AnswerFinder at TREC 2005

DIEGO MOLLÁ AND MENNO VAN ZAANEN

Centre for Language Technology, Macquarie University, Sydney, Australia

{diego, menno}@ics.mq.edu.au

ABSTRACT

AnswerFinder has been completely redesigned for TREC 2005. The new architecture allows for the fast development of question-answering systems for their deployment in the TREC tasks and other applications. The modules use XML to express the services they provide, and they can be queried with XML for their services. In terms of the QA method, a major difference with respect to previous years is the use of graph-based methods to compute the answerhood of a sentence and pin-point the answer. The system uses a set of graph-based patterns that are learned automatically.

1 Introduction

AnswerFinder

- is a research oriented question answering system.
- has been completely redesigned and reimplemented since last year.
- incorporates symbolic information.
- uses automatically induced structural semantic information.
- incorporates flat logical forms and logical graphs (new!).

2 System overview

Process

0 Requests The user can ask the system what functionality it provides. This includes the datasets that it knows about and all the algorithms with their parameters. Also, the user can request a question to be answered.

1 Question analysis The first step is to analyse the question. During this phase, question type classification, which indicates what sort of answer we are looking for, such as location, person, etc. is performed. Also, shallow semantics of the question are extracted.

2 Document selection Next, the most relevant documents are selected based on the information of the question. Only the selected documents are considered in the following phases.

3 Sentence selection Using the information extracted during the question analysis phase, sentences that are likely to contain the answer are extracted from the selected documents. Only these sentences are processed further.

4 Answer selection The information taken from the question is matched against the selected sentences and based on this, the answer is extracted.

Requirements

Flexibility The system should be flexible in several ways. Although it is being developed for the TREC competition, it should also be possible to easily modify it to handle different situations. For example, different input and output formats (of documents, questions, and answers), types of questions and answers, and new algorithms (in all the phases) should be easily to integrate in the system. This allows for easy testing of new ideas in different environments.

Configurability Having a system with many different algorithms that can be used in the phases, it should be easy to configure the system to run using specific parameters. Parameters in this case mean, not only the actual values needed in the algorithms, but also selecting a particular algorithm in a phase.

Others Other requirements, such as accuracy, speed, memory usage, etc. are also considered.

Implementation

- AnswerFinder is implemented using C++. Some external tools (named entity recogniser, connexor) are called as separate processes.
- The system is fully documented, which was done before development (requirements specification and design documents) and during the system development.
- Not only is all data stored in classes, all different algorithms for each of the phases are also contained in class hierarchies.
- Builders handle the class hierarchies. They can be used to query the functionality and also handle the creation of the correct algorithms.
- XML requests are sent to the builders of the different parts of the system. They handle requests for their own part of the system.
- Adding new functionality is done by simply adding a class to the correct hierarchy and registering it with the builder.

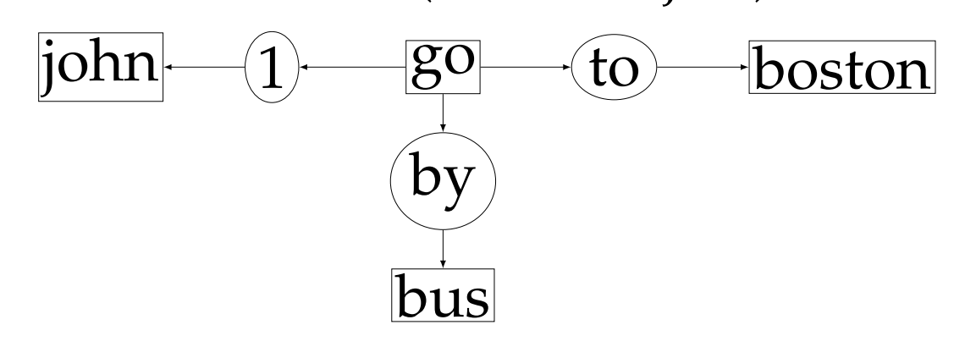
3 Logical Graphs (LG)

- AnswerFinder uses a shallow representation of the semantics of the question and the sentences to find relevant sentences.
- This year, LGs were tested. These are not only used to score sentences, but also to find actual answers.
- LGs are similar to Sowa's (1979) conceptual graphs, but LGs do not attempt to encode the full semantics.
- LGs are computed from the logical forms, which were used in last years competition (Mollá and Gardiner, 2004).
- Logical forms avoid representation of well-known problematic concepts such as quantification, plurality, tense, and aspect.

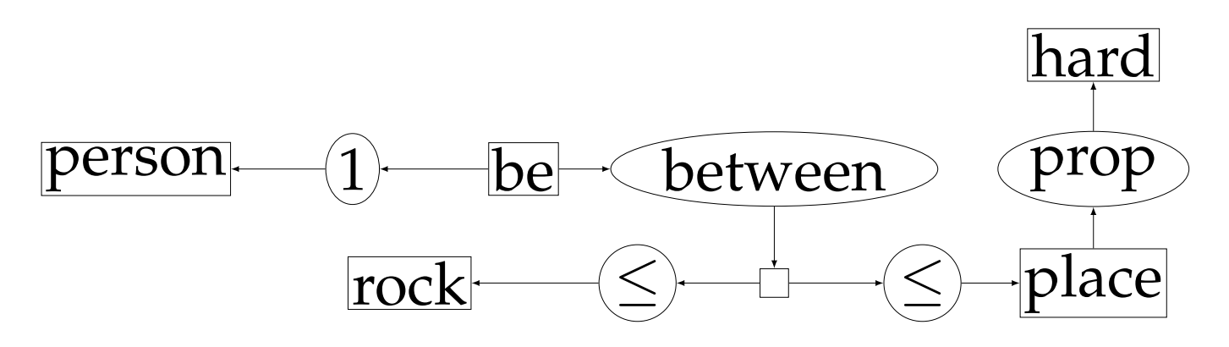
LGs are directed, bipartite graphs with two types of vertices: concepts and relations:

Concepts For example, objects **dog**, **table**, events and states **run**, **love**, and properties **red**, **quick**.

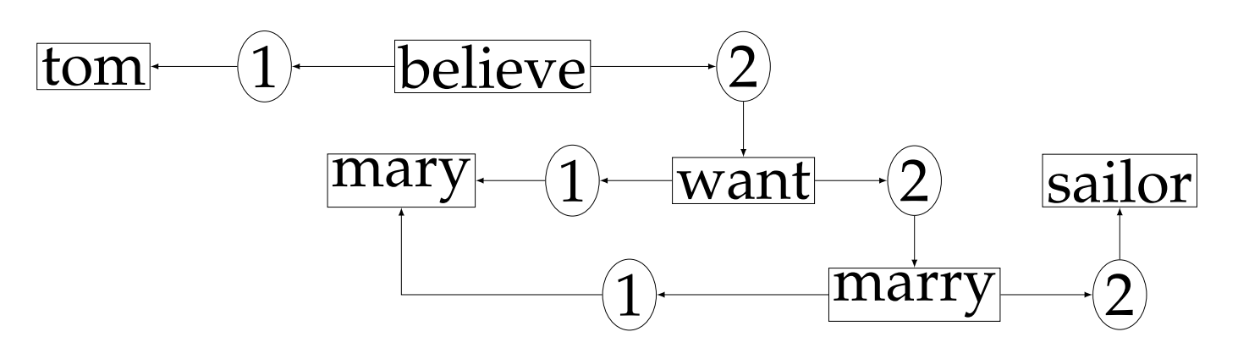
Relations Relations act as links between concepts. Examples of relations would be grammatical roles and prepositions. We use relation labels that are relatively close to the syntactic level, such as **subject**, **object**, etc. In fact, we use numbers that represent verb arguments. **1** indicates the first argument of a verb (that is, what is usually a subject). The relation **2** indicates the second argument of a verb (direct object), and so forth.



John is going to Boston by bus



A person is between a rock and a hard place



Tom believes that Mary wants to marry a sailor

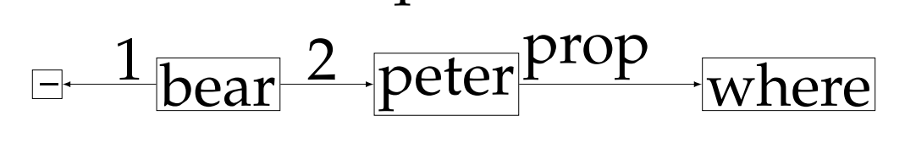
4 Logical Graph Rules (LGR)

- LGRs can be used to extract precise answers from sentences by applying the question LG to the sentence LG.
- Computing graph overlap and paths between subgraphs indicates relevance and finds possible answers. (Mollá and van Zaanen, 2005)
- Each rule r contains three components:

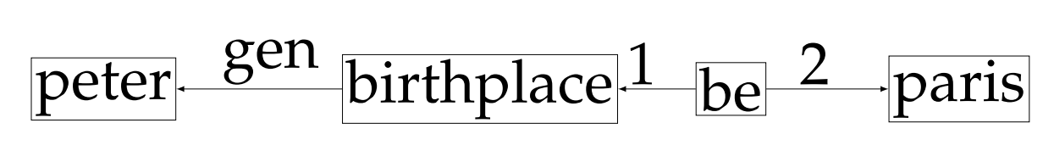
  r_o overlap between a question and its answer sentence;

  r_p path between the overlap and the answer in the sentence;

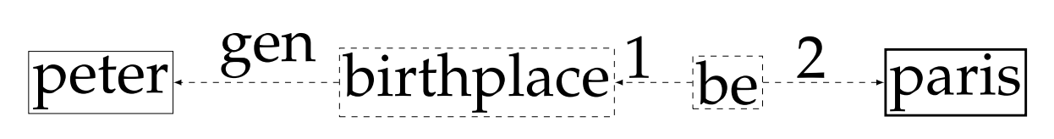
  r_a graph representing the exact answer.
- LGRs are learned from questions and sentences that are annotated with the answer.



Q:**Where was Peter born?**



A:**Peter's birthplace was Paris**



r_o**=regular,** r_p**=dashed,** r_a**=thick**

- Rules are generalised by making concepts wildcards (relations stay fixed).
- "Stop concepts" are not generalised.
- Scoring of rules ($\mathcal{W}(r)$) is done using the following formula that is computed on the training data:

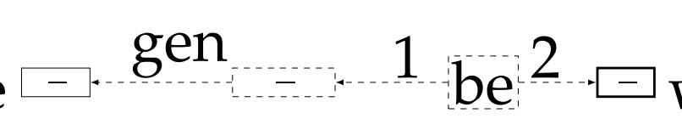
$$\mathcal{W}(r) = \frac{\#\text{ correct answers found}}{\#\text{ answers found}}$$

Acknowledgements

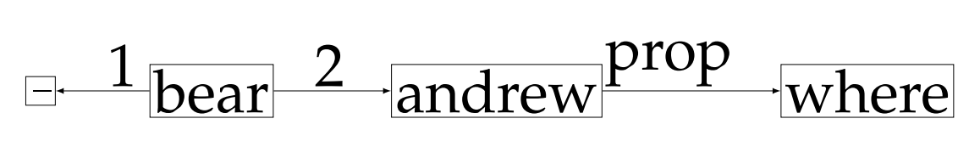
This research is funded by the Australian Research Council, ARC Discovery Grant no. DP0450750.

5 Graph-based Question Answering

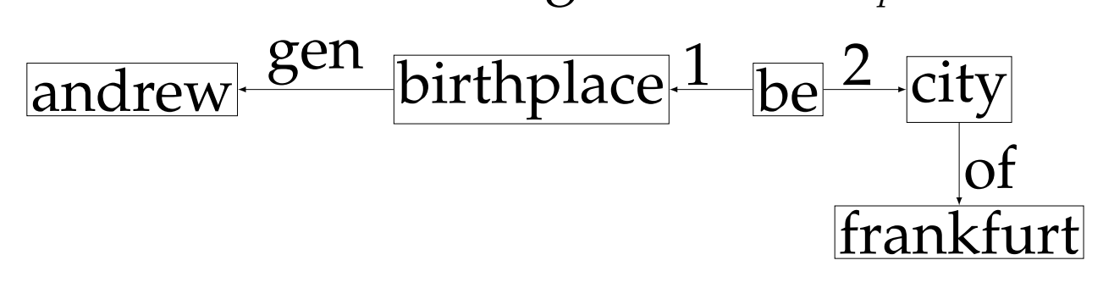
- All LGRs are applied to the sentences. A LGR triggers if r_o is a subgraph of the question. The LG of the question is then extended with r_p.
- The expanded LG is used to compute overlap with the sentences.
- The new LG has a higher overlap with LGs of relevant sentences and indicates possible answers.
- Possible answers are ranked using $\mathcal{A}(s) = \mathcal{W}(r) \times size(ovl(Q, S))$ where $size(ovl(Q, S))$ is the size of the overlap between the expanded LG of the question and the sentence.
- The size of a graph overlap is computed as the weighted sum of all concepts and relations in the overlap. The weight $\mathcal{W}_i$ of a concept or relation i in the overlap is determined using a variant of the Inverse Document Frequency (IDF) measure used in document retrieval. $\mathcal{W}_i = \frac{1}{\log N} \log \frac{N}{n}$ where n is the number of sentences using the concept (or relation) i and N is the total number of sentences.
- Sometimes, the overlap between the GLs of the expanded question and the sentence do not contain the complete answer. Answer expansion is then performed by adding all concepts that are accessible from the answer.

For example, when using rule 

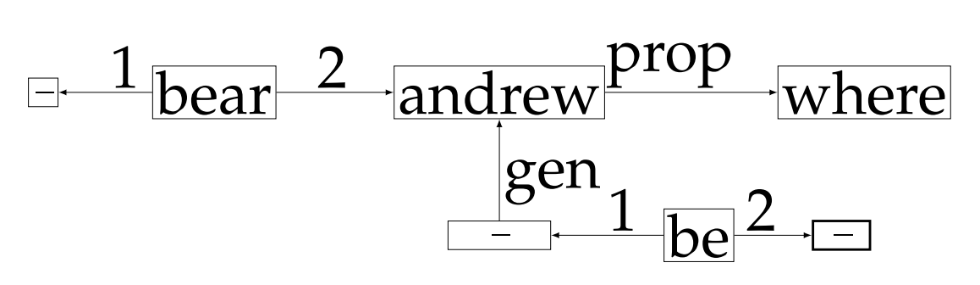
with r_o in regular lines, r_p in dashed lines, r_a in thick lines on



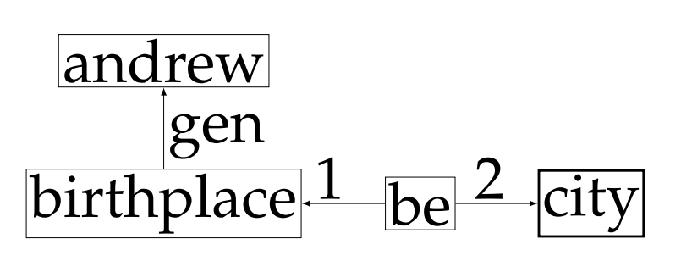
Where was Andrew born?



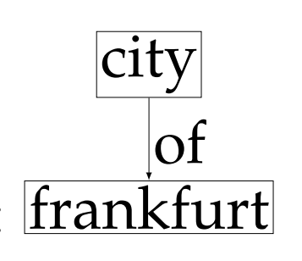
Andrew's birthplace was the city of Frankfurt



Question expanded with rule



Resulting overlap between expanded question and sentence

This says that **city** is the answer. This is now expanded to: 

6 Parameters

	af_run 1	af_run2
document selection	PRISE (NIST) best 50	PRISE (NIST) best 50
question classification	regular expressions	regular expressions
sentence selection	word overlap best 100	word overlap best 5
possible answer generation	flat logical forms	LG
named entity recogniser	LingPipe	LingPipe

7 Results

		af_run1		af_run2	
factoid	accuracy	0.028	10/362	0.014	5/362
factoid	precision no answ	0.077	3/39	0.075	4/53
factoid	recall no answ	0.176	3/17	0.235	4/17
list	average f	0.008		0.000	
other	average f	0.000		0.000	

- Implementation is too slow: only a very limited number of sentences can be handled.
- The system needs further testing and bugs need to be fixed.
- Tokenisation of sentences and words should be implemented as offset annotation.
- Similar, possibly overlapping answers need to be grouped.

8 Conclusions

- AnswerFinder is completely redesigned and reimplemented.
- The system is highly modular and therefore easy to extend.
- LGs are a new way of describing shallow semantics of questions and sentences.
- LGs and LGRs seem to work well in theory, but the implementation needs to be improved.
- Additional implementation problems need to be fixed.