Learning of Graph-based Question Answering Rules

Diego Mollá
Department of Computing
Macquarie University
Sydney 2109, Australia
diego@ics.mq.edu.au

Abstract

In this paper we present a graph-based approach to question answering. The method assumes a graph representation of question sentences and text sentences. Question answering rules are automatically learnt from a training corpus of questions and answer sentences with the answer annotated. The method is independent from the graph representation formalism chosen. A particular example is presented that uses a specific graph representation of the logical contents of sentences.

1 Introduction

Text-based question answering (QA) is the process of automatically finding the answers to arbitrary questions in plain English by searching collections of text files. Recently there has been intensive research in this area, fostered by evaluation-based conferences such as the Text REtrieval Conference (TREC) (Voorhees, 2001b), the Cross-Lingual Evaluation Forum (CLEF) (Vallin et al., 2005), and the NII-NACSIS Test Collection for Information Retrieval Systems workshops (NTCIR) (Kando, 2005). Current research focuses on factoid question answering, whereby the answer is a short string that indicates a fact, usually a named entity. An example of a factoid question is *Who won the 400m race in the 2000 Summer Olympic games?*, which has a short answer: *Cathy Freeman*.

There are various approaches to question answering. The focus of this paper is on **rule-based systems**. A rule could be, say, "if the question is of the form *Who is the <position> of <country>*" and a text sentence says *<position> of <country> Y* and Y consists of two capitalised words, then Y is the answer"). Such a rule was used by Soubbotin (2001), who developed a system who obtained the best accuracy in the 2001 Text REtrieval Conference (Voorhees, 2001a). The system developed by Soubbotin (2001) relied on the development of a large set of patterns of potential answer expressions, and the allocation of those patterns to types of questions. The patterns were developed by hand by examining the data.

Soubbotin (2001)'s work shows that a rule-based QA system can produce good results if the rule set is comprehensive enough. Unfortunately, if the system is ported to a new domain the set of rules needs to be ported as well. It has not been proven that rules like the ones developed by Soubbotin (2001), which were designed for the TREC QA task, can be ported to other domains. Furthermore, the process of producing the rules was presumably very labour intensive. Consequently, the cost of manually producing new rules for a specialised domain could become too expensive for some domains.

In this paper we present a method for the automatic learning of question answering rules by applying graph manipulation methods. The method relies on the representation of questions and answer sentences as graphs. Section 2 describes the general format of the graph-based QA rules and section 3 describes the method to learn the rules. The methods described on the above two sections are independent of the actual sentence representation formalism, as long as the representation is a graph. Section 4 presents a specific application using logical graphs. Finally, sections 5 and 6 focus on related research and final conclusions, respectively.

2 Question Answering Rules

In one form or another, a question answering rule must contain the following information:

1. a pattern that matches the question;
2. a pattern that matches the corresponding answer sentence; and
3. a pointer to the answer in the answer sentence

The patterns in our rules are expressed as graphs with vertices containing variables. A vertex with a variable can unify with a subgraph. For example, Figure 1 shows two graphs and a pattern that matches both graphs.

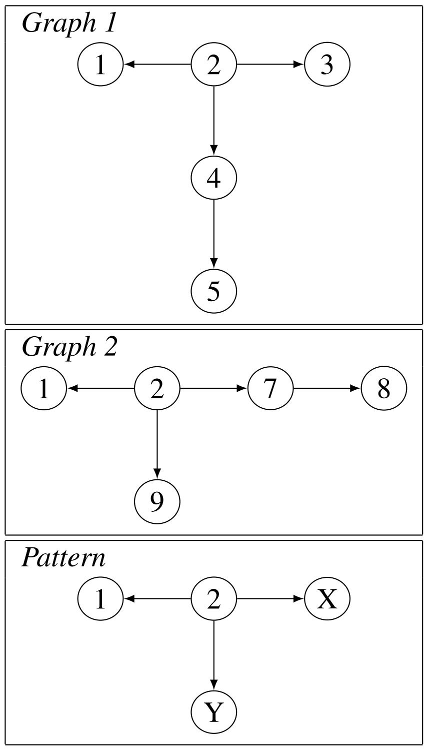


Figure 1: Two graphs and a pattern (variables in uppercase)

Such patterns are used to match the graph representation of the question. If a pattern defined in a rule matches a question sentence, then the rule applies to the sentence.

Our rules specify the pattern of the answer sentence in an unusual way. Instead of keeping a pattern to match the answer sentence, our rules define an *extension* graph that will be added to the graph of the question. The rationale for this is that we want to reward answer sentences that have a high similarity with the question. Therefore, the larger the number of vertices and edges that are shared between the question and the answer, the better. The extension graph contains information that simulates the difference between a question sentence and a sentence containing an answer.

For example, lets us use graph representations of syntactic dependency structures. We will base our representation on the output of Connexor (Tapanainen and Järvinen, 1997), but the choice of parser is arbitrary. The same method applies to the output of any parser, as long as it can be represented as a graph. In our choice, the dependency structure is represented as a bipartite graph where the lexical entries are the vertices represented in boxes and the dependency labels are the vertices represented in ovals. Figure 2 shows the graphs of a question and an answer sentence, and an extension of the question graph. The answer is shown in thick lines, and the extension is shown in dashed lines. This is what we aim to reproduce with our graph rules. In particular, the extension of the question graph is such that the graph of the answer sentence becomes a subgraph of the extended question graph.

The question and answer sentence of Figure 2 have an almost identical dependency graph and consequently the extension required to the question graph is very small. Sentence pairs with more differences would induce a more substantial extension graph.

Note that the extended graph still contains the representation of information that does not appear in the answer sentence, namely the question term *what book*. There is no need to remove any element from the question graph because, as we will see later, the criteria to score the answer extracted are based on the overlap between graphs.

In sum, a graph rule has the following components:

R_p a question pattern;

R_e an extension graph, which is a graph to be added

Q: *What book did Rachel Carson write in 1962?* A: *In 1962 Rachel Carson wrote* ***"Silent Spring"***

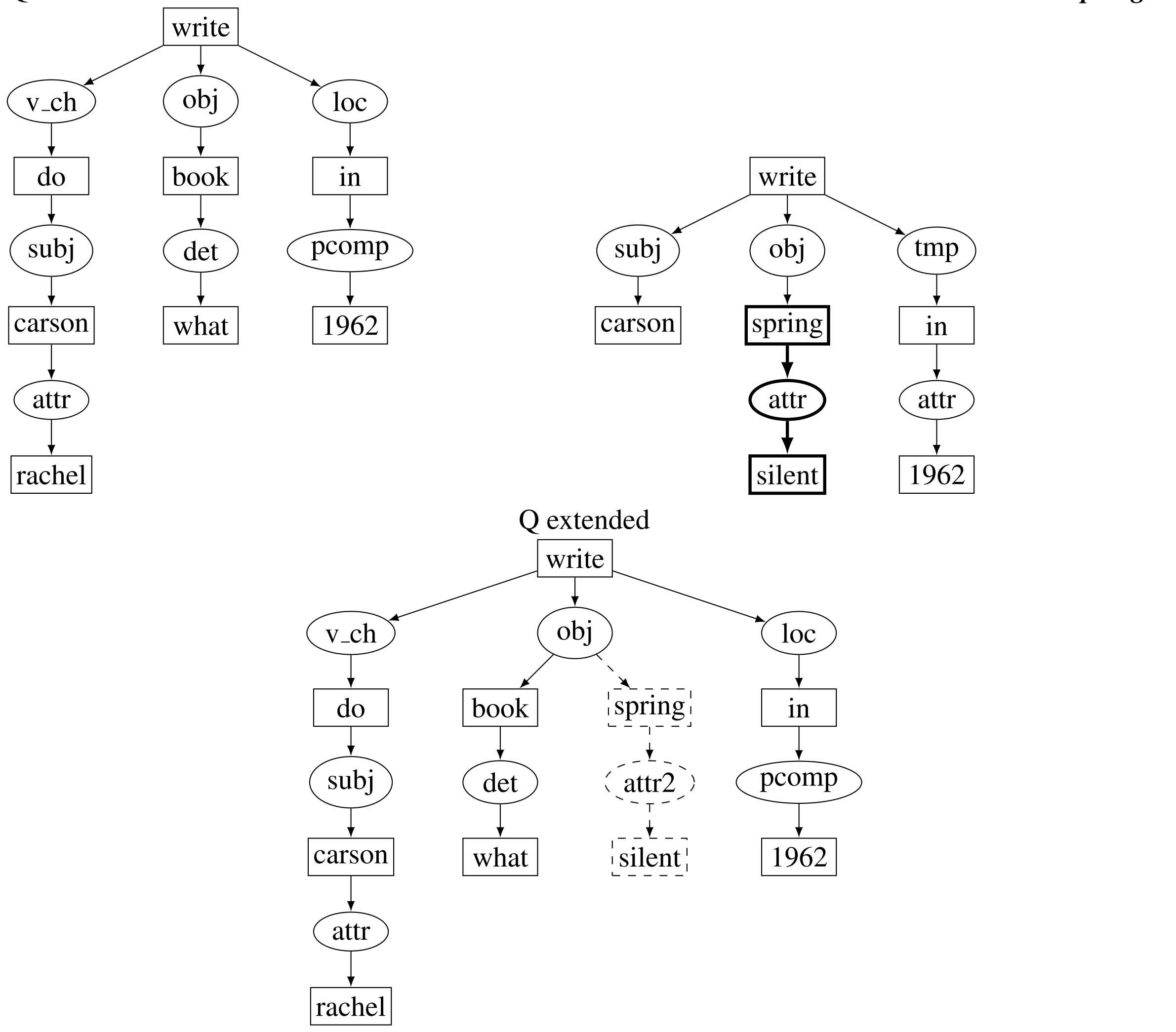


Figure 2: Graph of a question, an answer sentence, and an extension of the question graph

to the question graph; and

R_a a pointer to the answer in the extension graph

An example of a rule is shown in Figure 3. This rule is derived from the pair of question and answer sentence shown in Figure 2.

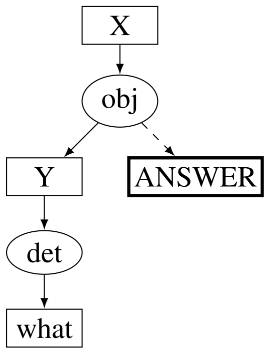


Figure 3: Example of a QA rule. R_p is in solid lines, R_e is in dashed lines, and R_a is in thick lines.

The rule can be used with a fresh pair of question q_i and answer sentence as_i. Let us use the notation $Gr(s)$ to denote the graph that represents the string s. Also, unless said explicitly, names starting with uppercase denote graphs, and names starting with lowercase denote strings. Informally, the process to find the answer is:

1. If $Gr(q_i)$ matches R_p then the rule applies. Otherwise try a new rule.
2. Extend $Gr(q_i)$ with r_e to produce a new graph $E_{q_i}^{R_e}$.
3. Compute the overlap between $E_{q_i}^{R_e}$ and $Gr(as_i)$.
4. If a part of R_a is in the resulting overlap, then expand its projection on $Gr(as_i)$.

The crucial point in the process is to determine the projection of an overlap on the answer sentence, and then to extend it. Once the overlap is found in step 3, if this overlap includes part of the annotated answer, that is if it includes R_a, then part of the answer will be the string in the answer sentence that corresponds to the overlap. The full answer can be retrieved by expanding the answer found in the overlap by following the outgoing edges in the graph of

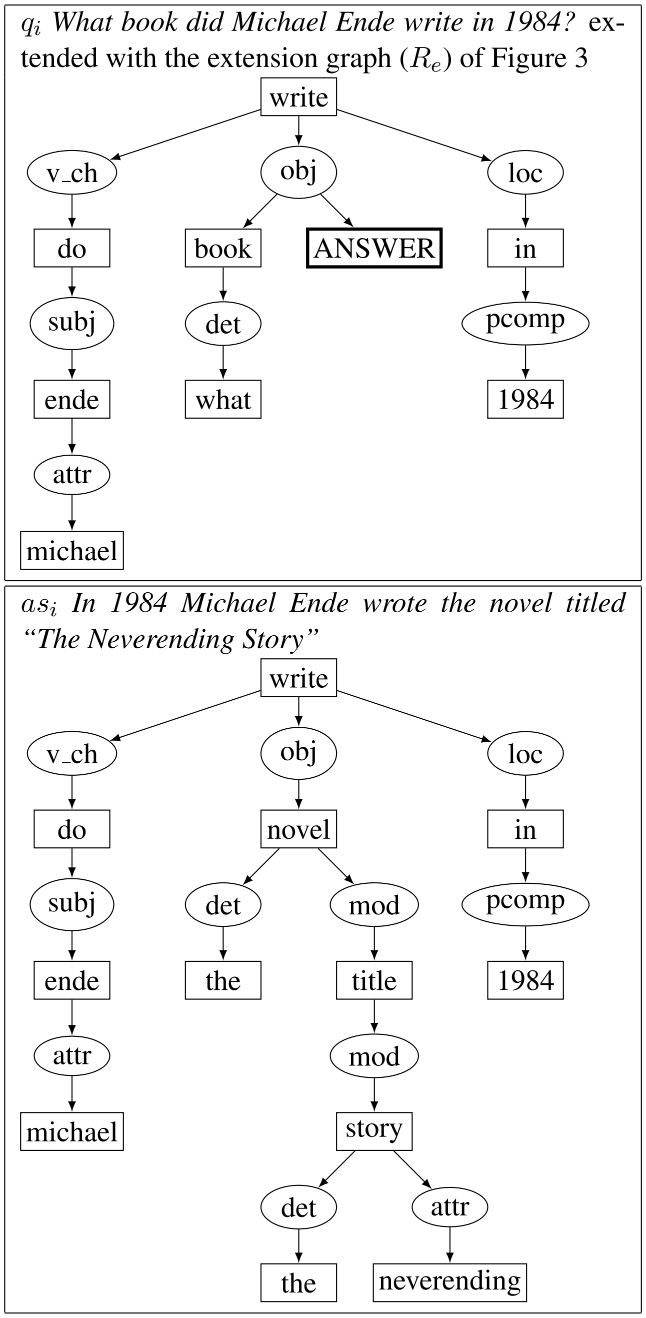

q_i *What book did Michael Ende write in 1984?* extended with the extension graph (R_e) of Figure 3
write
v_ch
obj
loc
do
book
ANSWER
in
subj
det
pcomp
ende
what
1984
attr
michael
as_i *In 1984 Michael Ende wrote the novel titled "The Neverending Story"*
write
v_ch
obj
loc
do
novel
in
subj
det
mod
pcomp
ende
the
title
1984
attr
mod
michael
story
det
attr
the
neverending

Figure 4: An extended graph of a question and a graph of an answer sentence

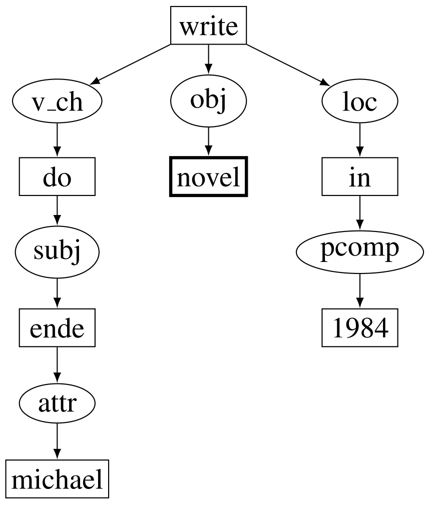


Figure 5: Overlap of the graphs of Figure 4

the answer. Part of the process is shown in Figures 4 and 5.

In Figure 5 the overlap between the extended question graph and the answer sentence graph contains the answer fragment *novel*. After expanding it we obtain the full answer *the novel titled "The Never Ending Story"*.[1]

3 Learning of Graph Rules

To learn a QA rule we need to determine the information that is common between a question and a sentence containing an answer. In terms of graphs, this is a variant of the well-known problem of finding the maximum common subgraph (MCS) of two graphs (Bunke et al., 2002).

The problem of finding the MCS of two graphs is known to be NP-complete, but there are implementations that are fast enough for practical uses, especially if the graphs are not particularly large (Bunke et al., 2002). Given that our graphs are used to represent sentences, their size would usually stay within a few tens of vertices. This size is acceptable.

There is an algorithm based on Conceptual Graphs (Myaeng and López-López, 1992) which is particularly efficient for our purposes.Their method follows the traditional procedure of building the association graph of the two input graphs. However, in contrast with the traditional approach, which finds the cliques of the association graph (and this is the part that is NP-complete), the method by Myaeng and López-López (1992) first simplifies the association graph by merging some of its vertices, and then it proceeds to searching the cliques. By so doing the algorithm is still exponential on the size of n, but now n is smaller than with the traditional approach for the same input graphs.

The method presented by Myaeng and López-López (1992) finds connected graphs but we also need to find overlaps that form unconnected graphs. For example, Figure 6 shows two graphs and their MCS. The resulting MCS is an unconnected graph, though Myaeng and López-López (1992)'s algorithm returns the two parts of the graph as independent MCSs. It is easy to modify the original algorithm to obtain the desired output, as we did.

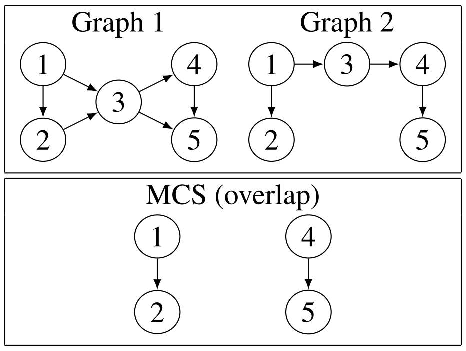


Figure 6: MCS of two graphs

Given two graphs G_1 and G_2, then their MCS is $MCS(G_1, G_2)$. To simplify the notation, we will often refer to the MCS of two sentences as $MCS(s_1, s_2)$. This is to be understood to be the MCS of the graphs of the two sentences $MCS(Gr(s_1), Gr(s_2))$.

Let us now assume that the graph rule R is originated from a pair (q,as) in the training corpus, where q is a question and as a sentence containing the answer a. The rule components are built as follows:

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

R_e is the path between the projection of R_p in $Gr(as)$ and the actual answer $Gr(a)$.

R_a is the graph representation of the exact answer.

[1]Note that this answer is not an exact answer according to the TREC definition since it contains the string *the novel titled*; one further step would be needed to extract the exact answer; this is work for further research.

Note that this process defines R_p as the MCS of question and answer sentence. Consequently, R_p is a subgraph of both the question and the answer sentence. This constraint is stronger than that of a typical QA rule, where the pattern needs to match the question only. The resulting question pattern is therefore more general than it could be had one manually built the rule. R_p does not include question-only elements in the question pattern because it is difficult to determine what components of the question are to be added to the pattern, and what components are idiosyncratic to the specific question used in the training set.

Rules learnt this way need to be generalised in order to form generic patterns. We currently use a simple method of generalisation: convert a subset of the vertices into variables. To decide whether a vertex can be generalised a list of very common vertices is used. This is the list of "stop vertices", in analogy to the concept of stop words in methods to detect keywords in a string. Thus, if a vertex is not in the list of stop vertices, then the vertex can be generalised. The list of stop vertices is fixed and depends on the graph formalism used.

For the question answering process it is useful to associate a weight to every rule learnt. The rule weight is computed by testing the accuracy of the rule in the training corpus. This way, rules that over-generalise acquire a low weight. The weight $\mathcal{W}(r)$ of a rule r is computed according to its precision on the training set:

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

4 Application: QA with Logical Graphs

The above method has been applied to graphs representing the logical contents of sentences. There has been a long tradition on the use of graphs for this kind of sentence representation, such as Sowa's Conceptual Graphs (Sowa, 1979), and Quillian's Semantic Nets (Quillian, 1968). In our particular experiment we have used a graph representation that can be built automatically and that can be used efficiently for QA (Mollá and van Zaanen, 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*.

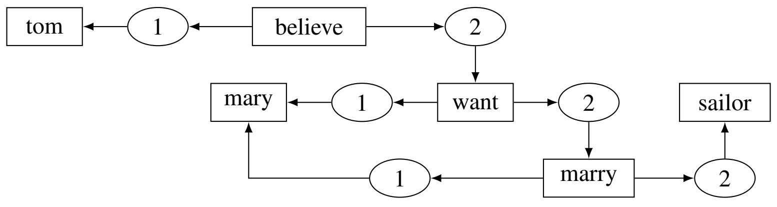
Relations Relations act as links between concepts. To facilitate the production of the LGs we have decided to use relation labels that represent verb argument positions. Thus, the relation *1* indicates the link to the first argument of a verb (that is, what is usually a subject). The relation *2* indicates 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.

An example of a LG is shown in Figure 7, where the concepts are pictured in boxes and the relations are pictured in ovals.

The example in Figure 7 shows LG's ability to provide the graph representation of sentences with embedded clauses. In contrast, other theories (such as Sowa (1979)'s Conceptual Graphs) would represent the sentence as a graph containing vertices that are themselves graphs. This departs from the usual definition of a graph, and therefore standard Graph Theory algorithms would need to be adapted for Conceptual Graphs. An advantage of our LGs, therefore, is that they can be manipulated with standard Graph Theory algorithms such as the ones described in this paper.

Using the LG as the graph representation of questions and answer sentences, we implemented a proof-of-concept QA system. The implementation and examples of graphs are described by Mollá and van Zaanen (2005) and here we only describe the method to generalise rules and the decisions taken to choose the exact answer.

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 relations edges were 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 were not generalised. The list of stop concepts is very small:



Tom believes that Mary wants to marry a sailor

Figure 7: Example of a Logical Graph

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

Every question/answer pair in the training corpus generates one rule (or more if we use a process of increasingly generalising the rules). Since the rule is based on deep linguistic information, it generalises over syntactic paraphrases. Consequently, a small training corpus suffices to produce a relatively large number of rules.

The QA system was trained with an annotated corpus of 560 pairs of TREC questions and answer sentences where the answers were manually annotated. We only tested the ability of the system to extract the exact answers. Thus, the system accepted pairs of question and answer sentences (where the sentence is guaranteed to contain an answer), and returned the exact answer. Given a question and answer sentence pair, the answer is found by applying all matching rules. All strings found as answers are ranked by multiplying the rule weights and the sizes of the overlaps. If an answer is found by several rules, its score is the sum of all scores of each individual sentence. Finally, if an answer occurs in the question it is ignored. The results of a five-fold cross validation on the annotated corpus gave an accuracy (percentage of questions where the correct answer was found) of 21.44%. Given that the QA system does not do any kind of question classification and it does not use any NE recogniser, the results are satisfactory.

5 Related Research

There have been other attempts to learn QA rules automatically. For example, Ravichandran and Hovy (2002) learns rules based on simple surface patterns. Given that surface patterns ignore much linguistic information, it becomes necessary to gather a large corpus of questions together with their answers and sentences containing the answers. To obtain such a corpus Ravichandran and Hovy (2002) mine the Web to gather the relevant data.

Other methods learn patterns based on syntactic information. For example, Shen et al. (2005) develop a method of extracting dependency paths connecting answers with words found in the question. However we are not aware of any method that attempts to learn patterns based on *logical* information, other than our own.

There is recent interest on the use of graph methods for Natural Language Processing, such as document summarisation (Mihalcea, 2004) document retrieval (Montes-y-Gómez et al., 2000; Mishne, 2004), and recognition of textual entailment (Pazienza et al., 2005). The present very workshop shows the current interest on the area. However, we are not aware of any significant research about the use of conceptual graphs (or any other form of graph representation) for question answering other than our own.

6 Conclusions

We have presented a method to learn question answering rules by applying graph manipulation methods on the representations of questions and answer sentences. The method is independent of the actual graph representation formalism.

We are studying to combine WordNet with a Named Entity Recogniser to produce generalised rules. This way it becomes possible to replace vertices with vertex types (e.g. “PERSON”, “DATE”, etc). We are also exploring the use of machine learning techniques to learn classes of vertices. In particular, grammar induction techniques (van Zaanen, 2002) could be applied to learn types of regularities in the strings.

Further research will also focus on developing methods to extend the question pattern R_p with information found in the question only. A possibility is to keep a database of question subgraphs that are allowed to be added to R_p. This database could be built by hand, but ideally it should be learnt automatically.

Additional research efforts will be allocated to determine degrees of word similarity or paraphrasing, such as the connection between *was born in* and *'s birthplace is*. In particular, we will explore the use of nominalisations. We will also study paraphrasing methods to detect these connections.

Considering that text information as complex as syntactic information or even logic and semantic information can be expressed in graphs (Quillian, 1968; Schank, 1972; Sowa, 1979), we are convinced that the time is ripe to explore the use of graphs for question answering.

References

H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento. 2002. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *Lecture Notes on Computer Science*, volume 2396, pages 123–132. Springer-Verlag, Heidelberg.

Noriko Kando. 2005. Overview of the fifth NTCIR workshop. In *Proceedings NTCIR 2005*.

Rada Mihalcea. 2004. Graph-based ranking algorithms for sentence extraction applied to text summarization. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, companion volume (ACL 2004)*.

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

Diego Mollá and Menno van Zaanen. 2005. Learning of graph rules for question answering. In *Proc. ALTW 2005*, Sydney.

Diego Mollá and Menno van Zaanen. 2006. Answerfinder at TREC 2005. In *Proceedings TREC 2005*. NIST.

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.

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.

Maria Teresa Pazienza, Marco Pennacchiotti, and Fabio Massimo Zanzotto. 2005. Textual entailment as syntactic graph distance: a rule based and a SVM based approach. In *Proceedings PASCAL RTE challenge 2005*.

Ross Quillian. 1968. Semantic memory. In *Semantic Information Processing*, pages 216–270. MIT Press.

Deepak Ravichandran and Eduard Hovy. 2002. Learning surface text patterns for a question answering system. In *Proc. ACL2002*.

Roger C. Schank. 1972. Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology*, 3(4):532–631.

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 Voorhees and Harman (Voorhees and Harman, 2001).

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.

Alessandro Vallin, Bernardo Magnini, Danilo Giampiccolo, Lili Aunimo, Christelle Ayache, Petya Osenova, Anselmo Pe nas, Maarten de Rijke, Bogdan Sacaleanu, Diana Santos, and Richard Sutcliffe. 2005. Overview of the CLEF 2005 multilingual question answering track. In *Proceedings CLEF 2005*. Working note.

Menno van Zaanen. 2002. *Bootstrapping Structure into Language: Alignment-Based Learning*. Ph.D. thesis, University of Leeds, Leeds, UK.

Ellen M. Voorhees and Donna K. Harman, editors. 2001. *The Tenth Text REtrieval Conference (TREC 2001)*, number 500-250 in NIST Special Publication. NIST.

Ellen M. Voorhees. 2001a. Overview of the TREC 2001 question answering track. In Voorhees and Harman (Voorhees and Harman, 2001).

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