Quartz: A Question Answering System for Dutch

David Ahn, Valentin Jijkoun, Joris van Rantwijk,
Maarten de Rijke, and Erik Tjong Kim Sang

ISLA, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{ahn,jijkoun,rantwijk,mdr,erikt}@science.uva.nl
http://ilps.science.uva.nl

Abstract. We describe a question answering system for Dutch that we used for our participation in the 2006 CLEF Question Answering Dutch monolingual task. We give an overview of the system and focus on the question classification module, the multi-dimensional markup for data storage and access, the answer processing component, and our probabilistic approach to estimating answer correctness.

1 Introduction

For our earlier participations in the CLEF question answering track (2003–2005), we had developed a question answering architecture that uses different competing strategies to find answers in a document collection. For the 2005 edition of CLEF-QA, we focused on converting our text resources to XML in order to facilitate a QA-as-XML-retrieval strategy. This paper describes the 2006 edition of our QA system *Quartz*. We focused on converting all of our data resources (text and linguistic annotations) to fit in a database that uses XML-based multi-dimensional markup to provide uniform access to all annotated information. Additionally, we devoted attention to improving known weak parts of our system: question classification, answer clustering, and answer candidate score estimation.

This paper is divided in eight sections. In section 2, we give an overview of the current system architecture. In the following four sections, we zoom in on recent developments in the system: question classification (section 3), storing data with multi-dimensional markup (section 4), and probabilistic answer processing (sections 5 and 6). In section 7 we present the results of the CLEF QA 2006 evaluation of our system. We conclude in section 8.

2 System Description

The architecture of our Quartz QA system is an expanded version of a standard QA architecture consisting of parts dealing with question analysis, information retrieval, answer extraction, and answer post-processing (clustering, ranking, and selection). The Quartz architecture consists of multiple answer extraction modules, or *streams*, which share common question and answer processing components. These streams can be divided into three groups based on the corpus

C. Peters et al. (Eds.): CLEF 2006, LNCS 4730, pp. 362–371, 2007.
© Springer-Verlag Berlin Heidelberg 2007

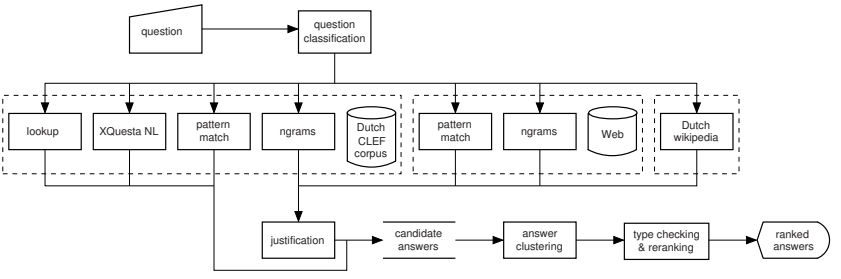


Fig. 1. Quartz: the University of Amsterdam's Dutch Question Answering System

they employ: the CLEF-QA corpus, the Web, or Wikipedia. We describe these streams in the rest of this section. The common question and answer processing parts will be detailed in later sections on the architecture updates.

The Quartz system (Figure 1) contains five answer generating streams. The *Table Lookup* stream searches for answers in specialized knowledge bases which are extracted from the corpus offline (prior to question time) by manually defined rules. These rules take advantage of the fact that certain answer types, such as birthdays, are typically expressed in one of a small set of easily identifiable ways. Currently, the system makes use of 15 specific knowledge bases, containing information about locations, persons, monetary units, measurements, etc. Our question analysis module (section 3) selects which fields in which knowledge bases should be consulted for an incoming question and selects question keywords that are used by the *Table Lookup* stream to retrieve a list of candidate answers.

The *Ngrams* stream implements an approach similar to [1]. It looks for answers by extracting the most frequent word ngrams occurring in passages retrieved either from the document collection or from the Web (using Google). Similarly, the *Pattern Match* stream obtains passages relevant to the topic of the question either from the collection or from the Web and then searches for patterns formulated as regular expressions; the latter are generated by means of a hand-coded list of rules that use the result of the question analysis.

The *Wikipedia* stream relies on the structure of the Dutch Wikipedia. The focus of the question—usually the main named entity—is identified and looked up in Wikipedia. We use the standardized structure of Wikipedia to obtain a list of sentences relevant to the question focus. Then, the patterns of the *Pattern Match* stream and NE extraction are used to produce answer candidates. The *Justification* module (used also in *Ngrams* and *Web Pattern Match* streams) tries to find support for answer candidates in the Dutch CLEF corpus using IR.

The most advanced of the five CLEF-QA corpus streams is *XQuesta*. It performs XPath queries against an XML version of the CLEF-QA corpus which contains the corpus text and additional annotations, including information about part-of-speech, syntactic chunks, named entities, temporal expressions, and dependency parses (from the Alpino parser [2]). XQuesta retrieves passages of at least 400 characters, starting and ending at paragraph boundaries.

3 Question Classification

An important problem identified in the error analysis of our previous CLEF-QA results was a mismatch between the type of the answer and the type required by the question. To address this issue, we collected training data for the question classifier. We used 600 questions from the three previous CLEF QA tracks, 1000 questions from a Dutch trivia game and 279 questions from other sources—most importantly, the questions provided by users of our online demo. All the questions were manually classified by a single annotator.

Our question classification scheme assigns several classes to a question. The main reason is that while a coarse-grained type such as `person` would often be sufficient, for answering *which*-questions a fine-grained type such as `American President` would be needed. We used three different types of question classes: a *table type* that linked the question to a particular column in our Table Lookup knowledge bases (17 classes), a *coarse-grained type* that linked the question to the types recognized by our named-entity recognizer (7 classes), and a *fine-grained type* that linked the question to WordNet synsets (166 classes).

In Quartz, question classification is performed by a machine learner that represents a question using 10 features, such as the question word, the main verb of the question, the first noun following the question word, the top hypernym of the noun according to our type hierarchy, etc. In order to determine the top hypernym of the main noun of the question we used EuroWordNet [3]. With this small set of features the classifier achieved reasonable performance: 90% correct predictions for the coarse-grained classes.

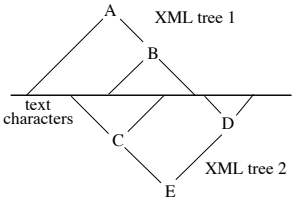
Our classifier uses the memory-based learner Timbl [4]. We performed feature selection in order to identify the best subset of features for the classification task (using bidirectional hill-climbing, see [5]).

4 Multi-dimensional Markup

Our system makes use of several kinds of linguistic analysis tools, including a POS tagger, a named entity recognizer and a dependency parser [2]. These tools are run offline on the entire corpus, before any questions are posed.

The output of an analysis tool takes the form of a set of annotations: the tool identifies text regions in the corpus and associates some metadata with each region. Ideally, we would store these annotations as XML in order to access them through the powerful XQuery language. However, storing and accessing the annotations produced by different tools is not straightforward because the tools may produce conflicting regions. For example, a named entity may partially overlap with a phrasal constituent in such a way that the corresponding XML elements cannot be properly nested. It is thus not possible, in general, to construct a single XML tree that contains annotations from all tools that we used.

To deal with this problem, we have developed a general framework for the representation of multi-dimensional XML markup [6]. This framework stores



Context	Axis	Result nodes
A	select-narrow	A B C
A	select-wide	A B C E
A	reject-narrow	E D
A	reject-wide	D

Fig. 2. Example document with two annotation layers

multiple layers of annotations referring to the same base document. Through an extension of the XQuery language, it is possible to retrieve information from several layers with a single query.

4.1 Stand-Off XML

We store all annotations as stand-off XML. Textual content is stripped from the XML tree and stored separately in a *blob* file (binary large object). Two *region attributes* are added to each XML element to specify the byte offsets of its start and end position with respect to the blob file. This process can be repeated for all annotation layers, producing identical blobs but different stand-off XML markup.

Figure 2 (left) shows an example of two annotation layers on the same text content. The stand-off markup in the two XML trees could be:

```
<A start="10" end="50">
  <B start="30" end="50"/>
</A>
```

```
<E start="20" end="60">
  <C start="20" end="40"/>
  <D start="55" end="60">
</E>
```

We use a separate system, called XIRAF [6], to coordinate the process of automatically annotating the corpus. XIRAF combines multiple text processing tools, each having an input descriptor that defines the format of the input (e.g., entire document or separate sentences), and a tool-specific wrapper that converts the tool output into stand-off XML annotation. We refer to [6] for a more detailed description of XIRAF.

4.2 Extending MonetDB/XQuery

The merged XML documents are indexed using MonetDB/XQuery [7], an XML database engine with full XQuery support. Its XQuery front-end consists of a compiler which transforms XQuery programs into a relational query language internal to MonetDB.

We extended the XQuery language by defining four new path axes that allow us to step between layers. The new axis steps relate elements by region overlap in the blob, rather than by nesting relations in the XML tree: `select-narrow` selects elements that have their region completely contained within the context

element's region; `select-wide` selects elements that have at least partial overlap with the context element's region; `reject-narrow` and `reject-wide` select the non-contained and non-overlapping region elements, respectively. The table in Figure 2 demonstrates the results of each of the new axes when applied to our example document.

In addition to the new axes, we added an XQuery function `so-blob($node)`. This function takes an element and returns the contents of that element's blob region. This function is necessary because all text content has been stripped from the XML markup, making it impossible to retrieve text directly from the XML tree.

The XQuery extensions were implemented by modifying the front-end of MonetDB/XQuery. An index on the offset attributes is used to make the new axis steps efficient even for large documents.

4.3 Using Multi-dimensional Markup for QA

Two streams in our QA system have been adapted to work with multi-dimensional markup: the *Table Lookup* stream and *XQuesta*. The table stream relies on a set of tables that are extracted from the corpus offline according to predefined rules. These extraction rules were rewritten as XQuery expressions and used to extract the tables from the collection text.

Previous versions of XQuesta had to generate separate XPath queries to retrieve annotations from several markup layers. Information from these layers had to be explicitly combined within the stream. Moving to multi-dimensional XQuery enables us to query several annotation layers jointly, handing off the task of combining the layers to MonetDB. Since XQuery is a superset of XPath, previously developed query patterns could still be used in addition to the new, multi-dimensional ones.

5 A Probabilistic Approach to Answer Selection

One major consequence of the multi-stream architecture of the Quartz QA system is the need for a module to choose among the candidate answers produced by the various streams. The principal challenge of such a module is making sense of the confidence scores attached by each stream to its candidate answers. This year, we made use of data from previous CLEF-QA campaigns in order to estimate correctness probabilities for candidate answers from stream confidence scores. Furthermore, we also estimated correctness probabilities conditioned on well-typedness in order to implement type-checking as Bayesian update. In the rest of this section, we describe how we estimated these probabilities. We describe how we use these probabilities in the following section.

5.1 From Scores to Probabilities

Each stream of our QA system attaches confidence scores to the candidate answers it produces. While these scores are intended to be comparable for answers

produced by a single stream, there is no requirement that they be comparable across streams. In order to make it possible for our answer re-ranking module (described in §6) to rank answers from different streams, we took advantage of answer patterns from previous editions of CLEF QA to estimate the probability that an answer from a given stream with a given confidence score is correct.

For each stream, we ran the stream over the questions from the previous editions of CLEF and binned the candidate answers by confidence score into 10 equally-sized bins. Then, for each bin, we used the available answer patterns to check the answers in the bin and based on these assessments, computed the maximum likelihood estimate for the probability that an answer with a score falling in the range of the bin would be correct. With these probability estimates, we can now associate with a new candidate answer a correctness probability based on its confidence score.

5.2 Type Checking as Bayesian Update

Type checking can be seen as a way to increase the information we have about the possible correctness of an answer. We discuss in section 6.2 how we actually type-check answers; in this section, we explain how we use Bayesian update to incorporate the results of type-checking into our probabilistic framework.

Given the prior probability of correctness for a candidate answer, $P(correct)$ (in our case, the MLE corresponding with the stream confidence score), as well as the information that it is well- or ill-typed (represented as the value of the random variable $well\_typed$), we compute $P(correct \mid well\_typed)$, the updated probability of correctness given well- (or ill-)typedness as follows:

$$P(correct \mid well\_typed) = P(correct) \times \frac{P(well\_typed \mid correct)}{P(well\_typed)} \quad (1)$$

In other words, the updated correctness probability of a candidate answer, given the information that it is well- or ill-typed, is the product of the prior probability and the ratio $P(well\_typed \mid correct)$ / $P(well\_typed)$.

We estimate the required possibilities by running our type-checker on assessed answers from CLEF-QA 2003, 2004, and 2005. For question types in which type-checking is actually possible, the ratio for well-typed answers is 1.25. For ill-typed answers the ratio is 0.34.

6 Answer Processing

The multi-stream architecture of Quartz embodies a high-recall approach to question answering—the expectation is that using a variety of methods to find a large number of candidate answers should lead to a greater chance of finding correct answers. The challenge, though, is choosing *correct* answers from the many candidate answers returned by the various streams. The answer processing module described in this section is responsible for this task.

```
 1: procedure AnswerProcessing(candidates, question, tc)
 2:     clusters ← Cluster(candidates)
 3:     for cluster ∈ clusters do
 4:         for ans ∈ cluster do
 5:             ans.well_formed, ans.well_typed ← Check(ans, question)
 6:         end for
 7:         P(cluster) ← 1 − ∏_{ans∈cluster} (1 − P(ans|ans.well_typed))
 8:         cluster.rep_answer ← argmax_{ans∈{ a | a∈cluster∧a.well_formed }} length(ans)
 9:     end for
10:     ranked_clusters ← sort_P(clusters)
11: end procedure
```

Fig. 3. High-level answer processing algorithm

The algorithm used by the Quartz answer processing module is given in Figure 3. Candidate answers for a question are clustered (line 2: section 6.1), and each cluster is evaluated in turn (lines 3–9). Each answer in a cluster is checked for well-formedness and well-typedness (line 5: section 6.2). The correctness probability of the cluster $P(cluster)$ (line 7) is the combination of the updated correctness probabilities of the answers in the cluster (the prior correctness probabilities $P(ans|ans.well\_typed)$ are described in section 5.2). The longer of the well-formed answers in the cluster is then chosen as the representative answer for the cluster (line 8). Finally, the clusters are sorted according to their correctness probabilities (line 10).

6.1 Answer Clustering

Answer candidates with similar or identical answer strings are merged into clusters. In previous versions of our system, this was done by repeatedly merging pairs of similar answers until no more similar pairs could be found. After each merge, the longest of the two answer strings was selected and used for further similarity computations.

This year, we moved to a graph-based clustering method. Formulating answer merging as a graph clustering problem has the advantage that it better captures the non-transitive nature of answer similarity. For example, it may be the case that the answers *oorlog* and *Wereldoorlog* should be considered similar, as well as *Wereldoorlog* and *wereldbeeld*, but not *oorlog* and *wereldbeeld*. To determine which of these answers should be clustered together, it may be necessary to take into account similarity relations with the rest of the answers.

Our clustering method operates on a matrix that contains a similarity score for each pair of answers. The similarity score is an inverse exponential function of the edit distance between the strings, normalized by the sum of the string lengths. The number of clusters is not set in advance but is determined by the algorithm. We used an existing implementation of a spectral clustering algorithm [8] to compute clusters within the similarity graph. The algorithm starts by putting all answers in a single cluster, then recursively splits clusters according to spectral

analysis of the similarity matrix. Splitting stops when any further split would produce a pair of clusters for which the normalized similarity degree exceeds a certain threshold. The granularity of the clusters can be controlled by changing the threshold value and the parameters of the similarity function.

6.2 Checking Individual Answers

The typing and well-formedness checks performed on answers depends primarily on the expected answer type of the question. Real type-checking can only be performed on questions whose expected answer type is a named-entity, a date, or a numeric expression. For other questions, only basic well-formedness checks are performed. Note that the probability update ratios described in section 5.2 are only computed on the basis of answers that are type-checkable. For answers to other questions, we use the same ratio for ill-formed answers as we do for ill-typed answers (0.34), but we do not compute any update for well-formed answers (i.e., we update with a ratio of 1.0).

To check typing and well-formedness of an answer we use a set of heuristic rules created by analysing the past performance of the system. E.g., for abbreviation or monetary unit questions the answer should consist of one word, answers to measurement question should contain numerical information, etc. For questions expecting names and dates as answers, we run an NE or date tagger, as appropriate, on the justification snippet to verify that the candidate answer is a name (for well-formedness) and is of the correct type (for well-typedness). Then, the results of answer checking are boolean values for well-formedness and well-typedness, as well as the possibly edited answer string.

7 Results and Analysis

For the CLEF 2006 QA task we submitted two Dutch monolingual runs. The run `uams06Tnlnl` used the full system with all streams and final answer selection. The run `uams06Nnlnl` used the full system but without type-checking.

The question classifier performed as expected for coarse question classes: 86% correct compared with a score of 85% on the training data. For most of the classes, precision and recall scores were higher than 80%; the exceptions are `miscellaneous` and `number`. For assigning table classes, the score was much lower than for the training data: 56% compared with 80%.

Table 1 lists the assessment counts for the two University of Amsterdam runs for the question answering track of CLEF-2006. The two runs had 14 different top answers of which four were assessed differently. In 2005 our two runs contained a large number of inexact answers (28 and 29). We are happy about those numbers being lower in 2006 (4 and 4) and the most frequent problem, extraneous information added to a correct answer, has almost disappeared. However, the number of correct answers dropped as well, from 88 in 2005 to 41. This is at least partly caused by the fact that the questions were more difficult this year.

The differences in the evaluation of the two runs are minimal and do not allow us to make any conclusions of the effectiveness of the typechecking in

Table 1. Assessment of the two runs and of one run per question type

Run	Total	Right	Unsupported	Inexact	Wrong	% Correct
`uams06Tnlnl`	200	40	2	4	154	20%
`uams06Nnlnl`	200	41	3	4	152	21%
Assessment for run `uams06Nnlnl` per question type						
factoid	116	28	1	2	85	24%
definition	39	9	0	1	29	23%
temporally restricted	32	3	1	1	27	8%
non-list	187	40	2	4	141	21%
list	13	0	6	0	31	0%

the Dutch system. This is in contrast with our experience with the English version of the system [9], where we noticed a substantial improvement. More experiments are needed to understand what causes the lack of improvement for Dutch: the quality of the Dutch type-checking itself or our probabilistic approach to combining different evidence for answer candidates (section 5).

Like last year, the questions could be divided in two broad categories: questions asking for lists and questions requiring singular answers. The second category can be divided in three subcategories: questions asking for factoids, questions asking for definitions and temporally restricted questions. We have examined the `uams06Tnlnl`-run answers to the questions of the different categories in more detail (Table 1). Our system failed to generate any correct answers for the list questions. For the non-list questions, 21% of the top answers were correct. Within this group, the temporally restricted questions[1] (8% correct) posed the biggest challenge to our system. In 2005, we saw similar differences between factoid, definition and temporally restricted questions. At that time the difference between the last category and the first two could be explained by the presence of a significant number of incorrect answers which would have been correct in another time period. This year, no such answers were produced by our system.

8 Conclusion

We have described the fourth iteration of our system for the CLEF Question Answering Dutch mono-lingual track (2006). This year, our work has focused on converting all data repositories of the system (text, annotation and tables) to XML and allowing them to be accessed via a single interface. Additionally, we modified parts of our system which we had suspected of weaknesses: question classification and answer processing.

Our experiments with the English version of the system [10] show that if a fine-tuned answer selection module is employed, all QA streams contribute to performance: removing any single stream results in fewer answered questions. We still need to perform similar experiments for the Dutch system.

[1] The official assessments indicate only one temporally restricted question, which seems unlikely. We count all questions with time expressions as temporally restricted.

Acknowledgments

This research was supported by various grants from the Netherlands Organisation for Scientific Research (NWO). Valentin Jijkoun was supported under project numbers 220.80.001, 600.065.120 and 612.000.106. Joris van Rantwijk and David Ahn were supported under project number 612.066.302. Erik Tjong Kim Sang was supported under project number 264.70.050. Maarten de Rijke was supported by NWO under project numbers 017.001.190, 220.80.001, 264.70.050, 354.20.005, 600.065.120, 612.13.001, 612.000.106, 612.066.302, 612.069.006, 640.-001.501, 640.002.501, and by the E.U. IST programme of the 6th FP for RTD under project MultiMATCH contract IST-033104.

References

1. Dumais, S., Banko, M., Brill, E., Lin, J., Ng, A.: Web question answering: Is more always better? In: Bennett, P., Dumais, S., Horvitz, E. (eds.) Proceedings of SIGIR'02, pp. 291–298 (2002)
2. van Noord, G.: At last parsing is now operational. In: Proceedings of TALN 2006, Leuven, Belgium (2006)
3. Vossen, P.: EuroWordNet: A Multilingual Database with Lexical Semantic Networks. Kluwer Academic Publishers, Dordrecht (1998)
4. Daelemans, W., Zavrel, J., van der Sloot, K., van den Bosch, A.: TiMBL: Tilburg Memory Based Learner, version 5.1, Reference Guide. University of Tilburg, ILK Technical Report ILK-0402 (2004), `http://ilk.uvt.nl/`
5. Caruana, R., Freitag, D.: Greedy attribute selection. In: Proceedings of the Eleventh International Conference on Machine Learning, New Brunswick, NJ, USA, pp. 28–36. Morgan Kaufman, San Francisco (1994)
6. Alink, W., Jijkoun, V., Ahn, D., de Rijke, M., Bonz, P., de Vries, A.: Representing and querying multi-dimensional markup for question answering. In: Proceedings of the 5th Workshop on NLP and XML, ACL, pp. 3–9 (2006)
7. CWI: MonetDB Website: `http://www.monetdb.nl/`
8. Dragone, L.: Spectral clusterer for WEKA (2002), `http://www.luigidragone.com/datamining/spectral-clustering.html`
9. Schlobach, S., Ahn, D., de Rijke, M., Jijkoun, V.: Data-driven type checking in open domain question answering. Journal of Applied Logic (2006)
10. Jijkoun, V., de Rijke, M.: Answer selection in a multi-stream open domain question answering system. In: McDonald, S., Tait, J. (eds.) ECIR 2004. LNCS, vol. 2997, Springer, Heidelberg (2004)
11. Jijkoun, V., de Rijke, M.: Retrieving answers from frequently asked questions pages on the web. In: Proceedings of the Fourteenth ACM conference on Information and knowledge management (CIKM 2005), ACM Press, New York (2005)