Learning of Graph Rules for Question Answering

Diego MOLLA and **Menno VAN ZAANEN**
Centre for Language Technology, Macquarie University
Sydney,
Australia,
{diego,menno}@ics.mq.edu.au

Abstract

AnswerFinder is a framework for the development of question-answering systems. AnswerFinder is currently being used to test the applicability of graph representations for the detection and extraction of answers. In this paper we briefly describe AnswerFinder and introduce our method to learn graph patterns that link questions with their corresponding answers in arbitrary sentences. The method is based on the translation of the logical forms of questions and answer sentences into graphs, and the application of operations based on graph overlaps and the construction of paths within graphs. The method is general and can be applied to any graph-based representation of the contents of questions and answers.

1 Introduction

Text-based question answering (henceforth QA) is the process whereby an answer to an arbitrary question formulated in plain English is found by searching through unedited text documents and returned to the user. The current availability of increasingly large volumes of text for human consumption has prompted an intensive research in QA. A well-known forum for the evaluation of QA systems is the question-answering track of the Text REtrieval Conference[1] (Voorhees, 2001), where systems developed by some of the most active researchers in the area are compared within the context of a common task. In addition, QA technology is being deployed in practical applications. For example, several Web-based question-answering systems are currently available (e.g. START[2], AnswerBus[3]), and recently popular Web search engines have started incorporating automated question-answering techniques (e.g. Google[4], as for September 2005).

The development of successful QA technology requires solid foundations both in the areas of software engineering and natural language processing. The nature of text-based question answering requires the use of a wide range of techniques, some of which are described in (Hirschman and Gaizauskas, 2001; Voorhees, 2001). For example, traditional document retrieval techniques are typically used to preselect the documents or document fragments that may contain the answer to the question. In addition, information extraction techniques are commonly used to extract all the named entities in the question and the preselected text, on the ground that fact-based questions typically expect one of these named entities as an answer. To analyse the questions, techniques range from the use of regular expressions to the use of machine-learning techniques that classify the questions according to the type of the expected answer. Finally, to find the answer, techniques may vary from a bag of words comparison of keywords used in the question and the answer sentence, to the use of full parsers and logical proof tools such as OTTER[5]. Additional resources are typically used, notably the WordNet lexical resource.[6] Consequently, the most successful QA systems are complex pieces of engineering that require frequent development and testing, such as (Moldovan et al., 2003). An unwelcome side-effect of this is that much of the effort spent in developing a QA system is spent, not in the developing of QA methodologies, but in defining the optimal parameters of a system.

On the other hand, QA presents challenging theoretical issues. One of the most salient theoretical challenges is related to the problem of

[1] http://trec.nist.gov
[2] http://www.ai.mit.edu/projects/infolab/
[3] http://www.answerbus.com/index.shtml
[4] http://www.google.com
[5] http://www-unix.mcs.anl.gov/AR/otter/
[6] http://wordnet.princeton.edu/

paraphrasing. There are many ways of expressing the same piece of information. For example, the simple question *Where was Peter born?* can be similarly asked as:

1. *In what city was Peter born?*
2. *What is Peter's birthplace?*
3. *What is the birthplace of Peter?*
4. *Name Peter's birthplace*

Whereas it may not be difficult to manually devise rules that account for the most popular ways of rephrase a question, variations in the sentences containing the answer are much less predictable. A human would not have any problem to find the answer to the above questions in the following examples:

1. *Peter was born in Paris.*
2. *Paris is Peter's birthplace.*
3. *Paris, Peter's birthplace, is located in France.*
4. *Mrs Smith gave birth to Peter in Paris.*

However, a machine would need to have access to lexical, syntactic, and world knowledge information if it is to find the answer.

The above are simple constructed examples. Real text with much more complex examples abounds, but the examples above suffice to illustrate the problem encountered by any text-based question-answering system. For further details about the problem of paraphrasing within the context of QA, see (Rinaldi et al., 2003).

Some systems have attempted to systematically build rules that link questions with answer sentences. For example, (Soubbotin, 2001) used a complex hierarchy of rules on surface strings. Other systems, such as (Echihabi et al., 2004), use a method for the automatic learning of surface-level rules. Other systems, such as (Bouma et al., 2005), use hand-crafted rules based on syntactic information.

Our hypothesis is that the accuracy of question-answering systems would improve if these rules are based on linguistic features located at a deeper level. Furthermore, to handle the problem of paraphrasing, the rules must be automatically learnt based on a representative corpus of questions and answers. In this paper we present our current work for developing and testing this hypothesis. Our work is being integrated in the AnswerFinder QA system, which is briefly described in Section 2. Section 3 describes the Logical Graph notation that we use to represent the logical contents of questions and answer sentences. Section 4 presents the rules based on Logical Graphs, and how they are automatically learnt from a corpus of question/answer pairs. Section 5 shows the use of these rules to find the exact answer to a question, and Section 6 shows the results of our evaluations. Sections 7 and 8 point to related research and give the final conclusions, respectively.

2 AnswerFinder, a Framework for Question Answering

Our solution to the need to use software engineering techniques for the development of practical QA systems is AnswerFinder. Initially designed as a simple QA prototype, AnswerFinder is currently being redesigned to allow the rapid development and test of QA techniques. Its primary application is the TREC Question Answering track,[7] but we also envisage its use, directly or indirectly, in other evaluation frameworks such as CLEF,[8] DUC,[9] and the PASCAL Recognising Text Entailment challenge.[10] For this reason, AnswerFinder's architecture is flexible and configurable, allowing to plug and test various modules easily.

The design of the system is functional and object-oriented. Focusing on function (instead of data) makes it easier to replace functions of the system with others. The system is implemented in C++. C++ was selected because it is a high-level language on the one hand, but can also be used in a more low-level way. It interfaces well with C, which allows for easy integration of many external systems. Furthermore, the resulting executable is relatively fast.

AnswerFinder consists of two main components, the client and the server. The client can get information from the server about the algorithms and files/document collections it provides to clients. The client can also send information to the server requesting question(s) to be processed using specific algorithms and data collections. The server can be fully configured via XML. For example, if the client calls

[7] http://trec.nist.gov
[8] http://www.clef-campaign.org/
[9] http://www-nlpir.nist.gov/projects/duc/
[10] http://www.pascal-network.org/Challenges/RTE/

the server without any configuration information, the server replies with an XML document listing all the available services. The client can then call the server with an XML file containing all the configuration information.

The request the server receives from the client contains all information needed to process the question(s). It specifies the document collection and the algorithms that should be used. The server then runs the required services by creating algorithm objects. An algorithm defines the full question-answering process, and it may use sub-algorithms for specific phases (such as question classification, document preselection, etc). The sub-algorithms are designed so that they can be called by any algorithm. Thus, different ways of trying QA techniques can be easily implemented by defining new algorithms that call the specific sub-algorithms with specific parameters.

The following sub-algorithms are currently defined in AnswerFinder. They are classified by the question-answering phase in which they are used:

Document Selection. Sub-algorithms in this phase are used to preselect the candidate documents.

- **NIST Doc Selection:** This sub-algorithm returns the documents provided by NIST for the TREC QA task.

Question Classification. Sub-algorithms in this phase are used to analyse and classify the question.

- **Regexp Q Classification:** This is a set of regular expressions that determines the type of the expected answer according to a simple hierarchy of types.

Sentence Selection. Sub-algorithms in this phase are used to determine what sentences are likely to contain an answer. These sub-algorithms can be cascaded to provide the final ranking of sentences.

- **Word Overlap:** Count the number of words in common between the question and the answer sentence. This sub-algorithm allows the use of a list of stop words that are not considered in the overlap.
- **Grammatical Relations Overlap:** Count the number of grammatical relations in common. We used a subset of the grammatical relations defined by (Carroll et al., 1998).
- **Logical Form Rules:** Count the number of logical form predicates in common, after applying a set of logical form rules. The process is explained in (Mollá and Gardiner, 2004).
- **Logical Graph Rules:** Count the graph overlap between the question and the answer after applying graph transformation rules. This process is explained in the remainder of this paper.

Named Entity. sub-algorithms in this phase are used to detect all named entities in the text (person and organisation names, locations, etc).

- **LingPipe:** This sub-algorithm uses the Alias-i LingPipe named entity recogniser.[11]

3 Logical Graphs

We are developing a graph notation for the expression of the logical contents of questions and answer sentences. Our Logical Graphs are inspired on Conceptual Graphs (Sowa, 1979), though our graphs do not attempt to encode the full semantics of a sentence. Instead, the focus of our Logical Graphs is on robustness and practicability.

Robustness. It should be possible to automatically produce the Logical Graph of any sentence, even of those sentences that are not fully grammatical. The importance of this feature becomes obvious once one looks at the quality of the English used in typical corpora used for QA.

Practicability. The Logical Graphs should be automatically constructed in relatively short run time. The operations with the graphs should be computable within relatively short time.

Like Sowa's Conceptual Graphs, our Logical Graphs are directed, bipartite graphs 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*. Concepts may be arranged in a network of word relations (such as ontologies), though our method does not yet exploit this possibility in full.

[11] http://alias-i.com/lingpipe/

Relations. Relations act as links between concepts. Traditional examples of relations are grammatical roles and prepositions. However, to facilitate the production of the Logical Graphs we have decided to use a labelling of relations which is relatively close to the syntactic level of linguistic information. For example, instead of using the usual thematic roles *agent*, *patient*, and so forth, we use syntactic roles *subject*, *object*, etc. For convenience, and to avoid entering into a debate about the possible names of the syntactic roles, we have decided to use numbers. 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.

Figure 1 shows various examples of Logical Graphs. The first example shows the use of a relation *1* to express the subject of the *go* event, and two relations, *to* and *by*, that represent two prepositions. The second example shows the use of lattice structures to represent complex entities (such as the ones formed when a conjunction is used). This use of lattices is inspired from the treatment of plurals and complex events (Link, 1983; Mollá, 1997). Finally, the third example shows the expression of clauses and control verbs. These examples only cover a few of the linguistic features but we hope they will suffice to show the expressive power of our Logical Graphs.

The Logical Graphs are constructed automatically from the output of the Conexor dependency-based parser (Tapanainen and Järvinen, 1997). The choice of the parser was arbitrary, and it would be easy to produce the same or similar graphs from the output of any dependency-based parser. It would be also possible to use the output of a constituency-based parser by applying well-known methods to convert from constituency structures to dependency structures like those described by Schneider (1998), or practical methods like the one described by Harabagiu et al. (2000).

4 Logical Graph Rules

The Logical Graph rules used by AnswerFinder are based on the concepts of *graph overlap* and *path* between two subgraphs in a graph.

The *graph overlap* between two sentences is the overlap of the Logical Graphs of the two sentences. A naïve definition of the overlap between two graphs would be the graph consisting of all the common concepts and relations. The actual definition of an overlap, however, must be made more complicated on the light of the existence of repeated vertex labels. The third example of Figure 1, for example, shows that the relations named *1* and *2* appear twice in the same graph. Concept labels can also be repeated in a graph if the sentence uses the same word to express two different concepts. For example, the sentence *John bought a book and Mary bought a magazine* describes two distinct events of buying.

Graph overlaps must therefore be defined on the basis of a *correspondence relation* so that each vertex (edge) of a graph correlates with one and only one vertex (edge) in the other graph (Montes-y-Gómez et al., 2001). Thus, there is a projection from the graph overlap to a subgraph of each of the original graphs, such that there is a correspondence from every vertex (or edge) of the graph overlap to a vertex (or edge) of the projected subgraphs. Figure 2 shows an example of two graph overlaps and their projections to subgraphs in the original graphs.

There could be several overlaps between two graphs. Of these, the most useful ones are the *maximal overlaps*, that is, the overlaps that are not subgraphs of any other overlaps. There could still exist several maximal overlaps between two graphs. For example, Figure 2 shows two different maximal overlaps between the Logical Graphs of two sentences.

A *path* between two subgraphs in a graph $\mathcal{G}$ is a subgraph of $\mathcal{G}$ that connects the two subgraphs. As is the case with graph overlaps, there may be several paths between two subgraphs, especially when the graphs have a high density of edges.

Each rule r will contain three components. For the sake of completion the components are listed here but their use will be described in detail in Section 5.

r_o An overlap between a question and its answer sentence. This overlap is used to determine when the rule should trigger.

r_p A path between the overlap and the actual answer in the answer sentence. This path is used to find the location of the exact answer.

r_a A graph representation of the exact answer.

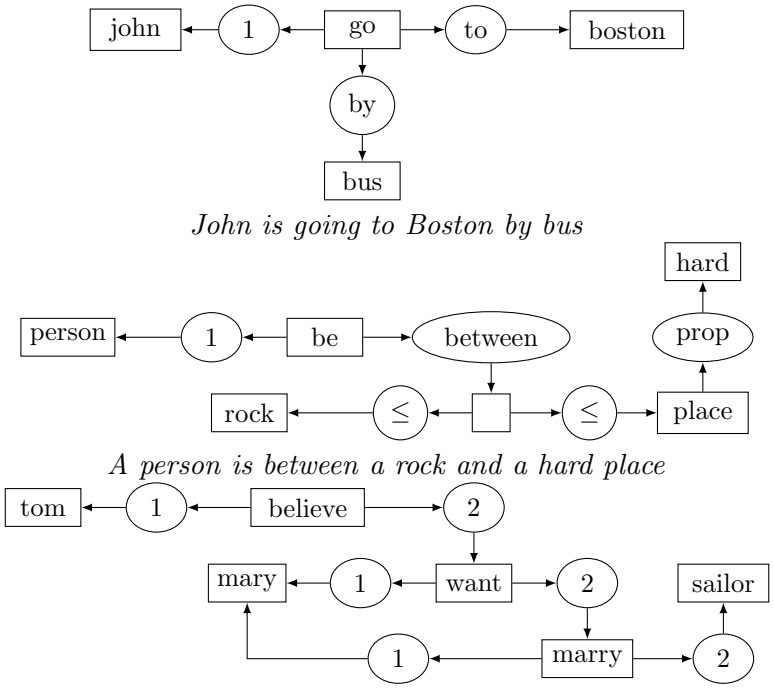

Figure 1: Examples of logical graphs

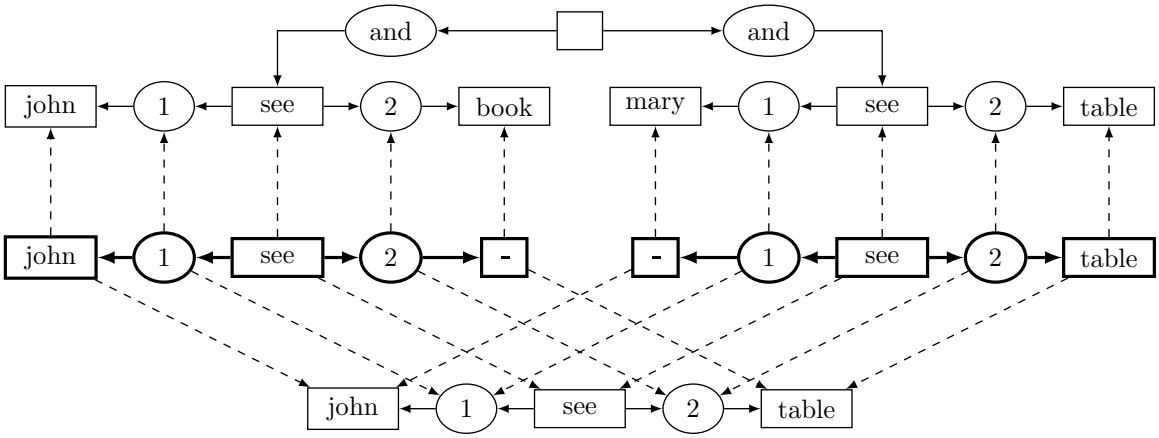

Figure 2: Graph overlaps of sentences *John saw a book and Mary saw a table* and *John saw a table*. The two overlaps are shown in thick lines. The dashed lines show the correspondence relation from the graph vertices of each overlap and the projected subgraphs in the original graphs (the correspondence relation from the edges is not shown to improve readability).

4.1 Learning of Logical Graph Rules

With the help of a training set of questions and sentences containing the answers, a set of Logical Graph rules can be learnt. Figure 3 shows an example of a rule learnt between two sentences. The graph notation has been simplified by replacing the relation vertices with labelled edges.

The algorithm for learning rules is fairly straightforward and is shown in Figure 4. Rules learnt with this algorithm are very specific to the question/answer pair. For example, the rule in Figure 3 would only trigger for questions about Peter and it would not trigger, say, for the question *Where was Mary born?*. The rule needs to be generalised. Our generalisation method is very simple: relations do not generalise (relations express syntactic or semantic relations and it is not advisable to over-generalise them), and concepts generalise to "_" (that is, concepts that would unify with anything). The generalisation process applies to every concept

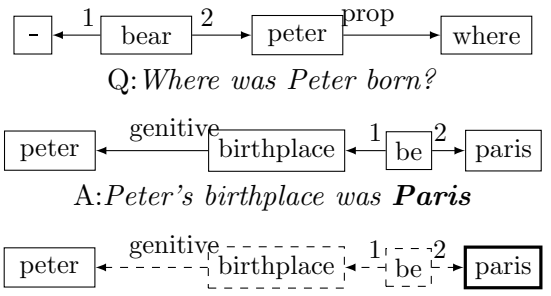

Figure 3: A logical graph rule

FOR every question/answerSentence pair
  $\mathcal{G}_q$ = the graph of the question
  $\mathcal{G}_s$ = the graph of the answer sentence
  $\mathcal{G}_a$ = the graph of the exact answer
  FOR every overlap $\mathcal{O}$ between $\mathcal{G}_q$ and $\mathcal{G}_s$
    FOR every path $\mathcal{P}$ between $\mathcal{O}$ and $\mathcal{G}_a$
      Build a rule $\mathcal{R}$ of the form
        $\mathcal{R}_o = \mathcal{O}$
        $\mathcal{R}_p = \mathcal{P}$
        $\mathcal{R}_a = \mathcal{G}_a$

Figure 4: Learning of graph rules

except those that belong to a specific list of "stop concepts" (in analogy to the idea of stop words in Information Retrieval). The current list of stop concepts is:

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

The resulting generalised rules may then overgeneralise and therefore they must be weighted according to their ability to detect the correct answer in the training corpus. The weight $\mathcal{W}(r)$ of a rule r is computed following the formula:

$$\mathcal{W}(r) = \frac{\text{\# correct answers found}}{\text{\# answers found}}$$

5 Graph-based Question Answering

Given a question q with graph Q and a sentence s with graph S, the process to find the answer iterates over all the rules. A rule r triggers if the overlap component of the rule r_o is a subgraph of Q (which can be easily determined by checking that $ovl(r_o, Q) = r_o$). When that happens, the graph of the question is expanded with the rule path r_p, producing a new graph Q_{r_p}. The resulting graph is more likely to produce a large overlap with an answer sentence similar to the one that generated the rule and, most importantly, the graph contains an indication of where the answer is located.

Once the graph of the question has been expanded with the path, one only needs to compute the overlap between this expanded graph and that of the answer sentence $ovl(Q_{r_p}, S)$. If the overlap retains part of the exact answer that was marked up by the graph rule, then we have found a possible answer.

The above method will cover simple cases, but it needs to be extended to cover two special cases that arise from the fact that the question/sentence pairs that generated the rule are likely to be different from the actual question and sentence being tested. First of all, a question may trigger several rules, and each rule may extract a different answer from the answer sentence. And second, it is possible that the overlap between the expanded graph and the sentence does not contain the complete answer but part of it. We will explain how to handle these two cases below.

To identify the correct answer among a set of possible answers it is necessary to establish a measure of "answerhood" so that the correct answer has a higher score than the score of other candidates. The rule weight gives an indication of the quality of the answer extracted, but we also need to keep in account the similarity (or otherwise) between the text originating the rule and the text being tested. Given that the graph of the question has been expanded with the path linking the question and the exact answer determined in the training corpus, then the size of the overlap between the expanded graph of the test question and the graph of the test answer can be used as an estimation. Thus, the measure of answerhood $\mathcal{A}(pa)$ of a possible answer pa is the product of the weight of the rule used r, and the size of the best overlap between the graph of the question sentence expanded with the rule path and the graph S of the answer sentence:

$$\mathcal{A}(a) = \mathcal{W}(r) \times size(ovl(Q_{r_p}, S))$$

The size of a graph is computed as the weighted sum of all concepts and relations in the path. The formula to determine the weight of each concept and relation is inspired on the use of the Inverse Document Frequency (IDF) measure used in Document Retrieval. The actual formula that we use is:

```
35.1
When did Jack Welch become chairman of General Electric?
Jack Welch took over GE in <answ>1981</answ>.
Welch became GE's chief executive in April <answ>1981</answ>.
Welch was named chief executive in <answ>1981</answ>.

35.4
How many people did he fire from GE?
He sold off underperforming divisions and fired about <answ>100,000</answ> people.
More than <answ>100,000</answ> GE jobs have been axed under Welch.
```

Figure 5: Extract of the training corpus

$$\mathcal{W}_i = \frac{1}{\log N} \log \frac{N}{n}$$

n = total number of sentences using the concept (or relation) i

N = total number of sentences

The formula includes the constant factor $1/\log N$ to ensure that the values range between 0 and 1.

6 Evaluation and Results

We have conducted an initial evaluation of the use of these rules within the task of question answering. For this evaluation we created a training and testing corpus based on the first 111 questions of the question-answering track of TREC 2004 (Voorhees, 2004). For each question, we applied Ken Litkowsky's patterns[12] to automatically extract sentences in the AQUAINT corpus containing possible answers. These sentences were checked manually, and only sentences containing the answer and a justification were selected. As a result we obtained about 560 question/answer pairs. The exact answers in the answer sentences of the training corpus were manually marked up to ensure a corpus without wrong answers. Figure 5 shows an extract of the training corpus.

The question/answer training corpus was split in 5 sets, and a 5-fold cross-validation was performed. Table 1 shows the results. In this table, the accuracy indicates the percentage of questions that are answered correctly. The MRR measure is as used in the TREC evaluations (Voorhees, 2001), and it measures the mean reciprocal rank for ranks from 1 to 5. For example, if the correct answer was ranked 3 (i.e. the system ranked two wrong answers higher than the correct answer), then the reciprocal rank is 1/3. If the correct answer was ranked beyond 5, then the reciprocal rank is 0. The MRR is the mean of the reciprocal ranks across all questions. Given that the results indicate an MRR with value higher than the accuracy, we can deduce that the system sometimes finds the correct answer but it does not assign it the top rank.

[12]Ken Litkowsky's patterns are available from the TREC website (http://trec.nist.gov).

	Accuracy	*MRR*
Test 1	25.76%	28.91%
Test 2	26.92%	31.09%
Test 3	9.21%	16.56%
Test 4	26.47%	29.78%
Test 5	18.82%	23.53%
Average	*21.44%*	*25.97%*

Table 1: Graph-based question answering results. The accuracy is the percentage of questions that are correctly answered. The MRR measure is as used in the TREC evaluations (see text).

Figure 6 shows the distribution of weights among the rules learnt (this is the sum of all the rules produced in all the five runs). To avoid computational overhead of handling rules with low weight we decided to set a threshold of 0.5. Any rules with weight below 0.5 were discarded. The figure indicates two clear regions, one with rules of weight lower than 0.6 and one with rules of weight above 0.9. It is probably desirable to set a threshold of 0.9 for a larger training corpus to ensure that only good quality rules are used. We refrained from doing so with our small corpus to avoid ending up with too few rules.

It is difficult to compare the above results with those of existing evaluations for various reasons. First of all, the system used in our

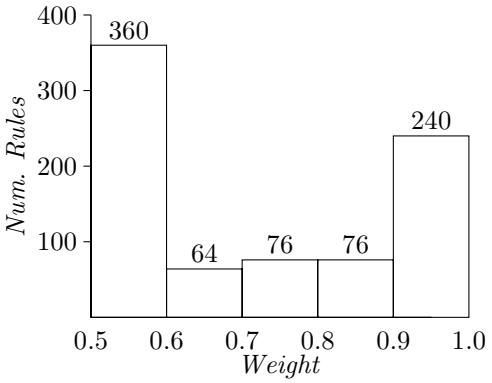

Figure 6: Distribution of weights among the rules learnt

evaluation did not have the usual modules of a full-blown QA system as described in Section 1. Second, the task is a simplification of a real QA task in that the answer is known to exist in the answer candidate. An area of further work is therefore to integrate this method into the AnswerFinder QA system and to evaluate the system in a task such as the QA track of TREC.

7 Related Work

There have been several attempts to automatically learn the correspondence between a question and an answer. For example, (Echihabi et al., 2004) describes three approaches to use question/answer rules, two of which use machine learning methods. In one approach, the system uses a methodical series of web searches containing a question phrase and the answer to collect a corpus of substrings linking the question phrase with the answer. The other machine-learning method described by Echichabi *et al.* uses techniques based on statistical machine translation to automatically learn the "translation" between a question and an answer.

A system that uses syntactic information in machine learning for QA has been recently published by Shen et al. (2005). This system is based on the extraction of dependency chains connecting a question word with an answer. The information is combined with other statistical features and fed to a Maximum Entropy model that ranks the answer candidates. The use of dependency chains in this system is similar in principle with our use of graph paths in that it provides a way to connect a question with its answer. We have not had time however to study this system in detail.

Another system that uses syntactic information to develop patterns is described by Bouma et al. (2005). Their system uses the output of a dependency parser combined with a set of equivalence rules between sets of dependency relations that paraphrase each other. In contrast with our method, however, the rules were developed manually and there were no indications in the paper about how to develop a method to automatically learn the rules. A method to discover similar sets of dependencies has been described by Lin and Pantel (2001), so in principle it is feasible to learn paraphrase rules and apply them to QA. However, the paraphrase rules described by these two systems do not attempt to connect a question with an answer, as we do.

The only system using logical-form rules that we are aware of is AnswerFinder at the time of participation in TREC 2004 (Mollá and Gardiner, 2004). The rules were based on AnswerFinder's minimal logical forms, and they were built manually. The system presented in our paper is a continuation of this research. Other than AnswerFinder, we are not aware of any QA system that attempts to learn rules based on logical forms.

There is some work on the use of conceptual graphs for information retrieval (Montes-y-Gómez et al., 2000; Mishne, 2004). However, we are not aware of any publication about the use of conceptual graphs (or any other form of graph representation) for question answering other than our own.

8 Conclusions and Further Work

We have introduced a methodology for the learning of graph patterns between questions and answers. Rules are learnt on the basis of two graph concepts: graph overlap, and paths between two subgraphs in a graph.

The techniques presented here use graph representations of the logical contents between questions and answer sentences. These techniques are being tested in AnswerFinder, a framework for the development of question-answering techniques that is easily configurable.

We believe that our method can generalise to any graph representation of questions and answer sentences. Further work will include the use of alternative graph representations, including the output of a dependency-based parser.

Finally, we plan to continue our evaluation of the method by integrating it into the AnswerFinder system and other QA systems to

fully assess its potential.

9 Acknowledgements

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

References

Gosse Bouma, Jori Mur, and Gertjan van Noord. 2005. Reasoning over dependency relations for qa. In *Proc. IJCAI-05 Workshop on Knowledge and Reasoning for Answering Questions*, pages 15–20.

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

Abdessamad Echihabi, Ulf Hermjakob, Eduard Hovy, Daniel Marcu, Eric Melz, and Deepak Ravichandran. 2004. How to select an answer string? In Tomek Strzalkowski and Sanda Harabagiu, editors, *Advances in Textual Question Answering*. Kluwer.

Sanda Harabagiu, Dan Moldovan, Marius Paşca, Rada Mihalcea, Mihai Surdeanu, Răzvan Bunescu, Roxana Gîrju, Vasile Rus, and Paul Morărescu. 2000. Falcon: Boosting knowledge for answer engines. In Ellen M. Voorhees and Donna K. Harman, editors, *Proc. TREC-9*, number 500-249 in NIST Special Publication, pages 479–488. NIST.

Lynette Hirschman and Rob Gaizauskas. 2001. Natural language question answering: The view from here. *Natural Language Engineering*, 7(4):275–300.

Dekang Lin and Patrick Pantel. 2001. Discovery of inference rules for question-answering. *Natural Language Engineering*, 7(4):343–360.

Godehard Link. 1983. The logical analysis of plurals and mass terms: a lattice-theoretical approach. In Rainer Bauerle, Christoph Schwarze, and Arnim von Stechov, editors, *Meaning, Use and Interpretation of Language*, pages 250–209. de Gruyter, Berlin.

Gilad Mishne. 2004. Source code retrieval using conceptual graphs. Master's thesis, University of Amsterdam.

Dan Moldovan, Marius Pas ca, Sanda Harabagiu, and Mihai Surdeanu. 2003. Performance issues and error analysis in an open-domain question answering system. *ACM Transactions on Information Systems*, 21(2):133–154.

Diego Mollá and Mary Gardiner. 2004. Answerfinder - question answering by combining lexical, syntactic and semantic information. In Ash Asudeh, Cécile Paris, and Stephen Wan, editors, *Proc. ALTW 2004*, pages 9–16, Sydney, Australia. Macquarie University.

Diego Mollá. 1997. *Aspectual Composition and Sentence Interpretation: a formal approach*. Ph.D. thesis, University of Edinburgh.

Manuel Montes-y-Gómez, Aurelio López-López, and Alexander Gelbukh. 2000. Information retrieval with conceptual graph matching. In *Proc. DEXA-2000*, number 1873 in Lecture Notes in Computer Science, pages 312–321. Springer-Verlag.

Manuel Montes-y-Gómez, Alexander Gelbukh, and Ricardo Baeza-Yates. 2001. Flexible comparison of conceptual graphs. In *Proc. DEXA-2001*, number 2113 in Lecture Notes in Computer Science, pages 102–111. Springer-Verlag.

Fabio Rinaldi, James Dowdall, Kaarel Kaljurand, Michael Hess, and Diego Mollá. 2003. Exploiting paraphrases in a question answering system. In *Proc. Workshop in Paraphrasing at ACL2003*, Sapporo, Japan.

Gerold Schneider. 1998. A linguistic comparison of constituency, dependency and link grammar. Master's thesis, University of Zurich. Unpublished.

Dan Shen, Geert-Jan M. Kruijff, and Dietrich Klakow. 2005. Exploring syntactic relation patterns for question answering. In Robert Dale, Kam-Fai Wong, Jian Su, and Oi Yee Kwong, editors, *Natural Language Processing IJCNLP 2005: Second International Joint Conference, Jeju Island, Korea, October 11-13, 2005. Proceedings*. Springer-Verlag.

M. M. Soubbotin. 2001. Patterns of potential answer expression as clues to the right answers. In Ellen M. Voorhees and Donna K. Harman, editors, *Proc. TREC 2001*, number 500-250 in NIST Special Publication. NIST.

John F. Sowa. 1979. Semantics of conceptual graphs. In *Proc. ACL 1979*, pages 39–44.

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

Ellen M. Voorhees. 2001. The TREC question answering track. *Natural Language Engineering*, 7(4):361–378.

Ellen M. Voorhees. 2004. Overview of the trec 2004 question answering track. In Ellen M. Voorhees and Lori P. Buckland, editors, *Proc. TREC 2004*, number 500-261 in NIST Special Publication. NIST.