The University of Amsterdam at QA@CLEF 2005

David Ahn Valentin Jijkoun Karin Müller
Maarten de Rijke Erik Tjong Kim Sang
Informatics Institute, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{ahn,jijkoun,kmueller,mdr,erikt}@science.uva.nl

Abstract

We describe the official runs of our team for the CLEF 2005 question answering track. We took part in the monolingual Dutch task, and focused most of our efforts on refining our existing multi-stream architecture, and porting parts of it to an XML-based platform.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: H.3.1 Content Analysis and Indexing; H.3.3 Information Search and Retrieval; H.3.4 Systems and Software; H.3.7 Digital Libraries; H.2.3 [**Database Management**]: Languages—*Query Languages*

General Terms

Measurement, Performance, Experimentation

Keywords

Question answering, Questions beyond factoids

1 Introduction

In previous participations in question answering tracks at both CLEF and TREC we developed a *multi-stream* question answering architecture which implements multiple ways of identifying candidate answers, complemented with elaborate filtering mechanisms to weed out incorrect candidate answers [1, 2, 8, 9]. Part of our efforts for the 2005 edition of the QA@CLEF track were aimed at improving this architecture, in particular the so-called table stream (see Subsection 2.2). Also, to accommodate the new questions with temporal restrictions, a dedicated module was developed. The bulk of our efforts, however, was aimed at porting one of the streams to a "pure" QA-as-XML-retrieval setting, where the target collection is automatically annotated with linguistic information at indexing time, incoming questions are converted to semistructured queries, and evaluation of these queries gives a ranked list of candidate answers.

Our main findings this year are the following. While our system provides *wrong* answers for less than 40% of the test questions, we identified some obvious areas for improvement. First, we should work on definition extraction so that both questions asking for definitions and questions requiring resolving definitions can be answered in a better way. Second, we should examine inheritance of document links in the answer tiling process to make sure that the associated module does not cause unnecessary unsupported answers. And third, and most importantly, we should improve our answer filtering module to make sure that the semantic class of the generated answer corresponds with the class required by the question.

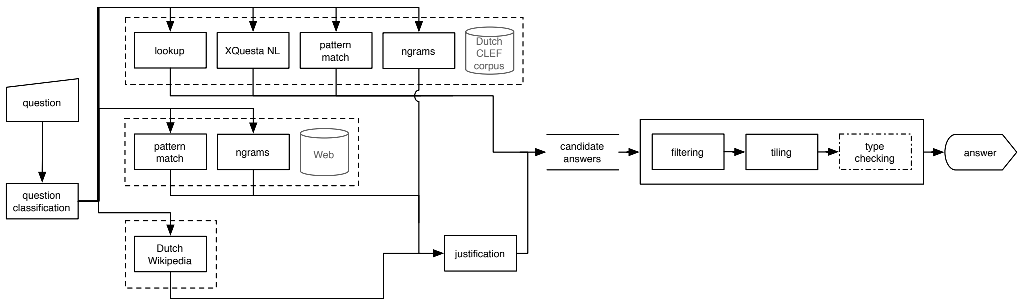

Figure 1: Quartz-2005: the University of Amsterdam's Dutch Question Answering System.

The paper is organized as follows. In Section 2, we describe the architecture of our QA system. In Section 3, we describe the new XQuesta stream, i.e., the stream that implements QA-as-XML-retrieval. In Section 4, we detail our official runs. In Section 5, we discuss the results we have obtained and give a preliminary analysis of the performance of different components of the system. We conclude in Section 6.

2 System Overview

2.1 Architecture

Many QA systems share the following pipeline architecture. A question is first associated with a question type, from a predefined set such as DATE-OF-BIRTH or CURRENCY. Then, a query is formulated based on the question, and an information retrieval engine is used to identify a list of documents that are likely to contain the answer. Those documents are sent to an answer extraction module, which identifies candidate answers, ranks them, and selects the final answer. On top of this basic architecture, numerous add-ons have been devised, ranging from logic-based methods [12] to ones that rely heavily on the redundancy of information available on the World Wide Web [4].

Essentially, our system architecture implements multiple copies of the standard architecture, each of which is a complete standalone QA system that produces ranked answers, though not necessarily for all types of questions. The overall system's answer is then selected from the combined pool of candidates through a combination of merging and filtering techniques. For a reasonably detailed discussion of our QA system architecture we refer to [2]. A general overview of the system is given in Figure 1.

Question Processing. The first stage of processing, *question processing*, is common to all the streams. Each of the 200 questions is tagged, parsed, and assigned a question class based on our question classification module. Finally, the expected answer type is determined. See Section 3.2 for more details about question processing for the XQuesta stream, in particular.

There are seven streams in our system this year, four of which use the CLEF corpus to answer questions and three of which use external sources of information. We now provide a brief description of these seven streams. Note that except for the table stream and XQuesta, which we discuss in Sections 2.2 and 3 respectively, these streams remain unchanged from our system for last year's evaluation [2].

Streams that consult the Dutch CLEF corpus. Four streams generate candidate answers from the Dutch CLEF corpus: *Table Lookup*, *Pattern Match*, *Ngrams*, and *XQuesta*. The *XQuesta* stream, which is based on the idea of QA-as-XML-retrieval, is completely new for this year's system; for more details about it, see Section 3.

The *Table Lookup* stream uses specialized knowledge bases constructed by preprocessing the collection, exploiting the fact that certain types of information (such as country capitals, abbreviations, and names of political leaders) tend to occur in the document collection in a small number of fixed patterns. When a question type indicates that the question might potentially have an answer in these tables, a lookup is performed in the appropriate knowledge base and answers which are found there are assigned high confidence. For a detailed overview of this stream, see [7]; for information about improvements made to this stream for this year, see Section 2.2.

In the *Pattern Match* stream, zero or more regular expressions are generated for each question according to its type and structure. These patterns, which indicate strings that are highly likely to contain the answer, are then matched against the entire document collection.

The *Ngram* stream, similar in spirit to [5], constructs a weighted list of queries for each question using a shallow reformulation process, similar to the Pattern Match stream. These queries are fed to a retrieval engine (we use the open-source Lucene, with a standard vector-space model [11]), and the top retrieved documents are used for harvesting word n-grams. The n-grams are ranked according to the weight of the query that generated them, their frequency, NE type, the proximity to the query keywords and more parameters; the top-ranking n-grams are considered candidate answers.

Streams that consult the Web. Quartz has two streams that attempt to locate answers on the web: *Ngram mining* and *Pattern Match*. For Web searches, we use Google, and n-grams are harvested from the returned *snippets*. Pattern matching, by contrast, is done against *full documents* returned by Google.

Wikipedia stream. This stream also uses an external corpus—the Dutch Wikipedia (`http://nl.wikipedia.org`), an open-content encyclopedia in Dutch (and other languages). However, since this corpus is much cleaner than news paper text, the stream operates in a different manner. First, the *focus* of the question is identified; this is usually the main named entity in the question. Then, this entity's encyclopedia entry is looked up; since Wikipedia is standardized to a large extent, this information has a template-like nature. Finally, using knowledge about the templates used in Wikipedia, information such as Date-of-death and First-name can easily be extracted.

After the streams have produced candidate answers, these answers must be processed in order to choose a single answer to return. Also, for those streams that do not extract answers directly from documents in the CLEF collection, documents in the corpus that support or justify their answers must be found. The modules that perform these tasks are described next.

Answer Justification. As some of our streams obtain candidate answers *outside* the Dutch CLEF corpus, and as answers need to be supported, or *justified*, by a document in the Dutch CLEF corpus, we need to find justification for externally found candidate answers. To this end we construct a query with keywords from a given question and candidate answer, and consider the top-ranking document for this query to be the justification, using an Okapi model, as this tends to do well on early high precision in our experience. Additionally, we use some retrieval heuristics such as marking the answer terms in the query as boolean (requiring them to appear in retrieved documents). Note: in addition to answers from the three streams that use outside data sources, answers from the corpus-based Ngram stream also need to be justified, since these answers are selected by the stream on the basis of support from multiple documents.

Filtering, Tiling, and Type Checking. We use a final answer selection module (similar to that described in [6]) with heuristic candidate answer filtering and merging, and with stream voting. To compensate for named entity errors made during answer extraction, our type checking module (see [14] for details) uses several geographical knowledge bases to remove candidates of incorrect type for location questions. Furthermore, we take advantage of the temporally restricted questions in this year's evaluation to re-rank candidate answers for such questions using temporal information; see Section 2.3 for more details.

2.2 Improvements to table stream

To the tables used in 2004, we added a table which contained definitions extracted (offline) with two rules: one to extract definitions from appositions and another to create definitions by combining proper nouns with preceding common nouns. This table was used in parallel with the existing roles table, which contained definitions only for people. The new table contained more than three times as many entries (611,077) as the existing one.

In contrast to earlier versions of the table module, all tables are now stored in SQL format and made available in a MySQL database. The type of an incoming question is converted to sets of tuples containing three elements: table, source field and target field. The table code will search in the source field of the specified table for a pattern and, in cases where a match is found, keep the contents of corresponding target field as a candidate answer. Ideally the search pattern would be computed by the question analysis module but currently we use separate code for this task.

The table fields only contain noun phrases that are present in the text. This means that they can be used for answering questions such as:

(1) *Wie is de president van Servië?*
Who is the President of Serbia?

because the phrase *president van Servië* can normally be found in the text. However, in general, this stream cannot be used for answering questions such as:

(2) *Wie was de president van Servië in 1999?*
Who was the President of Serbia in 1999?

because the modifier *in 1999* does not necessarily always follow the profession.

The table code contains backoff strategies (case insensitive vs. case sensitive, exact vs. inexact match) in case a search returns no matches.

2.3 Temporal restrictions

Twenty-six questions in this year's QA track are tagged as *temporally restricted*. As the name implies, such questions ask for information relevant to a particular time; the time in question may be given explicitly by a temporal expression (or *timex*), as in:

(3) *Q0094: Welke voetballer ontving "De Gouden Bal" in 1995?*
Which footballer won the European Footballer of the Year award in 1995?

or it may be given implicitly, with respect to another event, as in:

(4) *Q0160: Hoe oud was Nick Leeson toen hij tot gevangenisstraf werd veroordeeld?*
How old was Nick Leeson when he was sentenced to prison?

Furthermore, the temporal restriction may not be to a point in time but to a temporal interval, as in:

(5) *Q0156: Hoeveel medicinale produkten met tijgeringredienten gingen er tussen 1990 en 1992 over de toonbank?*
How many medicinal products containing tiger ingredients were sold between 1990 and 1992?

(6) *Q0008: Wie speelde de rol van Superman voordat hij verlamd werd?*
Who played the role of Superman before he was paralyzed?

In our runs this year, we took advantage of these temporal restrictions to re-rank candidate answers for temporally restricted questions. Because we had already built a module to identify and normalize timexes (see Section 3.1), we limited ourselves to explicit temporal restrictions (i.e., those signalled by timexes). Handling event-based restrictions would require identifying (and possibly temporally locating) events, which is a much more difficult problem.

For each temporally restricted question, the temporal re-ranker tries to identify the temporal restriction by looking for temporal prepositions (such as *in*, *op*, *tijdens*, *voor*, *na*) and timexes in the question. If it succeeds in identifying an explicit temporal restriction, it proceeds with re-ranking the candidate answers.

For each candidate answer, the justification document is retrieved, and sentences containing the answer and, if there is one, the question focus are extracted from it. If there are any timexes in these sentences, the re-ranker checks whether these timexes are compatible with the restriction. For each compatible timex, the score for the candidate answer is boosted; for each incompatible timex, the score is lowered. The timestamp of the document is also checked for compatibility with the restriction, and the score is adjusted accordingly. The logic involved in checking compatibility of a timex with a temporal restriction is relatively straightforward; the only complications come in handling times of differing granularities.

3 XQuesta

The XQuesta stream implements a QA-as-XML-retrieval approach [10, 13]. The target collection is automatically annotated with linguistic information offline. Then, incoming questions are converted to semistructured queries, and evaluation of these queries gives a ranked list of candidate answers. We describe the three stages in detail.

3.1 Offline annotation

We automatically processed the Dutch QA collection, identifying sentences and annotating them syntactically and semantically. For processing time reasons, we switched from the full parses used in 2004 to a Dutch part-of-speech tagger and a text chunker. Both were trained on the CGN corpus [15] in the same way as the tagger and chunker described in [17] but using the TnT tagger [3] rather than a general machine learner. Two extra layers of annotation were added: named entities, by the TnT tagger trained on CoNLL-2002 data [16], and extra temporal expressions, by a hand-coded rule-based system, because these were not included in the previously mentioned training data. The temporal expression annotation system not only identified temporal expressions but also, where possible, normalized them to a standard format (ISO 8601).

Here are some examples of the annotations. Example 7 shows a tokenized sentence in which each token has received an abbreviated Dutch part-of-speech tag.

(7) `<LID>`*de*`</LID>` `<ADJ>`*machtige*`</ADJ>` `<N>`*burgemeester*`</N>` `<VZ>`*van*`</VZ>`
`<N>`*Moskou*`</N>` `<LET>`*,*`</LET>` `<N>`*Joeri*`</N>` `<N>`*Loezjkov*`</N>` `<LET>`*,*`</LET>`
`<WW>`*veroordeelt*`</WW>` `<VNW>`*dat*`</VNW>`
the powerful mayor of Moscow , Joeri Loezjkov , condemns that

In Example 8, the sentence is divided into syntactic chunks. The three most frequently occurring chunk types are noun phrases (NP), verb phrases (VP) and prepositional phrases (PP). Some tokens, nearly always the punctuation signs, are not part of a chunk. By definition, chunks cannot be embedded.

(8) `<NP>`*de machtige burgemeester*`</NP>` `<PP>`*van*`</PP>` `<NP>`*Moskou*`</NP>` *,*
`<NP>`*Joeri Loezjkov*`</NP>` *,* `<VP>`*veroordeelt*`</VP>` `<NP>`*dat*`</NP>`

Example 9 shows the named entities in the text. These are similar to text chunks: they can contain sequences of words and cannot be embedded. Four different named entity types may be identified: persons (PER), organizations (ORG), locations (LOC) and miscellaneous entities (MISC).

(9) *de machtige burgemeester van* `<NE type="LOC">`*Moskou*`</NE>` *,*
`<NE type="PER">`*Joeri Loezjkov*`</NE>` *, veroordeelt dat*

The next two examples show temporal expressions which have been both identified and normalized to ISO 8601 format. In Example 10, normalization is straightforward: the ISO 8601 value of the year 1947 is simply “1947.”

(10) *Luc Jouret werd in* `<TIMEX val="1947">`*1947*`</TIMEX>` *geboren*
Luc Jouret was born in 1947

Normalization is more complicated in Example 11; in order to determine that *donderdagmorgen* refers to 1994-10-06, the system uses the document timestamp (in this case, 1994-10-08) and some simple heuristics to compute the reference.

(11) *Ekeus verliet Irak* `<TIMEX val="1994-10-06">`*donderdagmorgen*`</TIMEX>`
Ekeus left Iraq Thursday morning

The four annotation layers of the collection were converted to an XML format and stored in separate XML files, to simplify maintenance. Whenever the XQuesta stream requested a document from the collection, all annotation were automatically merged to a single XML document providing full access to the extracted information. In order to produce well-formed XML mark-up after merging, we used a mix of inline and stand-off (with character offsets to refer to original collection text) annotation schemes.

3.2 Question analysis

The current question analysis module consists of two parts. The first part determines possible question classes, such as LOCATION for the question shown in Example (12).

(12) *Q0065: Uit welk land komt Diego Maradona?*
What country does Diego Maradona come from?

We use 31 different question types, some of which belong to a more general class: for example, DATE_BIRTH and DATE_DEATH describe dates of birth and death and are subtypes of the class DATE. The assignment of the classes is based on manually compiled patterns.

The second part of our question analysis module is new. Depending on the predicted question class, an expected answer type is assigned. Our new system design allows us to search for structural and semantic information. The expected answer types describe syntactic, lexical or surface requirements which have to be met by the possible answers. The restrictions are formulated in XPath queries which are used to extract specific information from our preprocessed documents. For instance, possible answers to questions such as Example (13) and (14) require a named entity as an answer (PERSON and TIMEX, respectively). Besides the named entity information, the answer to the question in Example (14) has to start with a number. The corresponding XPath queries are `NE[@type="PER"]` and `TIMEX[@val=~/^\d/]` respectively.

(13) *Q0149: Wie is de burgemeester van Moskou in 1994?*
Who was the mayor of Moskow in 1994?

(14) *Q0014: Wanneer is Luc Jouret geboren?*
When was Luc Jouret born?

Table 1 displays the rules for mapping the question classes to the expected answer types which were manually developed.

3.3 Extracting and ranking answers

As described in Section 3.2, incoming questions are automatically mapped to retrieval queries (simply question texts) and XPath expressions corresponding to types of expected answers.

Retrieval queries are used to locate relevant passages in the collection. For retrieval, we use nonoverlapping passages of at least 400 characters starting and ending at paragraph boundaries. Then, the question's XPath queries are evaluated on the top 20 retrieved passages, giving lists of XML elements corresponding to candidate answers. For example, for the question in Example 14 above, with the generated XPath query "`TIMEX[@val=~/^\d/]`", the value "*1947*" is extracted from the annotated text in Example 10.

Question class	**Restrictions on the type of answer**
ABBREVIATION	word in capital letters
AGE	numeric value, possible word: jarige
CAUSE_REASON	sentence
CITY_CAPITAL	LOC
COLOR	adjective
DATE_DEATH, DATE_BIRTH, DATE	TIMEX, digital number
DEFINITION_PERSON	sentence
DEFINITION	noun phrase or sentence
DISTANCE	numeric value
DISTINCTION	noun phrase or a sentence
EXPANSION	MISC or ORG, noun phrase
HEIGHT	numeric value
LANGUAGE	MISC
LENGTH	numeric value
LOCATION	LOC
MANNER	sentence
MONETARY_UNIT	MISC
NAME	named entity
NUMBER_PEOPLE	numeric value, noun phrase
NUMBER	numeric value
ORGANIZATION	ORG
PERSON	PER
SCORE, SIZE, SPEED, SUM_OF_MONEY	numeric value
SYNONYM_NAME	PER
TEMPERATURE, TIME_PERIOD	numeric value

Table 1: Overview of the mapping rules from question classes to answer types

The score of each candidate is calculated as the sum of retrieval scores of all passages containing the candidate. Furthermore, the scores are normalized using web hit counts, producing the final ranked list of XQuesta's answer candidates.

4 Runs

We submitted two Dutch monolingual runs. The run uams051nlnl used the full system with all streams and final answer selection. The run uams052nlnl, on top of this, used an additional stream: the XQuesta stream with paraphrased questions. As a simple way of paraphrasing questions, we double-translated questions (from Dutch to English and then back to Dutch) using Systran, an automatic MT system. Question paraphrases were only used for query formulation at the retrieval step; question analysis (identification of question types, expected answer types and corresponding XPath queries) was performed on the original questions. Our idea was to see whether simple paraphrasing of retrieval queries would help to find different relevant passages and lead to more correctly answered questions.

5 Results and analysis

Our question classifier assigned a question type to 187 questions. All classified questions also received an expected answer type. Eleven of the unclassified questions are *What/Which NP* questions, such as Example 15. The remaining two unclassified questions are of the form *Name an NP*.

(15) *Q0024: Welke ziekte kregen veel Amerikaanse soldaten na de Golfoorlog?*
Which disease did many American soldiers contract after the Gulf War?

Table 2 lists the assessment counts for the two University of Amsterdam question answering runs for CLEF-2005 monolingual Dutch (NLNL) task. The two runs had 13 different answers, but the assessors evaluated only two pairs differently. We were surprised about the large number of inexact answers. When we examined the inexact answers of the first run, we found that a disproportional number of these were generated for definition questions: 85% (only 30% of the questions ask

Run	**Right**	**Unsupp.**	**Inexact**	**Wrong**
uams051nlnl	88	5	28	79
uams052nlnl	88	5	29	78

Table 2: Assessment counts for the 200 answers in the two Amsterdam runs submitted for Dutch monolingual Question Answering (NLNL) in CLEF-2005. Thirteen answers of the two runs were different but only two of these were assessed differently.

for definitions). Almost half of the errors (13 out of 28) were caused by the same information extraction problem: determining where a noun phrase starts. Here is an example:

(16) *Q0094: Wat is Eyal?*
A: leider van de extreem-rechtse groep

Here the answer *extreme right group* would be correct as an answer to the question *What is Eyal?*. Unfortunately the system generated the answer *leader of the extreme right group*. This extraction problem also affected the answers for questions that provided a definition and asked for a name. We expect that this problem can be solved by a check of the semantic class of the question focus word and head noun of the answer, both in the answer extraction process and in answer postprocessing. Such a check would also have prevented seven of the 15 other incorrect answers of this group.

When we examined the assessments of the answers to the three different question types, we noticed that the proportion of correct answers was the same for definition questions (45%) and factoid questions (47%) but that temporally restricted questions seemed to cause problems (27% correct). Of the 18 incorrect answers in the latter group, four involved an answer which would have been correct in another time period (questions Q0078, Q0084, Q0092 and Q0195). If these questions had been answered correctly, the score for this category would have raised to an acceptable 46% (including the incorrectly assessed answer to question Q0149).

The temporal re-ranking module described in Section 2 did make a positive contribution, however. For the following two temporally restricted questions:

(17) *Q0007: Welke voetballer ontving "De Gouden Bal" in 1995?*
Which footballer won the European Footballer of the Year award in 1995?

(18) *Q0159: Wie won de Nobelprijs voor medicijnen in 1994?*
Who won the Nobel Prize for medicine in 1994?

the highest ranking candidate answer before the temporal re-ranking module was applied was incorrect (*Ruud Gullit* and *Martin Luther King*), but the application of the temporal re-ranking module boosted the correct answer (*Weah* and *Martin Rodbell*) to the top position. Additionally, the temporal re-ranking module never demoted a correct answer from the top position.

The answers to the temporally restricted questions also indicate the overall problems of the system. Of the remaining 14 incorrect answers, only five were of the expected answer category while nine were of a different category. In the 62 answers of factoid and definition questions which were judged to be wrong, the majority (58%) had an incorrect answer class. An extra answer postprocessing filter which compares the semantic category of the answer and the one expected by the question would prevent such mismatches.

Our system produced five answers which were judged to be unsupported. One of these was wrong and another was right. A third answer was probably combined from different answers and a link to a document containing a part rather than the whole answer was kept. The remaining two errors were probably also caused by a document link which should not have been kept but the reason for this is unknown.

This error analysis suggests three ways to improve our QA system. First, we should work on definition extraction so that both questions asking for definitions and questions requiring the resolution of definitions can be answered in a better way. Second, we should examine inheritance of document links in the answer tiling process to make sure that the associated module does not cause unnecessary unsupported answers. And third, and most importantly, we should improve answer filtering to make sure that the semantic class of the generated answer corresponds with the class required by the question.

6 Conclusions

The bulk of our efforts for the 2005 edition of the QA@CLEF track was aimed at implementing a "pure" QA-as-XML-retrieval stream, as part of our multi-stream question answering architecture; here, the target collection is automatically annotated with linguistic information at indexing time, incoming questions are converted to semistructured queries, and evaluation of these queries gives a ranked list of candidate answers. The overall system provides *wrong* answers for less than 40% of the questions. Our ongoing work is aimed at addressing the main sources of error discovered: definition extraction, inheritance of document links in answer tiling, and semantically informed answer filtering.

Acknowledgments

This research was supported by various grants from the Netherlands Organization for Scientific Research (NWO). Valentin Jijkoun, Karin Müller, and Maarten de Rijke were supported under project number 220-80-001. Erik Tjong Kim Sang and Maarten de Rijke were supported under project number 264-70-050. David Ahn and Maarten de Rijke were supported under project number 612.066.302. In addition, Maarten de Rijke was also supported by grants from NWO, under project numbers 017.001.190, 365-20-005, 612-13-001, 612.000.106, 612.000.207, and 612.069.006.

References

[1] D. Ahn, V. Jijkoun, G. Mishne, K. Müller, M. de Rijke, and S. Schlobach. Using wikipedia at the trec qa track. In E. Voorhees and L. Buckland, editors, *The Thirteenth Text Retrieval Conference (TREC 2004)*, Gaithersburg, Maryland, 2005.

[2] D. Ahn, V. Jijkoun, K. Müller, M. de Rijke, S. Schlobach, and G. Mishne. Making stone soup: Evaluating a recall-oriented multi-stream question answering stream for dutch. In C. Peters, P. Clough, G. Jones, J. Gonzalo, M. Kluck, and B. Magnini, editors, *Multilingual Information Access for Text, Speech and Images: Results of the Fifth CLEF Evaluation Campaign*, LNCS 3491. Springer Verlag, 2005.

[3] T. Brants. *TnT – A Statistical Part-Of-Speech tagger*. Saarland University, 2000.

[4] E. Brill, S. Dumais, and M. Banko. An analysis of the AskMSR question-answering system. In *Proc. Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 257–264, 2002.

[5] S. Dumais, M. Banko, E. Brill, J. Lin, and A. Ng. Web question answering: Is more always better? In P. Bennett, S. Dumais, and E. Horvitz, editors, *Proceedings of SIGIR'02*, pages 291–298, 2002.

[6] V. Jijkoun and M. de Rijke. Answer Selection in a Multi-Stream Open Domain Question Answering System. In S. McDonald and J. Tait, editors, *Proceedings 26th European Conference on Information Retrieval (ECIR'04)*,, volume 2997 of *LNCS*, pages 99–111. Springer, 2004.

[7] V. Jijkoun, G. Mishne, and M. de Rijke. Preprocessing Documents to Answer Dutch Questions. In *Proceedings of the 15th Belgian-Dutch Conference on Artificial In telligence (BNAIC'03)*, 2003.

[8] V. Jijkoun, G. Mishne, and M. de Rijke. How frogs built the Berlin Wall. In *Proceedings CLEF 2003*, LNCS. Springer, 2004.

[9] V. Jijkoun, G. Mishne, C. Monz, M. de Rijke, S. Schlobach, and O. Tsur. The University of Amsterdam at the TREC 2003 Question Answering Track. In *Proceedings TREC 2003*, pages 586–593, 2004.

[10] K. Litkowksi. Use of metadata for question answering and novelty tasks. In *Proceedings of the Twelfth Text REtrieval Conference (TREC 2003)*, 2004.

[11] Lucene. The Lucene search engine. URL: http://jakarta.apache.org/lucene/.

[12] D. Moldovan, M. Pasca, S. Harabagiu, and M. Surdeanu. Performance issues and error analysis in an open-domain question answering system. *ACM Transactions on Information Systems*, 21(2):133–154, 2003.

[13] P. Ogilvie. Retrieval using structure for question answering. In V. Mihajlovic and D. Hiemstra, editors, *Proceedings of the First Twente Data Management Workshop (TDM'04)*, pages 15–23, 2004.

[14] S. Schlobach, M. Olsthoorn, and M. de Rijke. Type Checking in Open-Domain Question Answering. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, 2004.

[15] I. Schuurman, M. Schouppe, H. Hoekstra, and T. van der Wouden. CGN, an Annotated Corpus of Spoken Dutch. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*. Budapest, Hungary, 2003.

[16] E. F. Tjong Kim Sang. Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Taipei, Taiwan, 2002.

[17] E. F. Tjong Kim Sang, W. Daelemans, and A. Höthker. Reduction of Dutch Sentences for Automatic Subtitling. In *Proceedings of CLIN-2003*, pages 109–123. University of Antwerp, Antwerp Papers in Linguistics, 111, 2004.