the desired property of the target, in our example the profession *taxi driver*. For instance, the following is an answer pattern for the property *profession*:

```
<Target> works as a <Property>.
```

Our system learns the answer patterns from question-answer pairs, e.g. using data from former TREC evaluations. Each question is interpreted and a query string is formed, comprising the interpretation of the question and its answer, which is known from the training data. The query string allows to fetch text snippets from a Web search engine that contain answer patterns for a specific property, in our example patterns that relate working people to their jobs. An answer pattern covers the target, the property, an arbitrary string in between these objects plus one token preceding or following the property to indicate where it starts or ends.

In [1], we came to the conclusion that the pattern matching approach suffers from a relatively low recall because the answer patterns are often too specific. Thus, we extended our approach to derive generic answer patterns from the patterns returned by the learning algorithm. At first, we replace all named entities by their

Original pattern:	`<Target>, who was born in 1879 in <Property> and`
Generic pattern:	`<Target> [^<]*born [^<]*(<NEdate> )?[^<]*<Property_NElocation>`
Illustration:	We are looking for a location name. The string in between the given person's name and the location must contain the word "born" and it may also include a named entity of type date. In addition, it may comprise any other tokens but no further named entities.

Table 2. Answer pattern for the property *birthplace* and its generalization.

types, reusing the taggers that we devised for our NE hierarchy described in the previous section. We also try to associate the property with a NE type for two reasons: (1) If the NE type of the property is known, we do not need the tokens adjacent to the property as delimiters, thus we can drop them to create a more general pattern. (2) We can ensure that only entities of the desired type are extracted with that pattern. Furthermore, we make all tokens which are not NEs optional, except for words that express the desired relationship between the target and the answer (e.g. the verb "works" in patterns for the property *profession*). A keyword is considered important if it occurs in one of the question patterns for the respective property, regardless of its grammatical form. Table 2 gives an example of a very specific pattern and its generalization, along with a description in natural language to illustrate its meaning. The extracted property is the *birthplace* of a person.

The generic patterns greatly improved the recall of our pattern matching approach, allowing us to adapt it to little redundant sources such as the AQUAINT document collection. However, our ranking mechanism for the extracted answers turned out to be more stable when using a large, redundant knowledge source. We apply a statistical approach that calculates scores for the extracted answers by cumulating the confidence measures of the answer patterns used to extract them (see [1]). Thus, we decided to extract answers from the Web first and then project them onto the corpus.

3.6. Answer Projection

We reapply the previously described extraction strategies to the corpus to determine supporting documents for the answers extracted from the Web. To minimize the number of unsupported answers, we decided to always prefer documents identified with pattern matching over those found by the answer type approach. We assume that the answer patterns in our pattern matching approach express the desired semantic relationship between the question and the answer and thus a document that matches one of the patterns is likely to be supportive. In case neither approach detects the Web answer in the corpus, we simply browse through the paragraphs returned by the Indri IR system in the order of their relevance and select the first hit as the supporting document. Answers which cannot be found in the corpus are eliminated from the ranked list of answer candidates. To assess the performance of this answer projection strategy, we calculated its accuracy as follows:

$$Accuracy_{AP} = \frac{\#\ globally\ right}{\#\ globally\ right\ +\ \#\ unsupported}$$

Our best run achieved an accuracy of 0.877, while the average over all 59 runs is 0.855.

3.7. Answer Selection

Given the answer candidates from the document collection, the Sorter Filter arranges them according to their score. The Duplicate Filter then compares the answers pairwise. Whenever it detects two similar answers, it drops the lower ranked answer and adds its score to the score of the higher ranked one. Answers are considered similar, if they contain content words that have the same stem, e.g. "Americans" and "American people". This aggressive strategy did a good job on the TREC questions, but we are aware that it would fail for list questions like "What are the members of the Kennedy clan?".

The Stopword Filter drops malformatted answers, e.g. ones that contain a single bracket, answers that repeat information from the question string and ones that do not provide any information, i.e. concatenations of function words and special characters.

Finally, the Result Length Filter ensures that the list of answers does not exceed the maximum number of 7000 non-whitespace characters. In addition, we cut off all answers with scores below an absolute threshold. If the final list of answers is empty, we return "NIL" to indicate that we could not find an answer in the document collection. We submitted three runs, using different thresholds that resulted in a high accuracy on past TREC questions.

We did not take any action to address time dependent questions. Still, only two of our responses were judged locally correct, which indicates that time dependencies were not yet in the focus of this year's track.

3.8. Integration of Answer Extraction Approaches

The answer type approach turned out to have the higher precision, but it fails if the answer type cannot be determined or the answer cannot be tagged. Thus we always prefer its answers over results obtained with pattern matching, which we use as a backup for the remaining questions. On the other hand, our pattern matching approach is more suitable for determining supporting documents and is therefore the preferable approach for answer projection.

4. LIST COMPONENT

The component for list questions is built on top of the factoid component. Initially, we transform a list question into a factoid question asking for a single instance by replacing formulations such as "what are the names of all" or "list some of" with the string "name". The resulting question string can be handled in the same way as a factoid question, except that a list of factoid answers is returned rather than a single answer. We do not need to care about plural forms because all verbs and nouns are stemmed by the question normalizer in our factoid pipeline before pattern matching.

We do not apply an absolute threshold to the answers since the list questions are guaranteed to have a correct answer in the document collection. Instead, we use a threshold relative to the score of the top answer. For instance, a relative threshold of 10% cuts off answers with a score of less than 10% of the top score. Again, we tried out different thresholds that produced good results on past TREC data.

5. 'OTHER' COMPONENT

Our approach for the 'other' questions is based on a sequence of filters that gradually eliminate redundant and little relevant information, text snippets that do not refer to the target object, and ones that do not provide sensible information at all. The evidence used for determining redundancy ranges from above noun phrase level down to the individual terms. We assume relevance as closely related to redundancy, thus we aggregate the former when eliminating the latter. The modular architecture of Ephyra enabled us to evaluate various feature combinations and to arrange the features in an optimal manner. In the following subsections, we point out our key ideas and how we implemented them. Finally, we explain the order in which we apply the different filters.

5.1. Snippet Retrieval

The starting point is a set of paragraphs retrieved from the AQUAINT corpus using the Indri IR engine. As opposed to the factoid and list questions, we do not use answer projection for the 'other' questions, but retrieve the snippets directly from the document collection. Snippets returned by a Web search engine often contain noise and incomplete sentences and the projection of an information nugget would require a deep semantic analysis of the nugget.

5.2. Reduction of Answer Nugget Size

In order to increase precision, or to pack as many snippets as possible in the maximum answer size in favor of recall, one of our goals was to reduce the size of individual snippets as far as possible. Looking at the nuggets declared vital or OK in the TREC 13 and TREC 14 tasks, and investigating the snippets that were rated as covering them, we decided to return sentence-level nuggets as answers to the 'other' questions. The assumption of using sentences proved sensible in other recent systems as well [3]. We apply the sentence splitter from OpenNLP to split the paragraphs returned by Indri.

Further investigation of results from the recent years revealed that sentences often include unnecessary prefixes that can be cut. In particular, we cut leading news agency acronyms and locations (e.g. "PARIS (France) AFP _ ..."), and leading introductions of indirect speech (e.g. "Secretary of Foreign Affairs Condoleezza Rice said that ..."). This is accomplished relatively straightforward with regular expressions.

5.3. Elimination of Useless Snippets

An examination of the individual sentences yielded by the splitter revealed two major categories of likely useless snippets, which frequently appear if the target is a person or organization.

The first category of useless snippets subsumes enumerations of proper names. In case the target is an organization, this might be a list of stock prices, in case the target is a singer or band, it might be a complete track list of an album. To filter out such snippets, we exploit the observation that almost all words in such a snippet are either stopwords or part of a proper name, and consequently capitalized. In particular, we eliminate all snippets in which more than half of the non-stopwords are capitalized. The risk of useful snippets falling to this filter turned out to be very low.

The second category subsumes snippets that contain both the first and the last name of a person target, but as part of two different named entities. For the query "George Bush", the retrieval and splitting could also return a sentence like "Kate Bush wrote a song on the life of George Foreman", which well contains both of the query terms, but is not related to the target in any sense. Thus, for proper name targets, we aim to rate down sentences that do not contain the target name as a whole. On the other hand, we decided not to eliminate such sentences completely, since a sentence like "US President Bush often refers to George Washington in his speeches" would fall to this filter, but it might well cover a vital nugget. Therefore, we score up all snippets with the square of the number of subsequent, correctly ordered target terms, which we call keyword score.

It also occurs that a sentence does not contain a keyword from the target at all, since the Indri engine returns complete paragraphs in our setting. In this case, we do not immediately eliminate the sentence anyway, but first check the keyword score of the previous sentence. The idea is that if the latter is high, the current sentence might well contain a coreference to the target. In this case, we retain it in the set of possible result nuggets. If the keyword score of the previous sentence is low, we eliminate the sentence.

5.4. Step-by-Step Elimination of Redundancy

First we drop snippets that are exact duplicates or parts of other snippets, for each elimination cumulating the score in the snippet we retain.

The results of Roussinov et al. [6] have shown that redundancy elimination based on NP-VP-NP triplets is a promising approach to answering the 'other' questions. However, it does not properly eliminate reformulations of the same information. To overcome this problem, we extended Roussinov's idea and created an iterative approach for redundancy elimination, which starts with the triplets, but also contemplates smaller units of text.

The first step is close to the original method. We first sort the snippets by their current scores in descending order. Starting with the top one, we afterwards extract all triplets from each snippet. We eliminate all snippets that only contain triplets already covered by higher ranked ones, transferring their scores to those snippets that were the first ones to contain the triplets. In this way, we eliminate snippets that comprise only redundant associations of noun phrases. We do not use an ontology for the identification of synonymous verbs at the moment.

After the triplet step, we re-sort the remaining snippets according to their new scores and apply the same elimination procedure based on individual noun phrases instead of triplets. In this way, we eliminate snippets that do not provide new terms associated with the target.

As the last step, we again re-sort the snippets, and use individual terms as the criterion for the same procedure. Thus we eliminate snippets that do not contain terms not covered by higher ranked ones. Again, the score of eliminated snippets is transferred to the snippets due to which they are eliminated. This step aims to diversify the snippets as far as possible, increasing the breadth of information covered in the answers.

In this last step, we also filter out a third category of useless snippets, which is somewhat specific to the TREC task. It subsumes snippets providing information that was already the answer to one of the factoid or list questions on the target. To eliminate such snippets, we check which portion of the terms was part of a previous answer. If this portion exceeds a threshold, we eliminate the snippet. The threshold is not fixed, but depends on the total number of available snippets.

5.5. Order of Filters

The order in which to apply the filters is all but straightforward. Since the filters exploit different types of evidence, and serve different purposes, one might eliminate the evidence a subsequent one builds on if the order is unfavorable. Figure 2 displays the system setup for the 'other' questions.

To facilitate the detection of redundancy, we first apply the filter for splitting the snippets into sentences

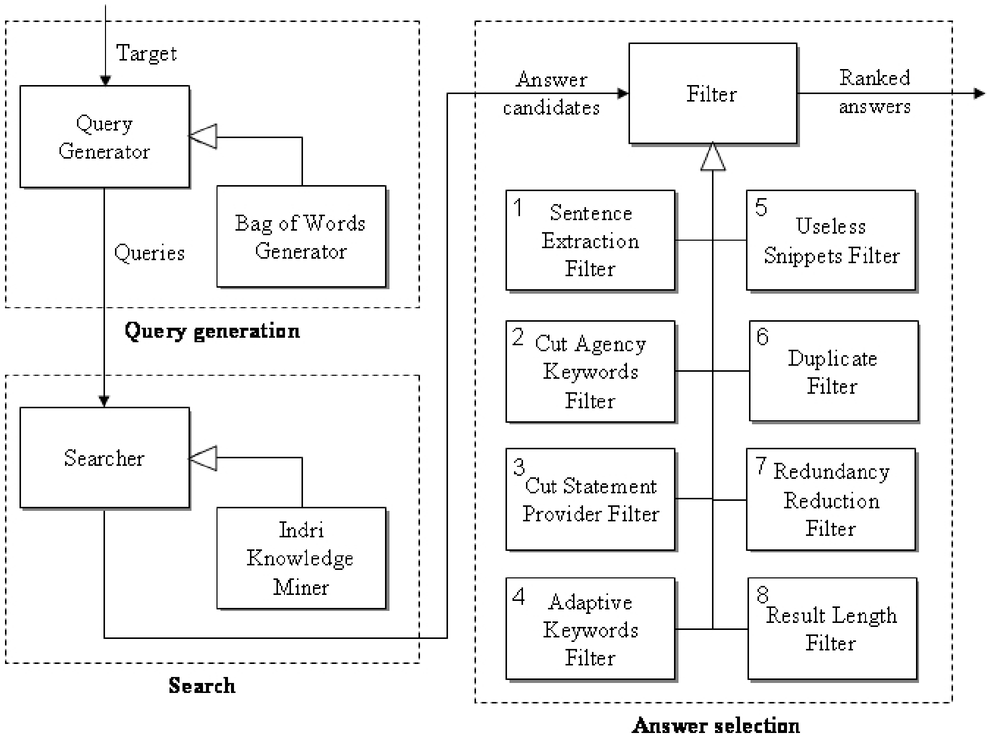

Fig. 2. Pipeline for 'other' questions.

(Sentence Extraction Filter, 1) and cut off unimportant parts (Cut Agency Keywords Filter, 2, Cut Statement Provider Filter, 3, and Adaptive Keywords Filter, 4). The idea is that the shorter the snippets are, the easier it becomes to detect redundancy. We then eliminate the snippets rated as useless due to one of the two critera discussed previously (Useless Snippets Filter, 5). This could not be done before the splitting and cutting because the unimportant parts might veil the evidence used. Now that we have made as sure as possible that only snippets with sensible information remain, we eliminate redundant snippets (Duplicate Filter, 6, and Redundancy Reduction Filter, 7). This could not be done before, since the useless or unimportant information could influence the elimination process.

At last, we re-sort the snippets according to their final scores, and return as many as (a) remain, or (b) stay below the 7000 character limit, whichever number is lower (Result Length Filter, 8).

6. EVALUATION RESULTS

We submitted three runs, differing only in the absolute threshold for factoid questions and the relative threshold for list questions. Table 3 shows the setup for each run, the resulting factoid, list and 'other' scores and the average per-series score. The last column gives the median over all 59 runs from all groups. The differences in the 'other' scores are due to variations in the assessors' judgments; we submitted identical responses for all three runs.

The results show that integrating different techniques for query generation and answer extraction in a pipeline of standardized components is a promising strategy. Our modular approach allowed us to develop a QA system from scratch in just a couple of months. Yet we had to cut short on time intensive tasks such as the generation of patterns for question analysis and the development of suitable NE taggers for answer extraction. As a consequence, our system often failed on manageable questions because of a limited coverage of our patterns or due to missing or incomplete taggers. Also, for some questions our answer selection component failed to eliminate obviously wrong answers, and thus the correct answer, although extracted from the corpus, was not among the top results.

We evaluated our 'other' component on TREC 13 and TREC 14 with results above 25% F-Score. Our re-

	ISL1	ISL2	ISL3	Median (59 runs)
Absolute factoid threshold	0	1	25	N/A
Relative list threshold	1%	2.5%	10%	N/A
Unsupported (U)	18	18	11	N/A
Inexact (X)	26	26	23	N/A
Locally correct (L)	2	2	2	N/A
Factoid accuracy	0.196	0.196	0.196	0.186
List F_1	0.092	0.096	0.097	0.087
Other F_3	0.143	0.150	0.145	0.125
Other F_3 (pyramid score)	0.160	0.162	0.160	0.139
Average per-series score	0.139	0.143	0.141	0.134

Table 3. TREC15 results.

sult on TREC 15 (15% F-Score) is significantly above the median (12.5%), but far from the best group (25%). This relative regress emphasizes the positive development in the 'other' question discipline. Our approach yielded rather good results for targets the corpus provides many snippets for. On the other hand, it did not produce any result snippets for some questions, which makes us think that our elimination techniques should (a) be less aggressive, and (b) better adapt to the number of available snippets.

7. CONCLUSION AND OUTLOOK

We deployed the Ephyra QA framework to build a QA system from standardized components, integrating different approaches for query generation and answer extraction. Individual techniques are often limited in the types of questions they can handle or have a low precision and by combining them, a strong overall system can be built. Our modular architecture facilitates the integration and improves the reusability of the individual modules.

The component for factoid and list questions combines pattern learning and matching techniques with answer type analysis. We devised a hierarchy of frequent answer types and associated each type with patterns for question analysis and an appropriate tagger for answer extraction. Our pattern matching approach interprets a question by creating a concise representation of the question string that preserves the semantics. Patterns for answer extraction are learned from question-answer pairs using the Web as a resource for pattern retrieval. We showed how the answer patterns can be generalized to allow answer extraction from little redundant knowledge sources such as AQUAINT.

For the 'other' questions, we focused on eliminating redundant and irrelevant information, using a set of filter components that are arranged in a pipeline. We returned nuggets on the sentence level, trying to cover as much relevant information on the target as possible.

For further work, we plan to deploy additional techniques for answer selection from candidates provided by our extraction strategies. By double-checking answers with multiple semantic resources such as gazetteers or WordNet, the accuracy of answer selection can be improved as shown in [7]. Additional type-specific filters could help to rule out non-responsive answers. Furthermore, we would like to incorporate semantic knowledge from ontologies and shallow semantic parsing to improve the capability of our system to extract answers from resources that provide little redundancy.

Acknowledgment

This work was supported in part by the DFG as part of the SFB 588 and by the European Commission under project CHIL (contract #506909).

8. REFERENCES

[1] N. Schlaefer, P. Gieselmann, T. Schaaf, and A. Waibel, "A pattern learning approach to question answering within the Ephyra framework," *Proceedings of the Ninth International Conference on TEXT, SPEECH and DIALOGUE*, 2006.

[2] J. Chu-Carroll, K. Czuba, P. Duboue, and J. Prager, "IBM's PIQUANT II in TREC2005," *Proceedings of the Fourteenth Text REtrieval Conference*, 2005.

[3] R. Gaizauskas, M. A. Greenwood, H. Harkema, M. Hepple, H. Saggion, and A. Sanka, "The University of Sheffield's TREC 2005 Q&A experiments," *Proceedings of the Fourteenth Text REtrieval Conference*, 2005.

[4] "OpenNLP," http://opennlp.sourceforge.net/.

[5] "Indri IR engine," http://www.lemurproject.org/.

[6] D. Roussinov, M. Chau, E. Filatova, and J. A. Robles-Flores, "Building on redundancy: Factoid question answering, robust retrieval, and the "other"," *Proceedings of the Fourteenth Text REtrieval Conference*, 2005.

[7] J. Ko, L. Hiyakumoto, and E. Nyberg, "Exploiting multiple semantic resources for answer selection," *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, 2006.