Integrating linguistic knowledge in passage retrieval for question answering

Jörg Tiedemann
Alfa Informatica, University of Groningen
Oude Kijk in 't Jatstraat 26
9712 EK Groningen, The Netherlands
j.tiedemann@rug.nl

Abstract

In this paper we investigate the use of linguistic knowledge in passage retrieval as part of an open-domain question answering system. We use annotation produced by a deep syntactic dependency parser for Dutch, Alpino, to extract various kinds of linguistic features and syntactic units to be included in a multi-layer index. Similar annotation is produced for natural language questions to be answered by the system. From this we extract query terms to be sent to the enriched retrieval index. We use a genetic algorithm to optimize the selection of features and syntactic units to be included in a query. This algorithm is also used to optimize further parameters such as keyword weights. The system is trained on questions from the competition on Dutch question answering within the Cross-Language Evaluation Forum (CLEF). We could show an improvement of about 15% in mean total reciprocal rank compared to traditional information retrieval using plain text keywords (including stemming and stop word removal).

1 Introduction

Improving information retrieval (IR) through natural language processing (NLP) has been the goal for many researchers. NLP techniques such as lemmatization and compound splitting have been used in several studies (Krovetz, 1993; Hollink et al., 2003). Linguistically motivated syntactic units such as noun phrases (Zhai, 1997), head-modifier pairs (Fagan, 1987; Strzalkowski et al., 1996) and subject-verb-object triples (Katz and Lin, 2003) have also been integrated in information retrieval. However, most of these studies resulted in only little success or even decreasing performance. It has been argued that NLP and especially deep syntactic analysis is still too brittle and ineffective (Katz and Lin, 2003).

Integrating NLP in information retrieval seems to be very hard because the task here is to match plain text keywords to natural language documents. In question answering (QA), however, the task is to match natural language questions to relevant answers within document collections. For this, we have to analyze the question in order to determine what kind of answer the user is expecting. Traditional information retrieval is used in QA systems to filter out relevant passages from the document collection which are then processed to extract possible answers. Hence, the performance of this *passage retrieval* component (especially in terms of recall) is crucial for the success of the entire system. NLP tools and linguistic resources are frequently used in QA systems, e.g. (Bernardi et al., 2003; Moldovan et al., 2002), although not very often for passage retrieval (some exceptions are (Strzalkowski et al., 1996; Katz and Lin, 2003; Neumann and Sacaleanu, 2004)).

Our goal is to utilize information that can be extracted from the analyzed question in order to match linguistic features and syntactic units in analyzed documents. The main research question is to find appropriate units and features that actually help to improve the retrieval component. Furthermore, we have to find an appropriate way of combining query terms to optimize IR performance. For this, we apply an iterative learning approach based on example questions annotated with their answers.

In the next section we will give a brief description of our question answering system with focus on the passage retrieval component. Thereafter we will discuss the query optimization algorithm followed by a section on experimental results. The final section contains our conclusions.

2 Question answering with dependency relations

Our Dutch question answering system, Joost (Bouma et al., 2005), consists of two streams: a table look-up strategy using off-line information extraction and an on-line strategy using passage retrieval and on-the-fly answer extraction. In both strategies we use syntactic information produced by a wide-coverage dependency parser for Dutch, Alpino (Bouma et al., 2001). In the off-line strategy we use syntactic patterns to extract information from unrestricted text to be stored in fact tables (Jijkoun et al., 2004). For the on-line strategy, we assume that there is a certain overlap between syntactic relations in the question and in passages containing the answers. Furthermore, we also use strategies for reasoning over dependency rules to capture semantic relationships that are expressed by different syntactic patterns (Bouma et al., 2005).

Our focus is set on open-domain question answering using data from the CLEF competition on Dutch QA. We have parsed the entire corpus provided by CLEF with about 4,000,000 sentences in about 190,000 documents. The dependency trees are stored in XML and are directly accessible from the QA system. Syntactic patterns for off-line information extraction are run on the entire corpus. For the on-line QA strategy we use traditional information retrieval to select relevant passages from the corpus to be processed by the answer extraction modules. This step is necessary to reduce the search space for the QA system to make it feasible to run on-line QA. As segmentation level we used paragraphs marked in the corpus (about 1.1 million).

Questions are parsed within the QA system using the same parser. Using their analysis, the system determines the question type and, hence, the expected answer type. According to the type, we try to find the answer first in the fact database (if an appropriate table exists) and then (as fallback) in the corpus using the on-line QA strategy.

2.1 Passage retrieval in Joost

Information retrieval is one of the bottle-necks in the on-line strategy of our QA system. The system relies on the passages retrieved by this component and fails if IR does not provide relevant documents. Traditional IR uses a bag-of-words approach using plain text keywords to be matched with word-vectors describing documents. The result is usually a ranked list of documents. Simple techniques such as stemming and stop word removal are used to improve the performance of such a system. This is also the base-line approach for passage retrieval in our QA system.

The passage retrieval component in Joost includes an interface to seven off-the shelf IR systems. One of the systems supported is Lucene from the Apache Jakarta project (Jakarta, 2004). Lucene is a widely-used open-source Java library with several extensions and useful features. This was the IR engine of our choice in the experiments described here. For the base-line we use standard settings and a public Dutch text analyzer for stemming and stop word removal. Now, the goal is to extend the base-line by incorporating linguistic information produced by the syntactic analyzer. Figure 1 shows a dependency tree produced for one of the sentences in the CLEF corpus. We like to include as much information from the parsed data as possible to find better matches between an analyzed question and passages that contain answers. From the parse trees, we extract various kinds of linguistic features and syntactic units to be stored in the index. Besides the dependency relations the parser also produces part-of-speech (POS) tags, named entity labels and linguistic root forms. It also recognizes compositional compounds and particle verbs. All this information might be useful for our passage retrieval component.

Lucene supports multiple index fields that can be filled with different types of data. This is a useful

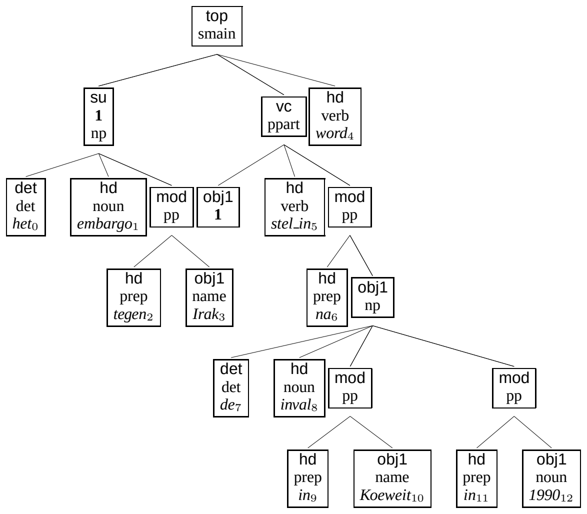

Figure 1: A dependency tree produced by Alpino: *Het embargo tegen Irak werd ingesteld na de inval in Koeweit in 1990.* (The embargo against Iraq has been declared after the invasion of Kuwait in 1990.)

feature since it allows one to store various kinds of information in different fields in the index. Henceforth, we will call these data fields *index layers* and, thus, the index will be called a *multi-layer index*. We distinguish between *token layers*, *type layers* and *annotation layers*. Token layers include one item per token in the corpus. Table 1 lists token layers defined in our index.

Table 1: Token layers

text	plain text tokens
root	root forms
RootPOS	root form + POS tag
RootHead	root form + head
RootRel	root form + relation name
RootRelHead	root form + relation + head

We included various combinations of features derived from the dependency trees to make it possible to test their impact on IR. Features are simply concatenated (using special delimiting symbols between the various parts) to create individual items in the layer. For example, the *RootHead* layer contains concatenated dependent-head bigrams taken from the dependency relations in the tree. Tokens in the *text* layer and in the *root* layer have been split at hyphens and underscores to split compositional compounds and particle verbs (Alpino adds underscores

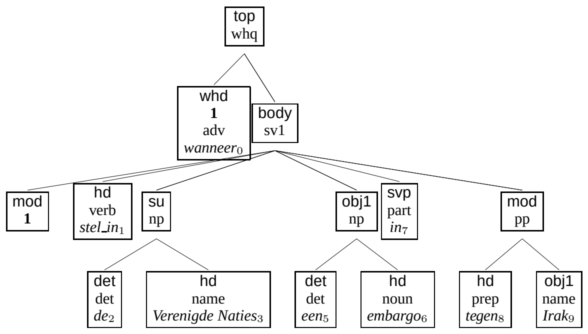

Figure 2: A dependency tree for a question: *Wanneer stelde de Verenigde Naties een embargo in tegen Irak?* (When did the United Nations declare the embargo against Iraq?)

between the compositional parts). Type layers include only specific types of tokens in the corpus, e.g. named entities or compounds (see table 2).

Table 2: Type layers

compound	compounds
ne	named entities
neLOC	location names
nePER	person names
neORG	organization names

Annotation layers include only the labels of (certain) token types. So far, we defined only one annotation layer for named entity labels. This layer may contain the items 'ORG', 'PER' or 'LOC' if such a named entity occurs in the text passage.

3 Query formulation

Questions are analyzed in the same way as sentences in documents. Hence, we can extract appropriate units from analyzed questions to be matched with the various layers in the index. For example, we can extract root-head word pairs to be matched with the RootHead layer. In this way, each layer can be queried using keywords of the same type. Furthermore, we can also use linguistic labels to restrict our query terms in several ways. For example, we can use part-of-speech labels to exclude keywords of a certain word class. We can also use the syntactic relation name to define query constraints. Each token layer can be restricted in this way (even if the feature used for the restriction is not part of the layer). For example, we can limit our set of root keywords to *nouns* even though part-of-speech labels are not part of the root layer. We can also combine constraints, for example, RootPOS keywords can be restricted to *nouns* that are in an *object* relation within the question.

Another feature of Lucene is the support of keyword weights. Keywords can be "boosted" using so-called "boost factors". Furthermore, keywords can also be marked as "required". These two features can be applied to all kinds of keywords (token layer, type layer, annotation layer keywords, and restricted keywords).

The following list summarizes possible keyword types in our passage retrieval component:

basic: a keyword in one of the index layers

restricted: *token-layer* keywords can be restricted to a certain word class and/or a certain relation type. We use only the following word class restrictions: *noun, name, adjective, verb*; and the following relation type restrictions: *direct object, modifier, apposition* and *subject*

weighted: keywords can be weighted using a *boost factor*

required: keywords can be marked as required

Query keywords from all types can be combined into a single query. We connect them in a disjunctive way which is the default operation in Lucene. The query engine provides ranked query results and, therefore, each disjunction may contribute to the ranking of the retrieved documents but does not harm the query if it does not produce any matching results. We may, for example, form a query with the following elements: (1) all plain *text* tokens; (2) named entities (*ne*) boosted with factor 2; (3) *RootHead* bigrams where the root is in an object relation; (4) *RootRel* keywords for all nouns. Applying these parameters to the question in figure 2 we get the following query:[1]

```
text:(Irak embargo Verenigde Naties stelde)
ne:(Irak^2 Verenigde_Naties^2)
RootHead:(Irak/tegen embargo/stel_in)
RootRel:(embargo/obj1)
```

Now, query terms from various keyword types may refer to the same index layer. For example, we may use weighted plain text keywords restricted to nouns together with unrestricted plain text keywords. To combine them we use a preference mechanism to keep queries simple and to avoid disjunctions with conflicting keyword parameters: (a) Restricted keyword types are more specific than basic keywords; (b) Keywords restricted in relation type *and* POS are more specific than keywords with only one restriction; (c) Relation type restrictions are more specific than POS restrictions. Using these rules we define that weights of more specific keywords overwrite weights of less specific ones. Furthermore, we define that the "required-marker" ('+') overwrites keyword weights. Using these definitions we would get the following query if we add two elements to the query from above: (5) plain text keywords in an object relation with boost factor 3 and (6) plain text keywords labeled as names marked as required.

```
text:(Irak^3 embargo^3 +Verenigde +Naties
stelde)
ne:(Irak^2 Verenigde_Naties^2)
RootHead:(Irak/tegen embargo/stel_in)
RootRel:(embargo/obj1)
```

Finally, we can also use the question type determined by question analysis in the retrieval component. The question type corresponds to the expected answer type, i.e. we expect an entity of that type in the relevant text passages. In some cases, the question type can be mapped to one of the named entity labels assigned by the parser, e.g. a *name question* is looking for names of persons (ne = PER), a question for a *capital* is looking for a location (ne = LOC) and a question for organizations is looking for the name of an organization (ne = ORG). Hence, we can add another keyword type, the expected answer type to be matched with named entity labels in the *ne* layer, cf. (Prager et al., 2000).

There are many possible combinations of restrictions even with the small set of POS labels and relation types listed above. However, many of them are useless because they cannot be instantiated. For example, an adjective cannot appear in subject relation to its head. For simplicity we limit ourselves to the following eight combined restrictions (POS + relation type): names + {direct object, modifier, apposition, subject} and nouns + {direct object, modifier, apposition, subject}. These can be applied to all token layers in the same way as the other restrictions using single constraints.

Altogether we have 109 different keyword types

[1]Note that stop words have been removed.

using the layers and the restrictions defined above. Now the question is to select appropriate keyword types among them with the optimal parameters (weights) to maximize retrieval performance. The following section describes the optimization procedure used to adjust query parameters.

4 Optimization of query parameters

In the previous sections we have seen the internal structure of the multi-layer index and the queries we use in our passage retrieval component. Now we have to address the question of how to select layers and restrict keywords to optimize the performance of the system according to the QA task. For this we employ an automatic optimization procedure that learns appropriate parameter settings from example data. We use annotated training material that is described in the next section. Thereafter, the optimization procedure is introduced.

4.1 CLEF questions and evaluation

We used results from the CLEF competition on Dutch QA from the years 2003 and 2004 for training and evaluation. They contain natural language questions annotated with their answers found in the CLEF corpus (answer strings and IDs of documents in which the answer was found). Most of the questions are factoid questions such as ’Hoeveel inwoners heeft Zweden?’ (How many inhabitants does Sweden have?). Altogether there are 631 questions with 851 answers.[2]

Standard measures for evaluating information retrieval results are precision and recall. However, for QA several other specialized measures have been proposed, e.g. mean reciprocal rank (MRR) (Vorhees, 1999), coverage and redundancy (Roberts and Gaizauskas, 2004). MRR accounts only for the first passage retrieved containing an answer and disregards the following passages. Coverage and redundancy on the other hand disregard the ranking completely and focus on the sets of passages retrieved. However, in our QA system, the IR score (on which the retrieval ranking is based) is one of the clues used by the answer identification modules. Therefore, we use the *mean of the total reciprocal ranks* (MTRR), cf. (Radev et al., 2002), to combine features of all three measures:

$$MTRR = \frac{1}{x} \sum_{i=1}^{x} \sum_{d \in A_i} \frac{1}{rank_{R_i}(d)}$$

A_i is the set of retrieved passages containing an answer to question number i (subset of R_i) and $rank_{R_i}(d)$ is the rank of document d in the list of retrieved passages R_i.

In our experiments we used the provided answer string rather than the document ID to judge if a retrieved passage was relevant or not. In this way, the IR engine may provide passages with correct answers from other documents than the ones marked in the test set. We do simple string matching between answer strings and words in the retrieved passages. Obviously, this introduces errors where the matching string does not correspond to a valid answer in the context. However, we believe that this does not influence the global evaluation figure significantly and therefore we use this approach as a reasonable compromise when doing automatic evaluation.

4.2 Learning query parameters

As discussed earlier, there is a large variety of possible keyword types that can be combined to query the multi-layer index. Furthermore, we have a number of parameters to be set when formulating a query, e.g. the keyword weights. Selecting the appropriate keywords and parameters is not straightforward. We like to carry out a systematic search for optimizing parameters rather than using our intuition. Here, we use the information retrieval engine as a black box with certain input parameters. We do not know how the ranking is done internally or how the output is influenced by parameter changes. However, we can inspect and evaluate the output of the system. Hence, we need an iterative approach for testing several settings to optimize query parameters. The output for each setting has to be evaluated according to a certain objective function. For this, we need an automatic procedure because we want to check many different settings in a batch run. The performance of the system can be measured in several ways, e.g. us-

[2]Each question may have multiple possible answers. We also added some obvious answers which were not in the original test set when encountering them in the corpus. For example, names and numbers can be spelled differently (*Kim Jong Il* vs. *Kim Jong-Il, Saoedi-Arabië* vs. *Saudi-Arabië, bijna vijftig jaar* vs. *bijna 50 jaar*)

ing the MTRR scores described in the previous section. We have chosen to use this measure and the annotated CLEF questions to evaluate the retrieval performance automatically.

We decided to use a simplified genetic algorithm to optimize query parameters. This algorithm is implemented as an iterative "trial-and-error beam search" through possible parameter settings. The optimization loop works as follows (using a sub-set of the CLEF questions):

1. Run initial queries (one keyword type per IR run) with default weights.
2. Produce a number of new settings by combining two previous ones (= *crossover*). For this, select two settings from an N-best list from the previous IR runs. Apply *mutation* operations (see next step) until the new settings are unique (among all settings we have tried so far).
3. Change some of the new settings at random (= *mutation*) using pre-defi ned mutation operations.
4. Run the queries using the new settings and evaluate the retrieval output (determine *fitness*).
5. Continue with 2 until some stop condition is satisfi ed.

This optimization algorithm is very simple but requires some additional parameters. First of all, we have to set the size of the *population*, i.e. the number of IR runs (*individuals*) to be kept for the next iteration. We decided to keep the population small with only 25 individuals. Then we have to decide how to evaluate fitness to rank retrieval results. This is done using the MTRR measure. *Natural selection* using these rankings is simplified to a top-N search without giving individuals with lower fitness values a chance to survive. This also means that we can update the population directly when a new IR run is finished. We also have to set a maximum number of new settings to be created. In our experiments we limit the process to a maximum of 50 settings that may be tried simultaneously. A new setting is created as soon as there is a spot available.

An important part of the algorithm is the combination of parameters. We simply merge the settings of two previous runs (*parents*) to produce a new setting (a *child*). That means that all keyword types (with their restrictions) from both parents are included in the child's setting. Parents are selected at random without any preference mechanism. We also use a very simple strategy in cases where both parents contain the same keyword type. In these cases we compute the arithmetic mean of the weight assigned to this type in the parents' settings (default weight is one). If the keyword type is marked as required in one of the parents, it will also be marked as required in the child's setting (which will overwrite the keyword weight if it is set in the other parent).

Another important principle in genetic optimization is mutation. It refers to a randomized modification of settings when new individuals are created. First, we apply mutation operations where new settings are not unique.[3] Secondly, mutation operations are applied with fixed probabilities to new settings.

In most genetic algorithms, settings are converted to genes consisting of bit strings. A mutation operation is then defined as flipping the value of one randomly chosen bit. In our approach, we do not use bit strings but define several mutation operations to modify parameters directly. The following operations have been defined:

- a new keyword type is added to new settings with a chance of 0.2
- a keyword type is removed from the settings with a chance of 0.1
- a keyword weight (boost factor) is modified by a random value between -5 and 5 with a chance of 0.2 (but only if the weight remains a positive value)
- a keyword type is marked as required with a chance of 0.01

All these parameters are intuitively chosen. We assigned rather high probabilities to the mutation operations to reduce the risk of local maximum traps. Note that there is no obvious condition for termination. In randomized approaches like this one the development of the fitness score is most likely not monotonic and therefore, it is hard to predict when we should stop the optimization process. However, we expect the scores to converge at some point and we may stop if a certain number of new settings does not improve the scores anymore.

[3]We require unique settings in our implementation because we want to avoid re-computation of fi tness values for settings that have been tried already. "Good" settings survive anyway using our top-N selection approach.

5 Experiments

We selected a random set of 420 questions from the CLEF data for training and used the remaining 150 questions for evaluation. We used the optimization algorithm with the settings as described above. IR was run in parallel on 3-7 Linux workstations on a local network. We retrieved a maximum of 20 passages per question. For each setting we computed the fitness scores for the training set and the evaluation set using MTRR. The top scores have been printed after each 10 runs and compared to the evaluation scores. Figure 3 shows a plot of the fitness score development throughout the optimization process in comparison with the evaluation scores.

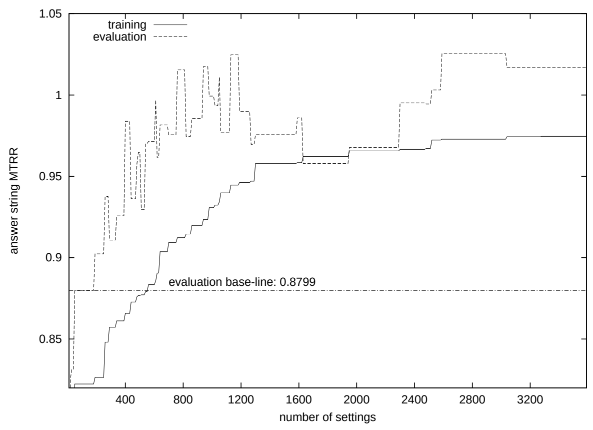

Figure 3: Parameter optimization

The base-line of 0.8799 refers to the retrieval result on evaluation data when using traditional IR with plain text keywords only (i.e. using the text layer, Dutch stemming and stop word removal). The base-line performance on training data is slightly worse with 0.8224 MTRR. After 1130 settings the MTRR scores increased to 0.9446 for training data and 1.0247 for evaluation data. Thereafter we can observe a surprising drop in evaluation scores to around 0.97 in MTRR. This might be due to overfitting although the drop seems to be rather radical. After that the curve of the evaluation scores goes back to about the same level as achieved before and the training curve seems to level out. The MTRR score after 3200 settings is at 1.0169 on evaluation data which is a statistically significant improvement of the baseline score (tested using the Wilcoxon matched-pairs signed-ranks test at $p < 0.01$). MTRR measured on document IDs and evaluation data did also increase from 0.5422 to 0.6215 which is statistically significant at p¡0.02. Coverage went up from 78.68% to 81.62% on evaluation data and the redundancy was improved from 3.824 to 4.272 (significance tests have not been carried out). Finally, the QA performance using Joost with only the IR based strategy was increased from 0.289 (using CLEF scores) to 0.331. This, however, is not statistically significant according to the Wilcoxon test and may be due to chance.

Table 3: Optimized parameters (3200 settings)

weighted keywords			required keywords	
layer	restriction	weight	layer	restriction
text		7.43	root	name
text	name	11.94		
text	adj	9.14	RootPOS	
text	mod	5.83	RootPOS	obj1
text	verb	4.33	RootPOS	noun-mod
text	noun-app	3.70		
root		4.45	RootRel	
root	noun-su	2.65	RootRel	app
root	name-mod	9.71	RootRel	noun-app
root	noun-obj1	0.09	RootRel	noun-mod
root	mod	0.81	RootRel	noun-obj1
root	verb	0.01		
RootHead	noun-app	7.65	RootRelHead	su
RootHead	noun-mod	5.24	RootRelHead	adj
RootHead	name-su	1	RootRelHead	name-app
RootRel	mod	4.45		
RootRel	name-app	2.17	Q-type	
RootRel	noun	2.49		
RootRelHead	obj1	1.60		
RootRelHead	name-su	1		
nePER		0.91		

Table 3 shows the features and weights selected in the training process. The largest weights are given to names in the text layer, to root forms of names in modifier relations and to plain text adjectives. Many keyword types use 'name' or 'noun' as POS restriction. A surprisingly large number of keyword types are marked as required. Some of them overlap with each other and are therefore redundant. For example, all RootPOS keywords are marked as required and therefore, the restrictions of RootPOS keywords are useless because they do not alter the query. However, in other cases overlapping keyword type definitions do influence the query. For example, RootRel keywords in general are marked as required. However, other type definitions replace some of them with weighted keywords, e.g., RootRel noun keywords. Finally, some of them may be changed back to required keywords, e.g., RootRel keywords of nouns in a modifier relation.

6 Conclusions

In this paper we describe an approach for integrating linguistic information derived from dependency analyses in passage retrieval for question answering. Our retrieval component uses a multi-layer index containing various combinations of linguistic features and syntactic units extracted from a fully analyzed corpus of unrestricted Dutch text. Natural language questions are parsed in the same way. Their analyses are used to build complex queries to our extended index. We demonstrated a genetic algorithm for optimizing query parameters to improve the retrieval performance. The system was trained on questions from the CLEF competition on open-domain question answering for Dutch which are annotated with corresponding answers in the corpus. We could show a significant improvement of about 15% in mean total reciprocal rank using extended queries with optimized parameters compared with the base-line of traditional information retrieval using plain text keywords.

References

Raffaella Bernardi, Valentin Jijkoun, Gilad Mishne, and Maarten de Rijke. 2003. Selectively using linguistic resources throughout the question answering pipeline. In *Proceedings of the 2nd CoLogNET-ElsNET Symposium*.

Gosse Bouma, Gertjan van Noord, and Robert Malouf. 2001. Alpino: Wide coverage computational analysis of Dutch. In *Computational Linguistics in the Netherlands CLIN, 2000*. Rodopi.

Gosse Bouma, Jori Mur, and Gertjan van Noord. 2005. Reasoning over dependency relations for QA. In *Knowledge and Reasoning for Answering Questions (KRAQ'05)*, IJCAI Workshop, Edinburgh, Scotland.

Joel L. Fagan. 1987. Automatic phrase indexing for document retrieval. In *SIGIR '87: Proceedings of the 10th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 91–101, New York, NY, USA. ACM Press.

Vera Hollink, Jaap Kamps, Christof Monz, and Maarten de Rijke. 2003. Monolingual document retrieval for European languages. *Information Retrieval*, (6).

Apache Jakarta. 2004. Apache Lucene - a high-performance, full-featured text search engine library. http://lucene.apache.org/java/docs/index.html.

Valentin Jijkoun, Jori Mur, and Maarten de Rijke. 2004. Information extraction for question answering: Improving recall through syntactic patterns. In *Proceedings of COLING-2004*.

Boris Katz and Jimmy Lin. 2003. Selectively using relations to improve precision in question answering. In *Proceedings of the EACL-2003 Workshop on Natural Language Processing for Question Answering*.

Robert Krovetz. 1993. Viewing morphology as an inference process,. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 191–203.

Dan Moldovan, Sanda Harabagiu, Roxana Girju, Paul Morarescu, Finley Lacatusu, Adrian Novischi, Adriana Badulescu, and Orest Bolohan. 2002. LCC tools for question answering. In *Proceedings of TREC-11*.

Günter Neumann and Bogdan Sacaleanu. 2004. Experiments on robust NL question interpretation and multi-layered document annotation for a cross-language question/answering system. In *Proceedings of the CLEF 2004 working notes of the QA@CLEF*, Bath.

John Prager, Eric Brown, Anni Cohen, Dragomir Radev, and Valerie Samn. 2000. Question-answering by predictive annotation. In *In Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Athens, Greece, July.

Dragomir R. Radev, Hong Qi, Harris Wu, and Weiguo Fan. 2002. Evaluating web-based question answering systems. In *Proceedings of LREC*, Las Palmas, Spain.

Ian Roberts and Robert Gaizauskas. 2004. Evaluating passage retrieval approaches for question answering. In *Proceedings of the 26th European Conference on Information Retrieval (ECIR)*, pages 72–84.

Tomek Strzalkowski, Louise Guthrie, Jussi Karlgren, Jim Leistensnider, Fang Lin, José Pérez-Carballo, Troy Straszheim, Jin Wang, and Jon Wilding. 1996. Natural language information retrieval: TREC-5 report.

Ellen M. Vorhees. 1999. The TREC-8 question answering track report. In *Proceedings of TREC-8*, pages 77–82.

Chengxiang Zhai. 1997. Fast statistical parsing of noun phrases for document indexing. In *Proceedings of the fi fth conference on Applied natural language processing*, pages 312–319, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.