Tequesta: The University of Amsterdam's Textual Question Answering System

Christof Monz **Maarten de Rijke**
Language & Inference Technology, University of Amsterdam
Nieuwe Achtergracht 166, 1018 WV Amsterdam
The Netherlands
E-mail: {christof,mdr}@science.uva.nl
URL: www.science.uva.nl/~{christof,mdr}

Abstract: We describe our participation in the TREC-10 Question Answering track. All our runs used the Tequesta system; we provide a detailed account of the natural language processing and inferencing techniques that are part of Tequesta. We also summarize and discuss our results, which concern both the main task and the list task.

1 Introduction

Current information retrieval systems allow us to locate documents that might contain the pertinent information, but most of them leave it to the user to extract the useful information from a ranked list. However, users want not whole documents but brief answers to specific questions. Question answering is meant to be a step closer to real information retrieval in that it attempts to facilitate just that.

For researchers (such as ourselves) who are interested in bringing natural language processing (NLP) and inferencing to bear on real-world tasks, question answering (QA) provides an ideal setting. Many years of experimental research have shown that advanced NLP techniques hurt more than they help for traditional document retrieval [1]. For any system that aims to address the QA task, however, issues such as question classification, partial parsing, and named entity recognition appear to be essential components. In addition, the best performing systems at the TREC-8 and TREC-9 QA tracks have demonstrated that various forms of inferencing (ranging from the use of semantic relations in WordNet to actually abducing answers from questions) make a significant positive contribution towards the effectiveness of QA systems [20, 19]. The recently released (and deliberately ambitious) vision statement that aims to guide future research in QA calls for approaches that are even more knowledge intensive than the current ones [8].

This paper describes our submissions for the question answering track at TREC-10; we submitted runs for the main task and for the list task. This is the first time that we participated in the QA track (and in TREC, for that matter), and our main focus was on evaluting a basic question answering system that exploits shallow NLP techiques in combination with standard retrieval techniques.

The remainder of the paper is organized as follows. Section 2 describes the Tequesta system that we developed for the QA track. We outline the underlying retrieval engine, the kind of document analysis that we perform (partial parsing and named entity recognition), as well as our question analysis and answer selection modules. Then, in Section 3 we describe the runs that we submitted to the main QA task, and discuss the outcomes. In Section 4 we do the same for the runs submitted to the list task. Section 5 contains our conclusions and plans for future work.

2 System Description

2.1 System Architecture

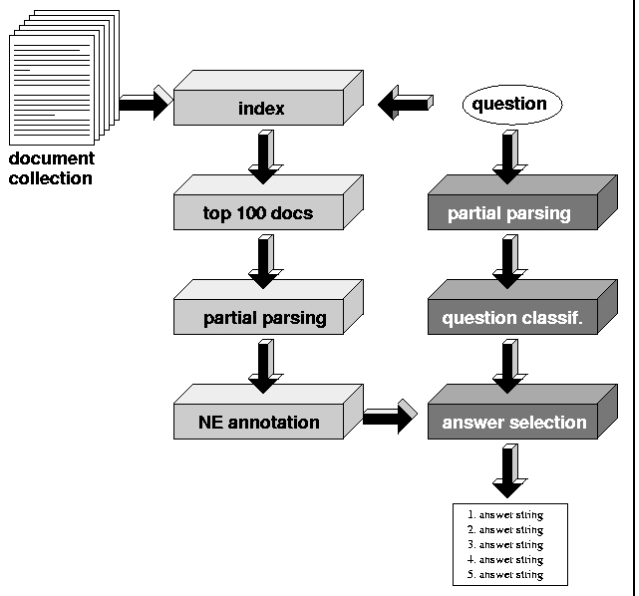
The system architecture of Tequesta is fairly standard; its overall architecture is displayed in Figure 1. Like most current QA systems, Tequesta is built on top of a retrieval system. The first step is to build an index for the document collection, in this case the TREC-10 collection. Then the question is translated into a retrieval query which is posed to the retrieval system. For retrieval we use FlexIR [13], a vector-space based retrieval system, described in Section 2.1.

The retrieval system is used to identify a set of documents that are likely to contain the answer to a question posed to the system. The top 100 documents returned by FlexIR are processed by a partial parser described in Section 2.2. Then, named entities are annotated with the appropriate type. Named entity recognition is discussed in Section 2.3.

Just like the top 100 documents, the question is also parsed. The parsed output is used to determine the focus of the question, i.e., what it is looking for. Question analysis is explained in Section 2.4.

The document analysis and question analysis are mostly done independently from each other, but in order to generate a top 5 list of answers, document information and question information are combined in the answer selection process, described in Section 2.5.

Figure 1: Tequesta system architecture.



2.2 Document Retrieval

For pre-fetching relevant documents that are likely to contain the answer, Tequesta uses FlexIR, an information retrieval system developed at the University of Amsterdam. The main goal underlying FlexIR's design is to facilitate flexible experimentation with a wide variety of retrieval components and techniques. FlexIR is implemented in Perl; it is built around the standard UNIX pipeline architecture, and supports many types of pre-processing, scoring, indexing, and retrieval methods.

The retrieval model underlying FlexIR is the standard vector space model. All our official runs for TREC-10 used the Lnu.ltc weighting scheme [4] to compute the similarity between a question and a document. For the experiments on which we report in this article, we fixed *slope* at 0.2; the pivot was set to the average number of unique words occurring in the collection.

To increase precision, we decided to use a lexical-based stemmer, or lemmatizer, because it tends to be less aggressive than rule-based stemmers such as Porter's [14] or Lovins' [9] stemmer. The lemmatizer is part of the TreeTagger part-of-speech tagger [17]. Each word is assigned its syntactic root through lexical look-up. Mainly number, case, and tense information is removed, leaving other morphological processes such as nominalization intact.

2.3 Document Analysis

2.3.1 Partial Parsing

At present, full parsing is still computationally rather expensive, and building a grammar that is able to cope with a large number of phenomena is very laborious. For these reasons we decided to use a partial parser which can at least identify simple phrases of various kinds. Our partial parser is based on finite-state technology and is therefore able to process large amounts of data efficiently.

We focused on identifying noun phrases (NPs), prepositional phrases (PPs) and verb groups (VGs). A verb group is the verbal complex containing the semantic head of a verb phrase (VP) and its auxiliaries (*have*, *be*) and modal modifiers (*can*, *would*, *should*, etc.).

For each noun phrase, its semantic head is marked. If the noun phrase is complex, the right most noun is identified as the head [21], which holds for almost all noun phrases in English.[1] Similarly, the semantic head of a prepositional phrase is the head of its noun phrase and the syntactic head is the preposition.

NPs, PPs, and VGs form the basic constituents of a dependency structure. A dependency structure is headed by a VG and the NPs and PPs in its vicinity are arguments or modifiers of the verb. We did not exploit subcategorization information derived from the verb in order to deal with ambiguities arising from more complex verb-argument relations such as controlling verbs (e.g., *promise*, *persuade*), and anaphoric relations. A disadvantage of this approach is that it is harder to distinguish between arguments and modifiers of a verb. On the other hand, using a flat and underspecified representation, one does not have to cope with these ambiguities. Since we did not try to resolve anaphoric relations, this approach has the additional advantage that noun phrases serving as antecedents to intra-sentential pronouns are considered to be part of the dependency structure. Consider, for instance, the sentence in (1), taken from document `AP900416-0132`.

(1) Teachers in Oklahoma City and some other districts said they feared reprisals if they took part in the strike.

Neglecting any context possibly preceding (1), there are four potential antecedents of the plural pronoun *they*:

(2) a. Teachers
    b. Teachers in Oklahoma City
    c. some other districts
    d. Teachers in Oklahoma City and some other districts

The correct antecedent of *they*, and therefore the subject of *fear* and *take part* is (2.d). Since resolving anaphora, and plural anaphora in particular, can be rather complex, we refrained from this task and relaxed our notion of dependency structure instead.

There are three dependency structures that can be identified in (1). An abstract representation is given in (3):

(3) a. `head`: say
       `arg(1,1)`: teacher
       `arg(1,2)`: in Oklahoma City
       `arg(1,3)`: some other district

[1] Exceptions of the Right-hand Head Rule (RHR) include some hyphenated noun phrases, such as *passer-by* and *mother-in-law*.

b. head: fear
arg(l,1): teacher
arg(l,2): in Oklahoma City
arg(l,3): some other district
arg(r,4): reprisal

c. head: take-part
arg(l,1): teacher
arg(l,2): in Oklahoma City
arg(l,3): some other district
arg(r,4): reprisal
arg(r,5): in the strike

The two parameters of arg indicate the direction of the argument with respect to the heading verb and the distance.

In the actual system, syntactic annotation is done in XML format. Below we show the dependency structure for (3.b).

```
<C CAT=NP SEMHEAD=teacher TYPE=SMTH ID=217-217>
   <C CAT=NNS ID=217 LEM=teacher>Teachers</C>
</C>
<C CAT=PP SEMHEAD=City SYNHEAD=in ID=218-220>
   <C CAT=IN ID=218 LEM=in>in</C>
   <C CAT=NP SEMHEAD=City TYPE=CITY_3 ID=>
      <C CAT=NNP ID=219 LEM=Oklahoma>Oklahoma</C>
      <C CAT=NNP ID=220 LEM=City>City</C>
   </C>
</C>
<C CAT=CC ID=221 LEM=and>and</C>
<C CAT=NP SEMHEAD=district TYPE=SMTH ID=222-224>
   <C CAT=DT ID=222 LEM=some>some</C>
   <C CAT=JJ ID=223 LEM=other>other</C>
   <C CAT=NNS ID=224 LEM=district>districts</C>
</C>
<C CAT=NP SEMHEAD=they TYPE=SMTH ID=226-226>
   <C CAT=PRP ID=226 LEM=they>they</C>
</C>
<C CAT=VG SEMHEAD=fear PART= VC=act
 DEP=l_217-217(teacher),l_218-220(City),
 l_222-224(district),r_228-228(reprisal)
 ID=227-227>
   <C CAT=VBD ID=227 LEM=fear>feared</C>
</C>
<C CAT=NP SEMHEAD=reprisal TYPE=SMTH ID=228-228>
   <C CAT=NNS ID=228 LEM=reprisal>reprisals</C>
</C>
```

A number of things require further explanation, and we will briefly discuss most of the features present in the XML structure above. The CAT feature represents the syntactic category of a word or a phrase. The word categories are based on the Penn Treebank tag set, cf. [16]. The morphologically normalized form of a word, its lemma, is given by the LEM feature. The SEMHEAD feature marks the semantic head of a phrase, and in case of a PP the SYNHEAD feature marks the preposition as the syntactic head. Each occurrence of a word in a document has a unique identifier, indicated by the ID feature. Similarly, each top-level phrase has a unique identifier indicating its scope. The TYPE feature assigns a semantic type to named entities, SMTH (something) being the default value. Named entity annotation is discussed in more detail in the next subsection. Whether a verb is in active or passive voice is marked by the VC feature.

Returning to the representation of dependency structures, this information is contained in the annotation of the VG phrase. The feature DEP has as its value a list of strings, separated by a comma. For instance, l_222-224(district) says that the phrase 222-224 is within the scope to the left of the verb group and that its semantic head is district. Anaphoric phrases, such as 226-226, are not mentioned in the dependency list.

In addition to VG phrases some noun phrases can also have dependency relations. Nominalizations, such as (4), behave very much like the verbs from which they are derived.

(4) Mr Paul Volcker, the former chairman of the US Federal Reserve Board, is considering an offer to serve as an *adviser* to the Russian government on economic and banking reform.

In (4), taken from document FT921-10181, *adviser*, or its underlying verb *advise*, takes *Mr Paul Volcker* as subject and *Russian government* as object. To identify nominalizations, we used CELEX [2] and NOMLEX [10] as lexical resources.

As we will see in Section 2.5, dependency structures are the basic constituents in the answer selection process for several types of questions. Especially questions of the form *Who VP?* make use of dependency structures to match the question with dependency structures within the document.

2.3.2 Named Entity Annotation

In addition to the syntactic annotation described in the previous subsection, we also annotate some named entities with their semantic types. The set of semantic types that we have used is shown in Table 1. Some of the semantic types, such as PERS and LOC, are further divided into subtypes.

Table 1: Types for named entity annotation.

Type	Subtypes	Description
COMP		companies and organizations
NUMERIC	MONEY	monetary expressions
	NUM-RATIO	percentages
DATE		explicit dates
TIME		time periods
LOC	COUNTRY	countries
	STATES	U.S. states
	PROVINCE	provinces
	CITY	cities
	PORT	harbors
	ISLAND	islands
PERS	MALE	male persons
	FEMALE	female persons
SMTH		other NPs

Type recognition is accomplished by fairly simple techniques such as pattern matching, gazetteer look-up, or a combination of both.

To identify companies, organizations, associations, etc. we compiled a list of names and extracted 20 features that occur frequently. For instance, *&*, *Inc.*, and *International* are likely to indicate the name of a company. If a company or organization name was followed by an expression between parentheses, adhering to some pattern, we took it to be the company's abbreviation and added this information to the annotation in order to allow for aliases. For instance, the annotation for the *Asia Pacific Economic Co-operation Group (APEC)* is as follows:

```
<C CAT=NP SEMHEAD= TYPE=COMP ABBR=APEC ID=356-364>
 <C CAT=DT ID=356 LEM=the>the</C>
 <C CAT=NNP ID=357 LEM=Asia>Asia</C>
 <C CAT=NNP ID=358 LEM=Pacific>Pacific</C>
 <C CAT=NNP ID=359 LEM=Economic>Economic</C>
 <C CAT=NN ID=360 LEM=cooperation>Co-operation</C>
 <C CAT=NNP ID=361 LEM=Group>Group</C>
 <C CAT=( ID=362 LEM=(>(</C>
 <C CAT=NNP ID=363  LEM=APEC>APEC</C>
 <C CAT=) ID=364 LEM=)>)</C>
</C>
```

Keeping track of abbreviations does not only allow one to match a name with its abbreviation when a question is matched with a dependency structure, it can also be used for questions concerning abbreviations directly; e.g., questions of the form *What does X stand for?*.

Phrases of type `NUMERIC`, `DATE`, and `TIME`, are recognized by pattern matching. The TIPSTER gazetteer, containing a list of more than 240,000 locations, is used to find names of cities, provinces, etc.

The identification of person names uses the U.S. census list of the 80,000 most frequent last names, 4275 most frequent female first names, and 1219 most frequent male first names in the U.S. as a gazetteer. In addition we look for particular indicators for a person name, including titles, such as *Mrs.*, *President*, *Dr.*, and relative clauses following an NP with capitalized nouns. If a name was identified by pattern matching, it was dynamically added to the list of known names. Whenever it was possible to identify the gender of a person by looking at the first name or title, the more specific subtype information was recorded. Although we did not yet exploit this distinction in the current version of our system, we plan to do so in the future in order to facilitate anaphora resolution.

If an NP cannot not be recognized by the techniques above, it receives the default semantic type `SMTH`.

Obviously, these techniques are rather simple and error prone. In particular, the use of gazetteers has the disadvantages of being inherently incomplete and causing false alarms; see e.g., [11] for a discussion of the use of gazetteers in the area of Information Extraction. More sophisticated systems such as IdentiFinder^{TM} [3] therefore use feature learning techniques for named entity annotation. On the other hand, the use of gazetteers has the advantage of being rather simple to implement, which was the main reason we opted for this solution.

In the current version of system, false alarms account for the majority of errors made by the name entity recognizer. This is caused mainly by the interference of location names and person names. As we do not allow for multiple typing, this has the effect that once a named entity is falsely recognized as being of type A, it cannot be identified as being of type B. Since it is rather unlikely that we will replace the gazetteer look-up by a feature-learning component in the near future — for the aforementioned reasons — we at least intend to allow for multiple typing. As a consequence, false alarms will continue to have a negative impact on precision, but recall should increase.

Our final remark on the named entity annotation component concerns the interaction between annotation and document retrieval. Currently, the named entity recognizer is applied to the top 100 documents returned by our retrieval system FlexIR. We did not apply the recognizer to the collection as a whole. Pre-processing the whole collection would have two advantages: First, it results in a more efficient system (although efficiency was not one of our major concerns at the current stage), and second, it is possible to index the collection with respect to the semantic types attached to named entities and exploit this additional information during retrieval, cf., e.g., [15]. The main reason for not doing so was that we developed the named entity recognizer in tandem with the other components. Since applying it to the whole collection is rather time consuming, it would have increased the duration of each development cycle in a significant way. We are hopeful that once we have enabled multiple typing, we will have a stable and reliable version of the recognizer which can used to assist the retrieval process.

2.4 Question Analysis

Just like the top 100 documents, the questions themselves were also part-of-speech tagged, morphologically normalized, and partially parsed. Since there is a significant difference between word order in questions and in declarative sentences, we needed to adjust the tagger for questions. To this end, TreeTagger was trained on a set of 500 questions with part-of-speech tags annotated. We used 300 questions taken from the Penn Treebank II data set together with the 200 TREC-8 questions, which we annotated semi-automatically.

We used 18 categories to classify the focus or target of a question; the first 16 of these are listed in Figure 2. The two missing categories (`what:X` and `unknown`) will shortly be discussed.

To identify the target of a question, pattern matching is applied to assign one of the 18 categories to the question. In total, a set of 67 patterns is used to accomplish this. Some of the patterns used are shown in Table 2.

If more than one pattern matches the question, it was assigned multiple targets. The patterns are ordered so that more specific patterns match first. Also, the answer selection component described in the next subsection obeys the order in which questions were categorized to find answers for more specific targets first.

Questions of type `what:X` form a special category. Here

Table 2: Types for question classification.

Question target	Example patterns
name	/(W|w)hat(wa| i|\')s the name/
pers-def	/[Ww]ho(wa| i|\')s [A-Z][a-z]+/
thing-def	/[Ww]hat(wa| i|\')s an? /,/ (was|is|are|were) a kind of what/
pers-ident	/[Ww]ho(wa| i|\')s the/
thing-ident	/[Ww](hat|hich)(wa| i|\')s the /
number	/[Hh]ow (much|many) /
expand-abbr	/stand(s)? for(what)?\s*?/,/is (an|the) acronym/
find-abbr	/[Ww]hat(i|\')s (the|an) (acronym|abbreviation) for
agent	/[Ww]ho /,/ by whom[\.\?]/
object	/[Ww]hat (did|do|does) /
known-for	/[Ww]hy .+ famous/ /[Ww]hat made .+ famous/
also-known-as	/[Ww]hat(i|\')s (another|different) name /
name-instance	/Name (a|one|some|an) /
location	/[Ww]here(\'s)? /,/ is near what /
date	/([Aa]bout)?(W|w)hen /,/([Aa]bout)?(W|w)(hat|hich) year /
reason	/[Ww]hy /
what:X	–
unknown	–

we use partial parsing to identify the appropriate target, symbolized by `X` in the type. Usually, `what:X` questions are of the form *What NP VP?* or *What NP PP VP?*. After parsing the question, we use the head of the NP following *what* as the target, potentially modified by further constituents from the NP or PP modifying the head. For instance, question 934 from the TREC-10 question set, shown in (5), is assigned `what:plant`, and question 1339, shown in (6), is assigned `what:breed:of dog` as question target.

(5) Material called linen is made from what plant?

(6) What breed of hunting dog did the Beverly Hillbillies own?

If none of the matching strategies described so far is able to assign a target to a question, the question is categorized as `unknown`. As a consequence, none of the answer selection strategies which are particularly suited for the respective question targets can be applied, and a general fall back strategy is used.

2.5 Answer Selection

Given the parsed and annotated top documents returned by FlexIR and given the parsed and classified questions, the actual process of identifying the answer starts. Although the top 100 documents are analyzed, earlier experiments on TREC-9 questions have shown that in some cases focusing on the top 25 or top 50 documents in the answer selection process results in a better performance. Therefore, we restricted ourselves to analyzing the top 100 documents and varied the parameter of documents analyzed during selection over the submitted runs.

While answer selection strongly depends on the question target, a basic strategy common to all question types is to match the dependency structure(s) present in the question with dependency structures in the top documents. More precisely, we try to find a *maximally* matching segment; in our implementation such a segment can be a sentence or a pair of adjacent sentences. Once such a segment has been found, we check whether it contains constituents that fit the appropriate question target. If this is the case, these constituents are marked as potential answers, and the next best matching segment is analyzed, etc.

For this strategy to work, it is important to have a proper matching algorithm that allows for partial matching and also assigns a weight or score to a match that allows to compare and rank different matches.

Matching dependency structures involves three steps: First it has to be checked whether the two heads, i.e., verbs, match, and then the overlap between the arguments of the two structures has to be determined. Since the arguments themselves can be complex phrases, it is necessary to also apply phrase matching on this lower level so as to determine to which extent two arguments match.

There is a number of ways to devise a phrase matching algorithm, although the literature on phrase matching is rather sparse. To our knowledge, there is only one algorithm described in the literature, viz. [7]. Note that phrase matching is different from phrase weighting, cf., e.g., [5, 18], which assigns a weight to a whole phrase but does not deal with partial matches between phrases, which is essential in this context.

Here, we will briefly describe one of our implementations of a phrase matching algorithm which was used for all submitted runs. Given two phrases `p1` and `p2`, the function `phrase_match` returns a real between 0 and 1 as the matching score. Stop words, such as *a*, *the*, *some*, *all*, etc., are removed before the phrases are passed as arguments to `phrase_match`. A pseudo algorithm for `phrase_match` is given in Figure 3.

First, the `if-than-else` statement in lines 2–6 checks

Figure 2: Question targets, plus examples from the TREC-9 and TREC-10 questions.

name the name of a person or an entity in general.
(Q-1094): *What is the name of the satellite that the Soviet Union sent into space in 1957?*

pers-def the function or role of a person
(Q-959): *Who was Abraham Lincoln?*

thing-def further explanation or definition of some entity
(Q-903): *What is autism?*

pers-ident a person fitting some description expressed in the question
(Q-973): *Who was the first governor of Alaska?*

thing-ident thing fitting some description expressed in the question
(Q-988): *What is the oldest university in the US?*

number some kind of numerical expression. Actually, the `number` target is subdivided into different subtypes such as `number-money`, `number-height`, `number-distance`, etc.
(Q-1156): *How many Admirals are there in the U.S. Navy?*

expand-abbr the full meaning of an abbreviation
(Q-1176): *What does I.V. stand for?*

find-abbr the abbreviation for some name
(Q-540): *What's the abbreviation for limited partnership?*

agent name or description of an animate entity
(Q-1239): *Who painted the ceiling of the Sistine Chapel?*

object `object` questions are near-reverses of the `agent` questions. Here, the object of an action described in the question is sought.
(Q-1354): *What did Jesse Jackson organize?*

known-for distinguishing feature of some entity
(Q-207): *What is Francis Scott Key best known for?*

also-known-as alternative name for some entity
(Q-1044): *What is another name for vitamin B1?*

name-instance an instance of some description expressed in the question
(Q-1268): *Name a food high in zinc.*

location location of some entity
(Q-1351): *Where was the first golf course in the United States?*

date date of an event
(Q-1302): *When was the Boston tea party?*

reason reason for an event or fact
(Q-1220): *Why is the sun yellow?*

whether the semantic heads of the two phrases are identical. If this is the case, the initial score is set to 0.5, otherwise, `phrase_match` returns with a matching score of 0. This reflects our strong emphasis on the head of a phrase. Of course this leaves room for other options, such as choosing a different value or not returning immediately if the heads do not match.

Figure 3: Phrase matching algorithm.

```
 1 float phrase_match(phrase p1, phrase p2) {
 2   if(head(p1) = head(p2)) {
 3       score = 0.5;
 4   } else {
 5       return 0;
 6   };
 7
 8   if(length(p1) > length(p2)) {
 9       max_length = length(p1) - 1;
10   } else {
11       max_length = length(p2) - 1;
12   };
13
14   if(max_length = 0) {
15     return score;
16   };
17
18   foreach const ∈ (p1∪p2)\head(p1) {
19     if(const ∈ (p1∩p2)\head(p1)) {
20         score += 0.5/max_length;
21     };
22   };
23   return score;
24 }
```

Lines 8–12 compare the lengths of the two phrases, initializing `max_length`. Since the heads were already compared, they can be neglected and `max_length` is decremented by 1 in line 9 and 11. `max-length` is the maximal number of constituents that the two phrases can have in common. Later on it is used for normalization. If `max_length` equals 0, this means that no constituents other than the heads are to be compared and `phrase_match` returns with the value 0.5, see lines 14–16.

Then, for each constituent occurring in either one of the phrases we check whether it occurs in both phrases (lines 18–22). If this is the case, `score` is incremented by `0.5/max_length`. Finally, line 23 returns the final matching score.

A couple of remarks are in order. First, except for the identification of the head, we do not consider word order; i.e., matching phrases of the form ABC and BAC get a score of 1 although they differ in word order. A side effect is that the distance of a constituent to the head of its phrase is not considered, although one might argue that the closer a constituent is to the head, the more important it is.

Another simplification is the fact that we neglect term importance such as *tf.idf* weighting. Each constituent or term occurring in both phrases contributes equally to the computation of the matching score, even though some terms are obvi-

ously more content bearing than others.

Finally, in the algorithm as it was described above, two constituents are compared with respect to identity. This is a very strict constraint which was softened in the actual implementation of Tequesta. We used WordNet [6] relations, such as synonomy and hyponomy, thus allowing for a match between two constituents if they are in linked by chain of these WordNet relations.

Phrase matching is used in the process of matching dependency structures, which, in turn, helps us to rank matching text segments taken from the top documents. Starting with the highest ranked segment, we apply strategies that depend on the question target to extract the answer string from these segments. In the remainder of the subsection we briefly discuss some of our strategies.

When selecting the answer to a question, we distinguish between the *focus*, or target, of a question and its *topic*. The focus is the element the question is asking for, or put differently, the element lacking. The focus, on the other hand, is the information providing some description or context, the answer should fit into.

Questions of type `pers-def` or `thing-def` ask for the function or role of person and some further explanation or definition of a thing, respectively. Often, this kind of information is contained in an apposition (as illustrated by (8.a)) or a relative clause following the occurrence of this person's name or thing's name (as illustrated by (8.b)).

(7) Who is Desmond Tutu?

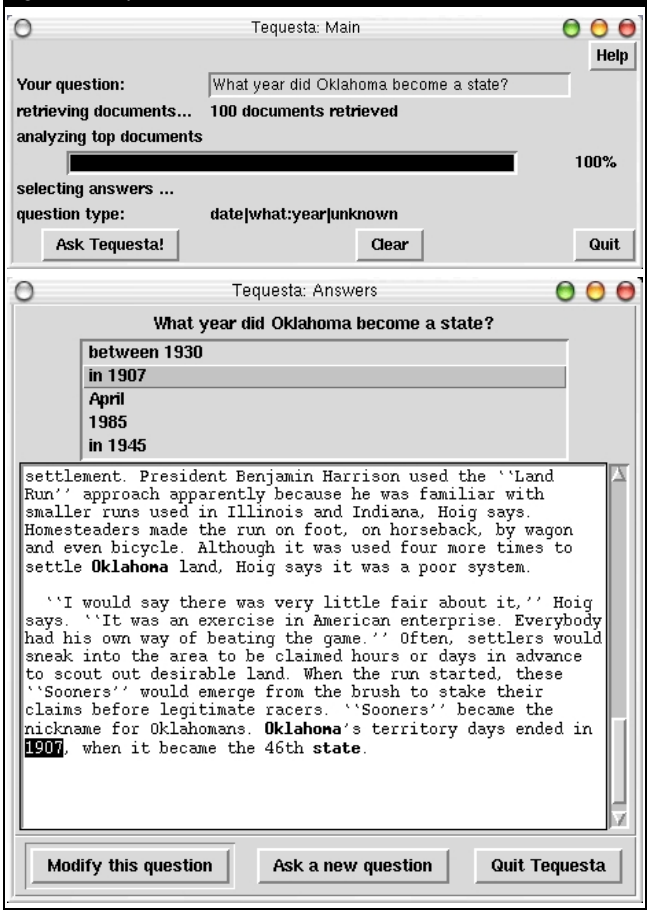
(8) a. Tutu, winner of the 1984 Nobel Peace Prize
b. Desmond Tutu, who is a member of Harvard University's governing board

In order to make sure that the apposition or relative clause forming the potential answer contains descriptive information rather than some other information we apply further heuristics. For instance, a potential answer is preferred if it contains superlative adjectives, such as *first*, *highest*, *most*, etc., or nouns ending in *-er* which are likely to describe some role, e.g., *winner*, *member*, etc.

Questions of type `agent` ask for an animate entity, such as a person or organization, being the logical agent of an event described in the question. If the dependency structure from the question matches a dependency structure from a document and there is an animate NP in subject position (positive sentence) or within a PP headed by the preposition *by*, we take this to be the logical agent. Of course, such an NP is disregarded if it already occurs in the question itself. Questions of type `object` are dealt with analogously.

Questions of type `what:X` are particularly interesting because they are very frequent (at least in the TREC data) and explicitly require some lexical knowledge base. Questions of type `what:X` ask for something that is a kind of `X` and that fits the further description expressed in the remainder of the question. For example, question 429, given in (9), asks for something which is a university.

Figure 4: Tequesta's user interface.



(9) What university was Woodrow Wilson President of?

In (9) *university* is the focus of the question and the further constraint *was Woodrow Wilson President of?* is the topic of the question. In order to establish the relationship between an entity found in a matching dependency structure and the predicate *university* it is necessary to access a lexical knowledge base. Tequesta exploits WordNet for this purpose. In particular, WordNet's hyponym relations are used.

While extracting potential answers, we also keep track of the number of steps that had to been taken while traversing WordNet, and the matching scores that were involved. The higher the matching scores and the smaller the number of lexical relations that had to be used from WordNet, the higher the overall answer score of a potential answer. Finally, the extracted answer strings are ordered and the top five are selected as the final set of answers.

Tequesta also provides a graphical user interface which we use for evaluation and demonstration purposes. Figure 4 shows the two windows that are used to interact with the user. The top window in Figure 4 is the main window; it allows the user to enter a question and provides information on the status of the subtasks involved in answering the question. The bottom window presents the results; in the upper part the extracted answer strings (at most 50 bytes long) are listed

and by clicking on them the answer document is displayed. Words occurring in the answer are high-lightened by reversing foreground and background color, and words occurring in the question are displayed in bold face; this is done to facilitate the search for justifications of the extracted answer.

3 Main Task

The main QA task in TREC-10 is similar to the main tasks in TREC-8 and TREC-9. The document set consists of data sets taken from Disks 1–5 of the TIPSTER/TREC document CDs. A total of 500 questions is provided that seek short, fact-based answers. Some questions are not known to have an answer in the document collection. At least one and no more than five ranked responses per question ranked were to be returned for each question, where the first response is to be preferred over the other responses. A response is either a [answer-string, docid] pair or the string "NIL," where the answer-string may contain no more than 50 bytes and the docid must be the id of a document in the collection that supports the answer-string as an answer.

An [answer-string, docid] pair is judged *correct* if the answer-string contains an answer to the question, the answer-string is responsive to the question, and the document supports the answer. If the answer-string is responsive and contains a correct answer, but the document does not support that answer, the pair will be judged "unsupported" and the pair will only contribute towards the "lenient" score, not to the "strict" score. Otherwise, the pair is judged incorrect.

As with TREC-8 and TREC-9, the score assigned to each question is the reciprocal of the rank for the first response to be judged correct, or 0 if no response is judged correct. The total score for a run is the mean reciprocal rank (MRR) over all questions.

3.1 Submitted Runs

We submitted three runs for the main task (`UAmsT10qaM1`, `M2`, and `M3`). Each of our runs employed the Tequesta system, which was given a total of 979,678 documents to index. The runs differed along 2 dimensions: the number of documents used as input for the answer selection process (either 25 or 50 documents), and the size of the text segments that were used to match the question during the answer selection process (either a single sentence or 2 consecutive sentences); see Table 3.

3.2 Results and Discussion

Of the 500 questions that were originally released, eight questions were removed from the evaluation due to various problems with those questions. Table 3 summarizes the statistics for each of our three submitted runs (`UAmsT10qaM1`, `M2`, and `M3`) over the (remaining) 492 questions.

Table 3: Summary of the results for the main task.

UAmsT10qa...	M1	M2	M3
Top documents used	25	50	25
# Sentences in segments	1	1	2
MRR strict	0.185	0.183	0.190
MRR lenient	0.197	0.196	0.203

As Table 3 indicates, it is unlikely that there are significant differences between the MRRs for the three runs that we submitted for the main task. Despite this, we took a closer look at the difference between `UAmsT10qaM2` and `UAmsT10qaM3`. We first ordered the questions with respect to the individual reciprocal ranks from run `UAmsT10qaM2` and, in case they were identical, with respect to the question's id. Then, we marked the extent to which run `UAmsT10qaM3` differs from run `UAmsT10qaM2` for each question. Figure 5 shows the differences for the first 164 ordered questions.

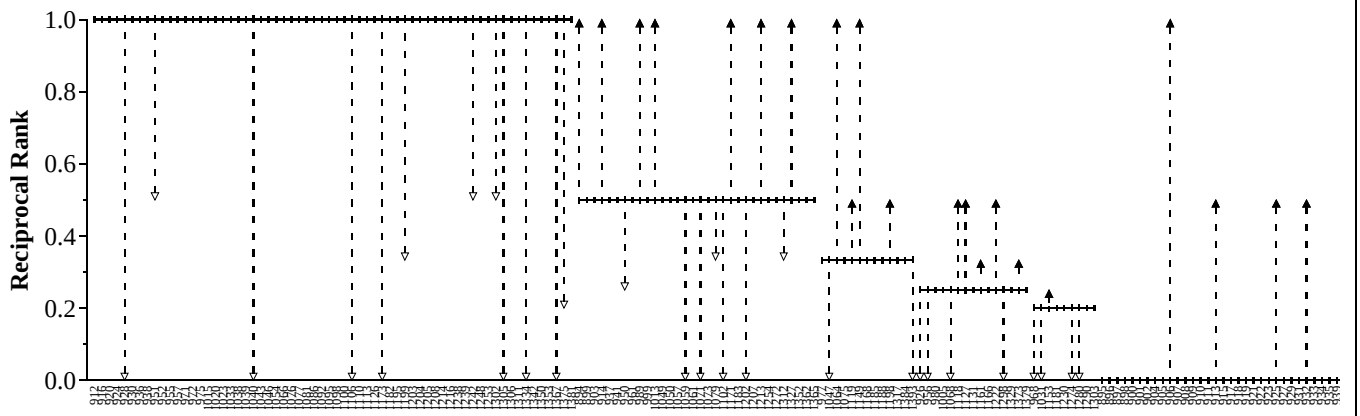
Although the overall effectiveness of run `UAmsT10qaM3` increased by only 3.86% in comparison to run `UAmsT10qaM2`, it is by no means consistently spread over the questions. For many questions there is a severe decrease in effectiveness. What causes this decrease for some questions is not clear to us at the moment, but we hope to gain further insights by analyzing the results more carefully.

Table 4: Analysis of the scores for `UAmsT10qaM3`.

Question class	#	MRR	Diff.	Rel. Con.
`name`	9	0.111	−41.5%	0.002
`pers-def`	3	0	−100%	0
`thing-def`	110	0.254	+33.8%	0.057
`pers-ident`	22	0.167	−12.3%	0.007
`thing-ident`	107	0.196	+3.30%	0.043
`number`	35	0.267	+40.4%	0.019
`expand-abbr`	4	0.125	−34.2%	0.001
`find-abbr`	0	N/A	N/A	N/A
`agent`	21	0.071	−62.4%	0.003
`object`	18	0.069	−63.5%	0.003
`known-for`	0	N/A	N/A	N/A
`also-known-as`	11	0.273	+43.5%	0.006
`name-instance`	2	0	−100%	0
`location`	27	0.272	+43.0%	0.015
`date`	41	0.250	+31.6%	0.021
`reason`	4	0	−100%	0
`what:X`	71	0.093	−50.8%	0.013
`unknown`	7	0	−100%	0
Total	492	0.190		

Table 4 provides a closer look at our best run for the main task, `UAmsT10qaM3`, and a breakdown in terms of the individual question types. Column 1 lists the question classes as discussed in Section 2.4; column 2 lists how many of the 492 questions belonged to a particular class. According to our question classifier two classes did not have any questions in this year's set of questions: `find-abbr` and `known-for`. Column 3 lists the mean reciprocal rank for each class of ques-

Figure 5: Run UAmsT10qaM2 vs. run UAmsT10qaM3.



tions. Column 4 ("*Diff.*") records the relative difference between the MRR for the class and the overall MRR for the run, while column 5 ("*Rel. Con.*") indicates the relative contribution of the question class.

The relative contribution of a question class is the MRR for the class multiplied by the proportion of the questions in that class. For example, if a class has an MRR of 0.25, and 10% of all the questions were in that class, the relative contribution would be $0.25 \times 0.10 = 0.025$. For development purposes it can be especially helpful to record differences in MRR and/or relative contribution. Differences in MRR give an indication of how well a question class was handled. Changes in relative contribution give an indication of how much this matters, and therefore where efforts should be focussed to alter the system's performance.

It is clear from Table 4 that our overall score for UAmsT10qaM3 is strongly positively influenced by our scores on the following classes: thing-def, thing-ident, number, location, and date, while our performance on pers-ident, agent, object, and, especially, what:X, contributed negatively towards our overall score.

4 List Task

TREC-10 featured a new task, the QA list task, where answers are to be collected from multiple documents. The list task consisted of 25 questions in the same format as the main task. Each list question specifies a number of instances to be retrieved; e.g., 10 flavors of ice cream in question 11, shown in (10).

(10) Name 10 different flavors of Ben and Jerry's ice cream.

Participants were not allowed to return more instances than specified in the question.

We modified Tequesta only minimally for this task. Since questions in the list task are typically looking for instances of some description, all questions were classified as what:X type questions. The major difference with the main task is that answers are collected from several documents. When compiling the list of answers we checked for duplicates and near duplicates by using simple techniques such as word overlap while ignoring stop words.

In the list task, the answers returned are not ranked. Performance is measured in terms of accuracy, which is computed as the number of distinct correct instances divided by the number of instances requested in the question. Table 5 summarizes the results for the two submitted runs.

Table 5: Summary of the results for the list task.

Runs	Avg. Accuracy
UAmsT10qaL1	0.12
UAmsT10qaL2	0.13

The strategies for run UAmsT10qaL1 and UAmsT10qaL2 only differ minimally from each other. Run UAmsT10qaL1 uses the top 50 documents to compile the answer list whereas run UAmsT10qaL2 uses the top 25 documents. This similarity between the runs probably also explains the small difference in performance (+8.33%).

5 Conclusions

In this paper we presented our question answering system Tequesta and evaluated its performance in the TREC-10 QA task. Clearly, Tequesta is still in its early stages and our participation in the TREC-10 QA task was very helpful in revealing aspects that need additional attention in future developments of the system. Most of the shortcomings were already discussed in more detail throughout the paper and we will just summarize some of them here.

First, the underlying information retrieval system FlexIR that was used for pre-fetching is not tuned for the overall task of question answering. Integrating further constraints into the retrieval process, such as phrase-indexing, locality, and Boolean operators, might help in formulating more structured queries that will increase the density of documents containing

an answer in the set of top documents.

One of the main problems of the named entity recognizer was that it does not allow for multiple semantic types, which results in a high error rate when using gazetteers to assign certain semantic types, such as locations and person names. In addition, we plan to include the annotated semantic types into the index which is used for retrieval.

Of course, improving the answer selection component remains the main challenge. Table 4 shows that there are significant differences in performance between the question types. Especially the performance for questions of type `agent`, `object`, and `what:X` is far below the average performance of the system.

In this year's participation, we did not spend much time or effort on customizing Tequesta for the list task, but we plan to further develop this aspect of our question answering system, as the problem of fusing information from different sources — in QA as well as in related areas such as multi-document fusion [12] — strikes us as an interesting challenge.

Acknowledgments

Christof Monz was supported by the Physical Sciences Council with financial support from the Netherlands Organization for Scientific Research (NWO), project 612-13-001. Maarten de Rijke was supported by the Spinoza project 'Logic in Action' and by grants from NWO under project numbers 612-13-001, 365-20-005, 612.069.006, 612.000.106, and 220-80-001.

References

[1] J. Allan. NLP for IR. Tutorial presented at NAACL/ANLP language technology joint conference, April 29, 2000.

[2] R.H. Baayen, R. Piepenbrock, and L. Gulikers. The CELEX lexical database (release 2). Distributed by the Linguistic Data Consortium, University of Pennsylvania, 1995.

[3] D. Bikel, R. Schwartz, and R. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 1(3):211–231, 1999.

[4] C. Buckley, A. Singhal, and M. Mitra. New retrieval approaches using SMART: TREC 4. In *Proceedings TREC-4*, pages 25–48, 1995.

[5] J. Fagan. *Experiments in Automatic Phrase Indexing for Document Retrieval: A Comparison of Syntactic and Non-Syntactic Methods*. PhD thesis, Department of Computer Science, Cornell University, 1987.

[6] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[7] G. Galbiati. A phrase-based matching function. *Journal of the American Society for Information Science (JASIS)*,, 42(1):36–48, 1991.

[8] S. Harabagiu, J. Burger, C. Gardie, V. Chaudri, R. Gaizauskas, D. Israel, C. Jacquemin, C.-Y. Lin, S. Maiorano, G. Miller, D. Moldovan, B. Ogden, J. Prager, E. Riloff, A. Singhal, R. Shrihari, T. Strzalkowski, E. Voorhees, and R. Weishedel. Issues, tasks, and program structures to roadmap research in question & answering (Q&A). URL: `http://www-nlpir.nist.gov/projects/duc/roadmapping.html`, October 2000.

[9] J.B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1–2):22–31, 1968.

[10] C. Macleod, R. Grishman, A. Meyers, L. Barrett, and R. Reeves. NOMLEX: A lexicon of nominalizations. In *Proceedings EURALEX'98*, 1998.

[11] C. Mikheev, M. Moens, and A. Grover. Named entity recognition without gazetteers. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL'99)*, pages 1–8, 1999.

[12] C. Monz. Document fusion for comprehensive event description. In M. Maybury, editor, *Proceedings of the ACL 2001 Workshop on Human Language Technology and Knowledge Management*, 2001.

[13] C. Monz and M. de Rijke. University of Amsterdam at CLEF 2001. In *Proceedings of the Cross Language Evaluation Forum Workshop (CLEF 2001)*, pages 165–169, 2001.

[14] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[15] J. Prager, E. Brown, A. Coden, and D. Radev. Question-answering by predictive annotation. In *Proceedings ACM SIGIR 2000*, pages 184–191, 2000.

[16] B. Santorini. *Part-of-speech tagging guidelines for the Penn Treebank*. Department of Computer Science, University of Pennsylvania, 3rd revision, 2nd printing edition, 1990.

[17] H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing*, 1994.

[18] T. Strzalkowski. Natural language information retrieval. *Information Processing & Management*, 31(3):397–417, 1995.

[19] E.M. Voorhees. Overview of the TREC-9 question answering track. In *Proceedings TREC-9*, 2001.

[20] E.M. Voorhees and D.M. Tice. The TREC-8 question answering track evaluation. In *Proceedings TREC-8*, pages 83–105, 2000.

[21] E. Williams. On the notions 'lexically related' and 'head of a word'. *Linguistic Inquiry*, 12:245–274, 1981.