# AnswerFinder at TREC 2006

**Diego Mollá** **Menno van Zaanen** **Luiz Pizzato**
Centre for Language Technology
Macquarie University
Sydney
Australia
{diego,menno,pizzato}@ics.mq.edu.au

## Abstract

This article describes the AnswerFinder question answering system and its participation in the TREC 2006 question answering competition. This year there have been several improvements in the AnswerFinder system, although most of them in the implementation sphere. The actual functionality used this year is almost exactly the same as last year, but many bugs are fixed and the efficiency of the system has improved much. This allows for more extensive parameter tuning. Here we will also present an error analysis of the current AnswerFinder system.

## 1 Introduction

This is the fourth year the AnswerFinder project is participating in the TREC question answering track. The underlying research question has been the same over the years: what shallow semantic representations are best suitable for representing the meaning of questions and sentences in the context of question answering?

The underlying framework of the AnswerFinder system has remained the same (or nearly the same). The system implements a step-wise reduction of data. This means that when AnswerFinder receives a question, the document selection phase selects a small number of relevant documents, out of which a smaller number of relevant sentences are selected during the sentence selection phase and finally within these sentences a selection of possible answers is selected. The best of these answers is then returned to the user.

So far, the focus has mainly been on the sentence selection and the answer extraction phases. In the sentence selection phase, several sentence selection methods have been implemented. Special attention has been paid to metrics that abstract over the exact word order used in sentences and questions. This is done by considering grammatical relations or more semantically oriented metrics.

In the answer extraction phase, a baseline approach based on the extraction of named entities from the expected answer type was combined with a rule-based method based on shallow semantics. We used logical graphs, and the answer extraction rules were learnt automatically from a small corpus of questions and answer sentences where the exact answers were manually annotated. Some initial experiments on this method were tried in the 2005 system. In the 2006 system, the method was reimplemented from scratch to increase speed.

For the 2005 competition, the system had been completely redesigned and rewritten (Mollá and van Zaanen, 2006). Based on our experiences from last year, we have decided to again rewrite large parts. Mainly third party tools that were integrated in the 2005 system introduced many problems, including memory leaks and random crashes. By isolating these problematic components and rewriting parts of these, the 2006 system is much more stable and much faster.

The rest of the article is structured as follows. We will first describe an overview of the AnswerFinder system in Sections 2 and 3, followed by a discussion on AnswerFinder in this year's competition. Next, we will do an error analysis of the system, followed by conclusions and future work.

## 2 AnswerFinder overview

AnswerFinder is a question answering system mainly developed for research in the use of shallow semantic representations of text. The underlying rationale behind this research is the aim for reducing the impact of paraphrases in text. There are different ways to say the same thing (or something very similar). When looking for answers to a question, it is necessary to reduce the impact of this.

The system consists of several phases, each reducing the amount of data that needs to be processed. This allows the system to perform more complex and computationally intensive algorithms further down the pipeline. Figure 1 gives a graphical representation of the system. Each of the phases will be described in more detail below.

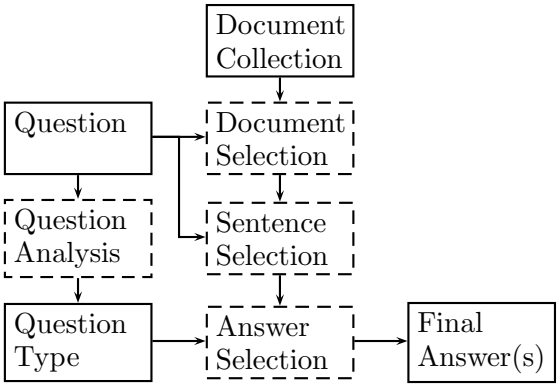


Figure 1: AnswerFinder system overview

**Document selection** When the system receives a question from the user, it will first search for documents relevant to the question in the full document collection. This is called the document selection phase. The aim in this phase is to only select relevant documents and thereby removing most of the document collection for further processing.

In previous competitions, we have always relied on the preselected documents provided by NIST. NIST provides the ranking of the top 1,000 documents retrieved by the PRISE search engine when using the target as the query. Last year, we selected the best 50 documents from the list of relevant documents.

This year, instead of having a list of documents *per topic*, as generated by NIST, we have experimented with a list of documents *per question*. The idea behind this is that the document selection is better tuned to the question. The list was generated using a document retrieval (DR) system built using the indexing and retrieval methods of Xapian[1] toolkit.

The list of relevant documents for each question is found by taking the words in the question as search keywords in the DR system. Additionally, since not every question contains the topic words, the queries for the DR stage were extended using the words of the topic.

[1] `http://www.xapian.org`

The introduction of the question words in the query may, in some cases, introduce undesirable information that could cause a query drift. To avoid this, we implemented another document selection system that performs a filtering process that only includes a document when it was found by the DR *and* was in the original NIST list.

**Sentence selection** Once the relevant documents are selected during the document selection phase, the sentence selection phase is started. The documents are segmented, resulting in a set of sentences. The size of this set of sentences is reduced by only selecting sentences that are considered relevant to the question.

Several sentence selection methods have been implemented. Each of the methods rely on a distance metric between a question and a sentence. The distance is computed for each of the sentences and based on that metric the best $n$ are selected.

The simplest metric is *word overlap*. The score of a sentence with respect to a question is the number of words that can be found in both. To reduce the impact of function words (such as "the" and "and"), we do not take these into account. The idea behind this metric is that a sentence should say something about one of the content words of the question. The answer will probably not be in the sentence if the sentence is about something else.

The next metric is called *grammatical relation overlap*. The underlying idea is the same as that of word overlap. However, instead of simply counting words in both question and sentence, the number of grammatical relations (Carroll et al., 1998) are counted. Grammatical relations are dependency relations computed by the Connexor Functional Dependency Parser (Tapanainen and Järvinen, 1997). In addition to having the same words in question and sentence, they must also be in a same relation to each other.

Additionally, previous versions of AnswerFinder also implemented *logic form overlap*. Logical forms can be generated from the grammatical relations that are output by Connexor. They are shallow semantic representations, which abstract away even more over the grammatical realisation of the actual sentence and question. The advantage of using this representation is that the impact of paraphrasing is reduced. However, the computation of the overlap was done using

SICStus Prolog, which seemed to introduce memory leaks.

Instead of using the flat logical forms, in the current version we use a different representation of the same information. Last year, we introduced *logical graph overlap*. The implementation used then was slow and this year, the algorithm that computes the overlap between the graphs has been completely redesigned. The format of the logical graphs and the method used is presented in Section 3.

**Question analysis** Once the relevant sentences are selected, the actual answer selection process starts. The extraction of possible answers is discussed in the next section. After the answers are extracted, an answer selection step is performed. This selects the actual answer (one or more) that will be returned to the user. During the answer selection step, information from the question is used. The question is analysed during the question analysis phase for this information.

The question analysis phase looks for the kind of answer the question is asking for. The question is matched against a list of regular expressions. Depending on which regular expression matches, it indicates the question type. For example, if the regular expression "Where" matches the question, the answer should be a location.

**Answer selection** From the small set of relevant sentences, possible answers are extracted. Again, several answer extraction methods have been implemented. At the moment, only the logical graph overlap sentence selection method finds possible answers while computing the similarity score of a sentence with respect to a question (the previous AnswerFinder system also used the logical form overlap, which found possible answers as well).

In addition to the answers found by the sentence selection method, we find named entities in sentences and keep these as possible answers. We are currently investigating the requirements for the used named entity recogniser (Mollá et al., 2006). However, for this competition, we used GATE's ANNIE named entity recogniser[2], like in previous competitions.

Finally, all found possible answers are considered. The scores associated with the answers are computed. The logical graph overlap metric assigns a score to the answer and answers found as named entities receive a score of one.

The answers are now matched against the question type. The answer has to be of a type that fits with what the question is asking for. Within these answers, the one with the highest score is returned. For non-factoid questions, all answers that match the correct question type are returned. If no such answer exists, the question type restriction is removed and all possible answers are considered.

## 3 Logical Graphs

The main goal of using logical graphs is to provide a graph representation of the shallow semantics of the sentences. By using a graph representation it becomes possible to use graph-based algorithms to compute the similarity between sentences and to find the answer to a question (Mollá, 2006). A logical graph (LG) is a directed, bipartite graph with two types of vertices, concepts and relations.

**Concepts** Examples of concepts are objects *dog*, *table*, events and states *run*, *love*, and properties *red*, *quick*.

**Relations** Relations act as links between concepts. To facilitate the production of the LGs we have decided to use relation labels that represent verb argument positions. Thus, the relation *1* indicates the link to the first argument of a verb (that is, what is usually a subject). The relation *2* indicates the link to the second argument of a verb (usually the direct object), and so forth. Furthermore, relations introduced by prepositions are labelled with the prepositions themselves. Our relations are therefore close to the syntactic structure.

By using graphs, the task of finding the common elements between a question and a text sentence is reduced to the task of finding the maximum common subgraph (MCS) between the corresponding LGs. The similarity between two sentences can be computed as the size of the resulting MCS.

To find the exact answer we use the MCS to automatically learn question-answering rules. The method is basically the same as described in our entry to TREC 2005 and is presented in Figure 2. Given a question with LG $q$, an answer candidate sentence with LG $s$ and the actual answer with LG $a$, the rules learnt have the following structure:

[2] `http://gate.ac.uk/`

$R_p$ is the MCS of $q$ and $s$, that is, $MCS(q, s)$.

$R_e$ is the path between the projection of $R_p$ in $s$ and the actual answer $a$.

$R_a$ is the graph representation of the exact answer, that is, $a$.

```
FOR every question/answerSentence pair
  q = the graph of the question
  s = the graph of the answer sentence
  a = the graph of the exact answer
  FOR every overlap O between q and s
    FOR every path P between O and a
      Build a rule R of the form
        R_p = O
        R_e = P
        R_a = a
```

Figure 2: Learning of graph rules

The rules learnt this way are generalised so that they can be applied to unseen data. The process to generalise rules takes advantage of the two kinds of vertices. Basically, relation vertices represent names of relations and we considered these to be important in the rule. Consequently relation vertices are left unmodified in the generalised rule. Concept vertices are generalised by replacing them with generic variables, except for a specific set of "stop concepts" which are left unmodified. The list of stop concepts is very small:

> *and, or, not, nor, if, otherwise, have, be, become, do, make*

The only difference between the system presented in TREC 2006 and that of TREC 2005 is the implementation of the algorithm that computes the MCS, which is based on the algorithm developed by (Myaeng and López-López, 1992) and summarised in Figure 3. The basic approach of the algorithm is to reduce the size of the graphs in a preliminary stage by merging some of the vertices of the related association graph. The problem of finding the MCS of two graphs is NP-complete but, given that the graphs are relatively small, the algorithm is fast enough for the task of learning and applying the QA rules.

The system used in TREC 2006 applies the above method to logical graphs, but the methodology is independent of the actual graph formalism. We are exploring the use of other graph representations, such as syntactic dependencies.

1. Find the association graph $G_A$
2. For each vertex $(x, y)$ in $G_A$, find the set of incoming vertices $IN_{(x,y)}$ and outgoing vertices $OUT_{(x,y)}$ in the original graphs
3. Split $IN_{(x,y)}$ and $OUT_{(x,y)}$ into sets of compatible vertices
4. Propagate elements of $IN_{(x,y)}$ by exploring the value of $IN_{(z,w)}$ when $(x, y)$ is in $IN_{(z,w)}$ for active vertices of $G_A$; do similar treatment with $OUT_{(x,y)}$
5. Build a new association graph $G'_A$ by taking each set of $IN_{(x,y)}$ and $OUT_{(x,y)}$ as a vertex and the compatibility between vertices as an edge
6. An MCS is a clique of $G'_A$

Figure 3: Sketch of the MCS algorithm by (Myaeng and López-López, 1992)

## 4 TREC 2006

This year, our aim was to consolidate the implementation we are current working on. The results generated for last year's competition required a lot of manual restarting of the system due to memory leaks and bugs. This year we concentrated on removing the components that created these problems. The current system is much faster and much more stable.

Unfortunately, while running the final test experiments before submitting the actual results for this year, we found a bug in the system in the final answer selection phase. Due to time constraints, we could not fix this bug in time, which meant that the submitted results were generated by the system with this bug.

It turns out that when a sentence contains several possible answers, all of these answers are counted as many times as there are answers in that sentence. This means that the system has a high preference for answers that occur in sentences that also contain many other (possibly incorrect) answers. In practice, even fewer sentences were considered than were returned by the sentence selection phase.

We will now describe the parameters of the system that were used to generate the results submitted to the TREC QA track and investigate how the results would be different when the bug had been fixed. Next, we will take a look at the error analysis of the system on the TREC 2005 data.

### 4.1 Parameters

We have submitted three runs: lf10w10g5, lf10w10g5l5, and lf10w20g5l5. These names already indicate the most important settings of the system. The run tags can be divided into different components. The first part of the run "lf" indicates that we have used our own document selection method. The "l" stands for Luiz (Pizzato), who implemented the information retrieval part and the "f" stands for filtered. His system filters the preselected documents provided by NIST.

The rest of the tags are composed of sentence selection methods. The "w" stands for word overlap sentence selection and the number indicates the number of selected sentences based on that metric. Similarly, "g" stands for grammatical relation overlap and "l" is the logical graph overlap sentence selection method.

Finding possible answers is done using logical graph overlap (when that sentence selection method is used) and by applying the ANNIE named entity recogniser to the remaining sentences after sentence selection. The best answer is then selected as described above.

### 4.2 Results

AnswerFinder currently concentrates on answering factoid questions and therefore, we will look at the results of these kinds of questions. List type questions are answered by returning all answers that are found in the sentences that are selected and match the question type. "Other" type questions are found by treating "What is ⟨topic⟩?" as a list question. We do not expect these types of questions to return very useful answers.

Even when considering only factoid questions, the results of the submitted runs are very disappointing. Accuracies of 0.072, 0.042, and 0.040 are reported for runs lf10w10g5, lf10w10g5l5, and lf10w20g5l5, respectively.

To be able to compare the submitted results to the results of the system with the bug fixed (and with the same settings), we have used Ken Litkowski's patterns to compute the results of the runs. The results are shown in Table 1. The "submitted" part in the table contains the results we submitted to the competition, while "fixed" are the results with the bug fixed.

The parameters used in the submitted runs have been selected by performing a simple parameter search by hand on the 2005 question answering track data. Due to time constraints, only low values of the parameters have been tried. Although one would expect that a larger number of selected sentences would perform better, initial parameter tuning indicated that this is not the case. In the next section, we will investigate this further.

Table 1: Results according to Ken Litkowski's patterns on TREC2006 data

| | submitted | | fixed | |
|---|---|---|---|---|
| | correct | exact | correct | exact |
| lf10w10g5 | 8.5% | 7.8% | 9.8% | 7.9% |
| lf10w10g5l5 | 5.4% | 4.1% | 6.8% | 5.3% |
| lf10w20g5l5 | 4.9% | 3.9% | 7.0% | 5.7% |

The fixed results are actually more like what we would expect. Fixing the bug increased the results. However, the systems with logical graphs are clearly outperformed by the one without. The main reason for that is the way answers are combined. Logical graphs can generate answers with much higher scores than those generated from named entities. This indicates that a better method for consolidating the scores of the answers needs to be designed. Finally, as expected, the system that selects the best sentences from 20 sentences selected based on word overlap outperforms that which is based on 10 sentences.

You have to keep in mind that we did our parameter tuning beforehand and we used the system with the bug in it. We cannot expect the parameters to be optimal. Further parameter tuning may improve the results greatly with the current system. In fact, one would expect that returning more ($> 20$) sentences after sentence selection may result in higher scores.

## 5 Error analysis

In this section, we will take a closer look at where the system looses the correct answers. We will investigate all the phases in turn, starting with the document selection phase, followed by the sentence selection phases, and finally we quickly look at the answer selection phase.

We can evaluate the results of the system after each phase by taking the full output of that phase as an answer. For example, after document selection, we concatenate the contents of all selected documents and pretend that that is the answer. We can then compute the score using Ken Litkowski's patterns (as substring

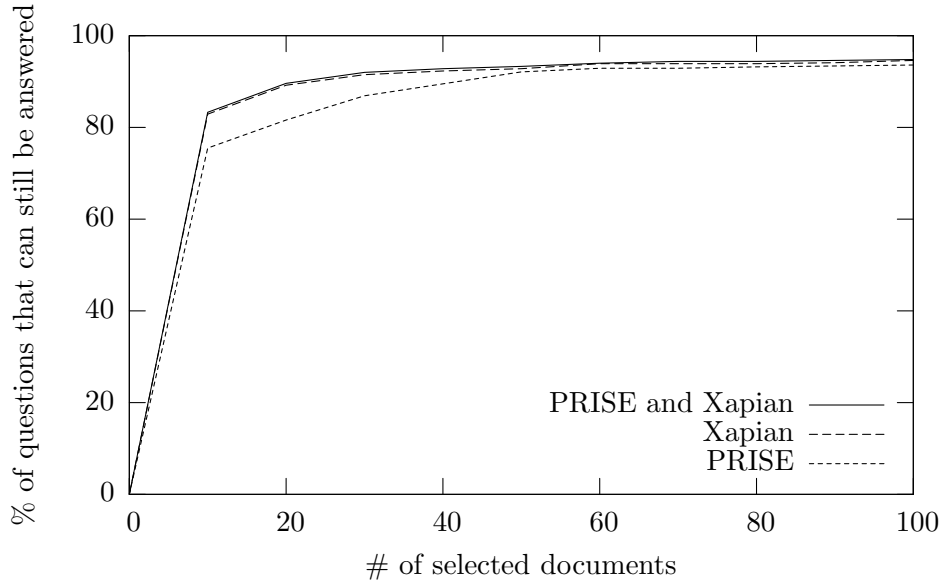


Figure 4: Results after document selection

matching) and take that as the upper bound of the percentage of questions that can still be answered.

After each phase, we expect the score to go down. The aim is to reduce the amount of data as much as possible on one hand and keep the score as high as possible on the other hand. In other words, we want to reduce the data that does not contain the answer.

### 5.1 Document selection

The first phase in AnswerFinder is the document selection phase. From the 1,033,461 documents contained in the Aquaint corpus, only a small selection can be handled in the next phases due to time and memory constraints.

In Figure 4, we show the percentage of questions that can still be answered after document selection with different selection methods. PRISE is the system provided by NIST, whereas Xapian is the system developed for AnswerFinder. The combination of PRISE and Xapian is the list of documents found by the Xapian system that are also found in the PRISE list. The last system is used the current AnswerFinder system.

As could be expected, increasing the number of documents for further processing will increase the upper bound of the number of questions that can be answered. The combined Xapian and PRISE systems outperform the PRISE system by itself. However, after about 50 documents, all upper bounds do not increase much anymore, whereas the amount of data that needs to be considered in the next phase does increase.

### 5.2 Sentence selection

The first two curves of Figure 5 indicate the percentage of questions that can still be answered against the number of sentences selected using word overlap. Here, we see that overall, we loose several percent of answerable questions with respect to document selection. This is obvious, since we remove a lot of sentences from the set of sentences contained in the selected documents.

Selecting from sentences from 50 documents generates better results than selecting from 10 documents. However, this difference is relatively small when only a small number of sentences are selected. This difference can perhaps be explained from the fact that the maximum score when selecting 10 documents is just over 80% whereas after selecting 50 documents still over 90% possible questions can be answered.

The third curve of Figure 5 indicates what happens when we select 50 documents, from these select 100 sentences using word overlap and from these 100 sentences, select sentences based on grammatical relation overlap. It is interesting to see that selecting 100 sentences first based on word overlap and then apply a new selection based on grammatical relation overlap is often better than directly reducing the number of sentences based on word overlap only. This shows the need for multi-stage sentence selec-

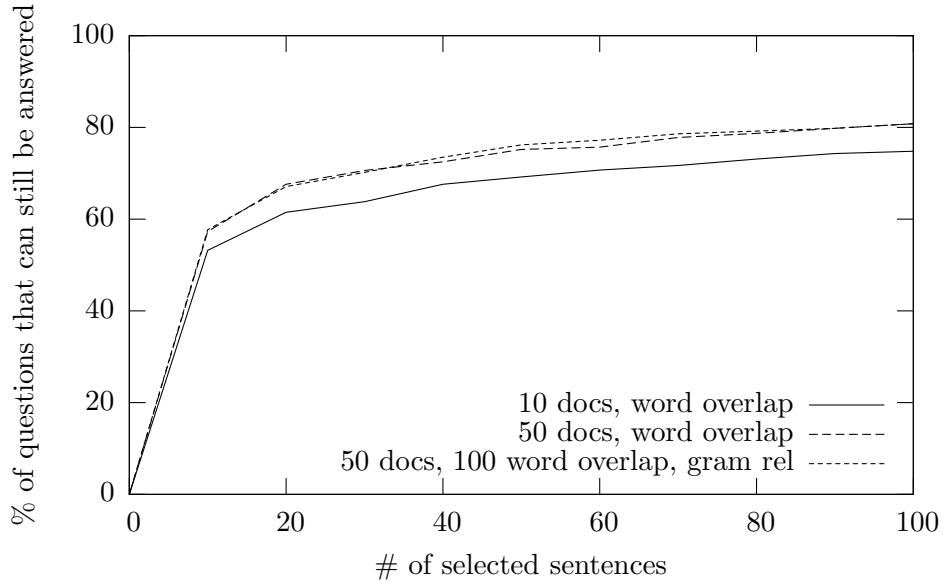


Figure 5: Results after sentence selection

tion. Similar effects occur with other settings of the number of sentences for each sentence selection method. Further work needs to go into this aspect of sentence selection.

### 5.3 Answer selection

Once the sentence selection phase has been done, the actual answers are found. Named entities are found in the relevant sentences and if logical graph sentence selection was used, the found answers in these are taken into account as well.

Table 2 illustrates the results of this phase. The first column contains the percentage of questions that can still be answered after sentence selection. The next column has the percentage of questions that can be answered using the named entities and answers found by the logical graphs. The third column gives the results when the question type restriction is applied and the final column gives the actual system output.

The results of the experiments including the logical graphs show some interesting aspects of the system. The first two runs perform similarly up to the point where question information is used to select answers. This is interesting. Apparently, the logical graphs do not add many more new answers. However, the scoring of the answers found by the logical graphs is typically higher than that of the named entities. The fact that the full system results are quite similar indicates that the answers found by logical graphs are still quite good. A better integration of the scores of the named entities and logical graphs could possibly further increase the results.

Interestingly, selecting the best 5 sentences from 20 sentences selected based on word overlap gives worse results. Apparently selecting sentences using grammatical relations should only be performed on relatively small collections. The rest of the third system simply percolates the slightly lower results.

From the final results, we see that in this case, we lost about half of the answers (that were still possible after document selection) to sentence selection. Finding possible answers seems to work quite well, although we still lost over 10%. The question type hardly lost us answerable questions, but the final selection should be improved.

## 6 Conclusions and future work

The current implementation of AnswerFinder is efficient and stable. All of the phases have different methods, which can be used in combination with each other.

The interaction between the algorithms in the different phases needs to be investigated further. In this article, we have included an error analysis, which indicates that choosing the parameters for each of the phases is not trivial. The choice for parameters is driven by two conflicting goals: reducing the amount of data and keeping the number of questions that can be answered as high as possible. We need to measure

Table 2: Results from sentence selection to final system output on TREC2005 data

| | sentence selection | possible answers | with question type | full system |
|---|---|---|---|---|
| lf10w10g5 | 44.7% | 30.8% | 28.2% | 8.9% |
| lf10w10g5l5 | 44.7% | 30.8% | 28.2% | 8.2% |
| lf10w20g5l5 | 43.8% | 29.6% | 27.0% | 7.6% |

the impact of selecting parameters on these two different goals to come to the best setting for the whole system.

At the moment, selecting the exact answers from the set of possible answers is the phase where most answers are lost. This is to be expected as it is in a sense the most difficult phase. The final reduction to an exact answer has to be made. Whereas previous phases are allowed to include some spurious text for further processing, the final phase does not have this luxury. This is the phase that we will focus our work on for the next competition.

## Acknowledgements

This research is funded by the Australian Research Council, ARC Discovery Grant no. DP0450750.

## References

John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proc. LREC98*.

Diego Mollá and Menno van Zaanen. 2006. Answerfinder at TREC 2005. In Ellen M. Voorhees and Lori P. Buckland, editors, *Proc. TREC 2005*. NIST.

Diego Mollá, Menno van Zaanen, and Luiz A.S. Pizzato. 2006. Named entity recognition for question answering. In *Proceedings ALTW 2006*, page 8 pages.

Diego Mollá. 2006. Learning of graph-based question answering rules. In *Proc. HLT/NAACL 2006 Workshop on Graph Algorithms for Natural Language Processing*, pages 37–44.

Sung H. Myaeng and Aurelio López-López. 1992. Conceptual graph matching: a flexible algorithm and experiments. *Journal of Experimentation and Theoretical Artificial Intelligence*, 4:107–126.

Pasi Tapanainen and Timo Järvinen. 1997. A non-projective dependency parser. In *Proc. ANLP-97*. ACL.