qa42: Web-Based Question Answering System

CS224N Final Project

Jyotika Prasad
jyotika@stanford.edu

Antone R. Vogt
ubiquity@stanford.edu

Riku Inoue
rikui@stanford.edu

1. Introduction

qa42, named in memory of Douglas Adams [8], is an open domain question-answering system that builds upon and extends prior work in [1] by exploiting the redundancy of the world wide web [9]. qa42 returns specific answers to factoid questions rather than summaries as is done in search engines and traditional question-answering systems. The current version of qa42 is geared to answer only questions for persons, organizations, locations, dates, and quantities. Additional types of questions are planned for future work.

An outline of the process is diagrammed in Figure 1. The question to be answered is written into one or more search engine queries, which are then sent to the Google search engine [10]. Summaries returned by Google are scored against answer models also generated from the question. Similar viable answers are clustered together and rescored based upon frequency. qa42 presents the three answers that score the highest.

For the reader's convenience, examples and several tables are presented at the end of this document. We now present the details.

2. Question Processing

2.1. Parse Analysis

As a first step, the `qa42.query.Question` Java class parses the question using [2] as the parser trained on data from [11]. From the resultant parse tree, it ascertains the pronoun type, pronoun subtype, main verb, subject noun phrases, and object noun phrases.

The *pronoun type* is defined as *who/whom*, *where*, *when*, *why*, *how*, *which*, *what*, and *other*. In general, the pronoun type is taken to be the interrogative pronoun used in the question. In the case where more than one interrogative pronoun appears in the question, the outermost from the parse tree is used.

The *other* category is used when no such interrogative pronoun exists in the question. This is commonly the case when the question is worded as an imperative. An example is: *Name the designer of the shoe that spawned millions of plastic imitations, known as "jellies".* As such, qa42 does not perform particularly well on imperatives. The fact that [11] contains relatively few imperatives also impacts the effectiveness of analyzing such sentences.

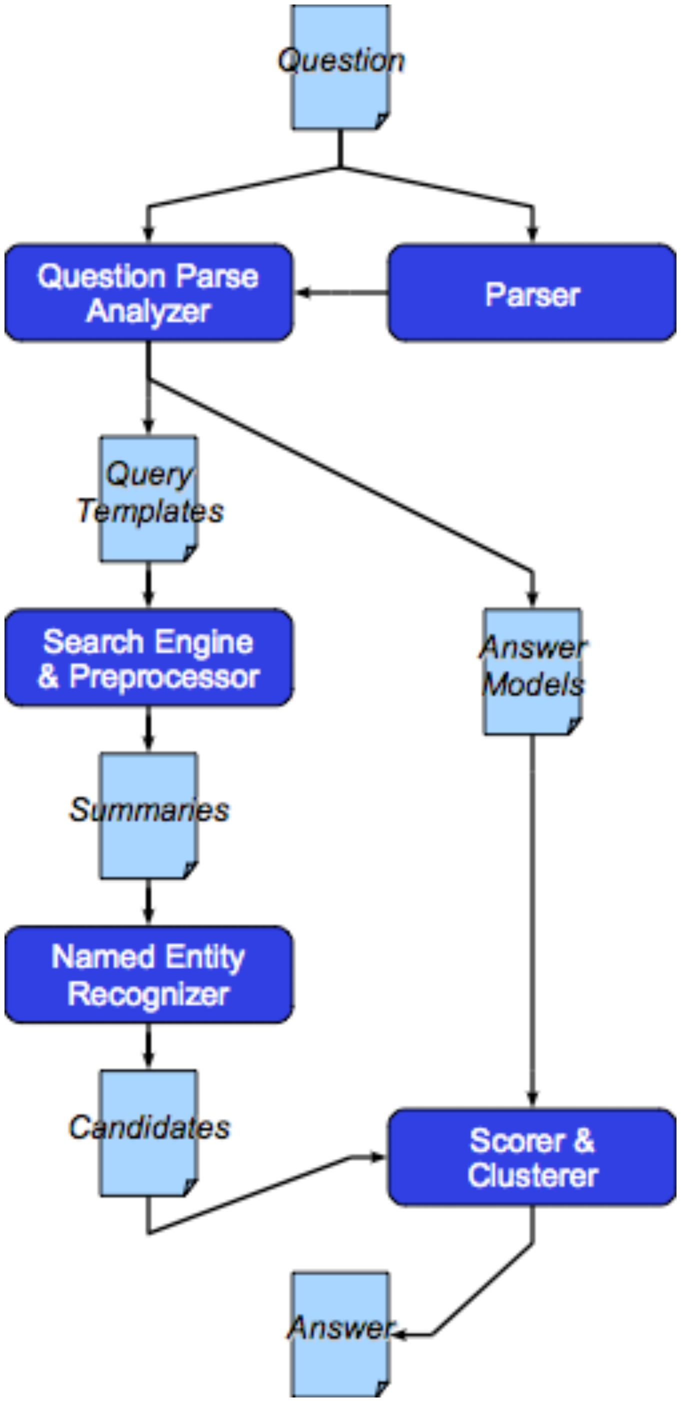

Figure 1. qa42 process outline.

The pronoun subtype is defined as the subordinate phrase or clause that is headed by the interrogative pronoun. Thus, the pronoun type and subtype respectively are *what* and `company` in this example: `What company is the largest Japanese ship builder?`

Since the *what* and *which* pronoun type categories give little clue as to the intended answer form, qa42 converts these categories to either *where* or *when* in the case that the subtype respectively indicates a location or time. qa42 contains a hard-coded set of 42 location words and 40 time words respectively [6] represented by the `qa42.word.LocationList` and `qa42.word.TimeList` classes.

We use minimal analysis of pronoun types and subtypes since it is not the major focus of our experiment. Deeper evaluation of these natural language features is reserved for future work.

The *main verb* is defined as the word heading the outermost verb phrase identified by the parser that is not a form of `do` or `have`. In the case of these *helper verbs*, the tense and number is noted so that the main verb can be represented either in its original form or in a *revised* form to match the helper. For example, the main verb is `die` and the revised main verb is `died` in this question: `When did Nixon die?` Similarly, the main verb is `mean` and the revised main verb is `means` in this example: `What does El Nino mean in Spanish?`

The main verb is also converted to a past participle form for the purpose of forming passive voice phrases in query templates (§2.2).

The `qa42.word.VerbFormConverter` Java class makes these conversions. It contains a set of 388 past participle mappings and 344 tense mappings to handle irregular verbs. The list of irregular verbs was reduced from an exception list in [3]. For all other words, qa42 applies heuristics based on standard rules of English.

The *subject* and *object noun phrases* are defined as any and all noun phrases identified by the parser that respectively come before or after the main verb. As such, `El Nino` and `Spanish` are the subject and object nouns phrases in the earlier example: `What does El Nino mean in Spanish?`

The effect of this processing is that words with little semantic value, such as prepositions, are dropped. Additionally, since our definition of *subject* and *object* are more general than standard English, it is frequently the case that there are multiple subjects and/or object to be identified. For example, since the main verb is `ask`, both `the FBI` and `a word processor` are object noun phrases in this question: `Why did David Koresh ask the FBI for a word processor?` The fact that we have multiple phrases becomes significant in generating answer models (§2.3).

Each quoted phrase is taken as a simple noun phrase regardless of its parsing. The assumption is that the quoted phrase refers to a single title, quote, etc. Quoted phrases are handled specially in template generation (§2.2).

2.2. Query Templates

Once parse analysis is complete, qa42 generates a list of one or more *query templates*. From each template, qa42 generates a single search engine query and a (possibly empty) set of answer models (§2.3), the latter of which are matched against search engine summaries. These templates are generally represented by objects of the `qa42.query.Template` Java class.

Each template represents a sequence of phrases with at most one *pronoun position marker*. This marker indicates the place within the phrase sequence in which the answer expected. It is used in generating answer models but not in the formatting of search-engine queries.

As a typical example, the question `What state does Charles Robb represent?` results in this template (among others): *pronoun*, `state`, `represented`, `Charles Robb`. When this template is formatted for the Google search engine, the pronoun is dropped, so the query reads as follows: `state represented Charles Robb.` Since the order of words is significant [10], Google is likely to match a search-engine summary such as `Virginia is the southern state represented by Charles Robb`, assuming that it exists.

qa42 always generates one *simple query template* for each question, which is the verbatim text of the question itself. Since this special template is not the result of analysis, it is not broken into phrases, does not contain a pronoun, and therefore is not used to generate answer models (§2.3). A special `qa42.query.SimpleTemplate` Java class, which is a subclass of `Template`, is used to handle this special case.

Depending on the structure of the question parse, qa42 may generate as many as eight additional templates per question, each one with a pronoun position marker. Table 7 (near the end of this document) summarizes the possible formats of these templates, along with examples of matching sentences from a potential search engine summary. Since many of these formats differ only in the placement of the pronoun position marker, duplicate search-engine queries are discarded.

In general, search-engine queries do not contain quotes or special operators. The exception applies when phrases are quoted in the original question. In this case, the same phrase is quoted in the query under the assumption that it refers to a title or quotation.

2.3. Answer Models

qa42 generates several *answer models* from each query template (other than simple templates) with the purpose of predicting and scoring search-engine summaries.

With some exception, the set of answer models is the set of all possible subsequences of the phrases from the template from which it is generated. That is to say, qa42 creates an answer model by including or excluding each template phrase. The exceptions are that qa42 never generates an empty answer model nor a model with no pronoun.

Therefore, the number of models *m* generated from a single template with *n* phrases is:

$$m = 2^{n-1} - 1 \qquad (2.1)$$

since *n*–1 represents the number of phrases other than the pronoun position marker.

Each model is ranked by the ratio of the number of phrases it contains to the number of phrases contained by its template. That is, for a model with *n'* phrases and a template with *n* phrases, the specificity rank *r* is given by:

$$r = \frac{n'}{n} \qquad (2.2)$$

For example, the template *pronoun*, *state*, *represented*, *Charles Robb* results in these models:

pronoun, state, represented, Charles Robb	[1.00]
pronoun, represented, Charles Robb	[0.75]
pronoun, state, Charles Robb	[0.75]
pronoun, state, represented	[0.75]
pronoun, Charles Robb	[0.50]
pronoun, represented	[0.50]
pronoun, state	[0.50]

When reviewing this in the context of Table 7, it is useful to keep in mind that each *subject* or *object* may represent several phrases. For example, in the question *Why did David Koresh ask the FBI for a word processor?*, there are one subject phrase, *David Koresh*, and two object phrases, *the FBI* and *a word processor*. Therefore, from the template *the FBI, a word processor, asked by, David Koresh, pronoun*, 15 models are generated.

Since there are generally multiple templates per question, there is a potential for duplicate answer models. Consequently, qa42 eliminates the duplicates.

Since the number of answer models is a function of the number of templates and the number of phrases in each template, it can be thought of as a function of the question complexity and the length of the sentence. As such, some of the questions can have a fairly large number of models. For example, *How much did Manchester United spend on players in 1993?* results in 8 non-simple query templates and 158 answer models due to the complexity. In contrast, *What is the name of the rare neurological disease with symptoms such as: involuntary movements (tics), swearing, and incoherent vocalizations (grunts, shouts, etc.)?* results in 4094 models from only one template.

3. Information Retrieval

3.1. Search Engine Module and Google

The *search engine module* of qa42 sends the queries generated from the query templates to Google [10]. The results returned from the search engine are preprocessed (§3.2) before being passed to the named entity recognition module (§3.3).

qa42 uses the Google SOAP API to retrieve the query results. qa42 processes only the page summaries returned by Google and not the referenced pages. This improves on temporal performance, since looking at these pages would involve separate network URL requests and introduce a bottleneck. Also, under the assumption is that the answer appears close to the query phrases, the summary proves to be sufficient. For every query, we request a maximum of ten results per query from Google.

3.2. HTML Preprocessor

qa42 *preprocesses* the summaries returned by Google to transform the HTML into a more usable format. The preprocessor starts by removing HTML tags such as `<b>` and character references such as `&#39`, since these data elements carry little or no natural language information.

The preprocessor also inserts white space between adjacent digit and non-digit characters to aid the named entity recognizer (§3.3) in identifying quantities. For instance, *One Big Mac costs 24EEK in Estonia, 1$ = 13.64EEK* is converted to *One Big Mac costs 24 EEK in Estonia, 1 $ = 13.64 EEK*.

3.3. Named Entity Recognizer

The *named entity recognizer* (NER) module analyzes the preprocessed search-engine summaries to extracts candidate answers. Specifically, the Stanford NER [4], which was trained on three corpora [12,13,14], is used to identify *person*, *organization*, and *location* entities over the summaries for *who* and *where* pronoun types. qa42 also uses an augmented NER for *when* and *how* pronoun types. The current version of qa42 does not handle other types of pronouns.

This logic is augmented for *when* and *how* pronoun types since [4] does not provide sufficiently fine-grained entity types in the cases. qa42 uses logic based on regular expression to identify *date* and *quantity* entities. The approximating assumption here is that *when* pronoun types usually refer to a date (as opposed to a time) and *how* types usually refer to a quantity (as in *how much* or *how far*).

The augmented NER module contains the `DateMatcher` Java class, which extracts from the search-engine summaries phrases with patterns that indicate a date or year. Examples include *1776*, *July 4th 1776*, *07/04/76*, and *4th* *of July*. This logic searches the summary string for dates using regular expressions, then reformats these candidates into a common American format of *July 4, 1776* to simplify clustering (§4.2).

Quantities are extracted using a simple rule. qa42 defines a quantity to be any number with a unit, such as *$ 40,000*, *20 meters*, *30 ft*, etc. Several lists of units for distance, area, money, etc. [7] are hard-coded into qa42 to facilitate this process. This module has room for further development but is not the focus of our project.

From each of these NER modules, qa42 extracts entities that match the pronoun type as specified in Table 1. For example, if the question has a *who* pronoun type, the NER module returns a list of all persons and organizations found in the search-engine summaries. All such entities are sent to the scoring module as *candidate answers*.

4. Candidate Analysis

4.1. Scorer

The *scoring module* scores the candidate answers as a function of the number of times the candidate appears among and the answer model (§2.3) it matches.

We experimented with basic scoring, which simply counts the number of times c that a particular answer occurs across all the search-engine summaries. In many cases, this fails to give the correct answer, since there are examples where a particular incorrect candidate occurs more frequently than the correct answer.

To compensate for this, qa42 uses a scoring function which approximately captures the semantic corrects of a candidate. The answer models, which are generated during the parse analysis phase, represent an ordering of the phrases that we expect in the answer. The models do not contain most stop words, allowing for some flexibility in scoring.

The list of models are ordered by specificity rank r (Equation 2.2). For each candidate, the list is searched starting with the highest specificity rank until a match is found. If there is no match, r=0.2 is used. Each candidate answer is given a score $s=r$.

This differs from other question answering systems such as AskMSR [15] in that it scores each candidate based on the query that was used. Contrastingly in qa42, the answer models used in scoring are related to the query only in that they are generated from the same templates. However, qa42 makes no attempt to trace a particular model or candidate back to the query or template. In other words, the score of a candidate depends on the specificity of the predicted answer model rather than the query from which it came.

4.2. Clustering Module

The clustering module aggregates the candidates in a manner identical to [1]. In the first of two passes, qa42 merges candidates that are the same (ignoring case and white space) and sums their scores.

In the second pass, a candidate is merged into a larger candidate if the former is a subsequence of the latter. As in the first pass, the score of the merged candidate is the sum of the other two. Therefore, in either pass:

$$s' = \sum s \qquad (4.2)$$

where s' is the score of the cluster and s are the scores of the candidates which are merged together.

The top three scoring candidate are returned as answers.

5. Conclusion

5.1. Evaluation Framework

We test the system on the questions from the TREC 8 [5] question bank, which are classified by one of the categories listed in Table 1. Other classifications are outside the purview of our experiment.

Pronoun Types	NER Entity Types	TREC-8 Classifications
who	*person* *organization*	*person* *organization*
where	*location*	*location* *city* *country*
when	*date*	*duration* *time* *date*
how	*quantity*	*money* *distance* *age* *measure* *measurement* *age*

Table 1. Relation among pronoun types, NER entity types, and TREC-8 classifications.

Our evaluation has two criteria: (i) *accuracy*, the ratio of questions where qa42 presents the correct answer as its most likely candidate, and (ii) *mean retrieval rate* (MRR), which is defined as follows: For each question, 1 point is awarded if the first answer is correct, 0.5 points if the second answer is correct, and 0.333 if the third answer is correct. Zero points are awarded for candidates rated as fourth or lower. The MRR is the mean average of these points.

We used four testing *configurations*: the full qa42 system, qa42 with nonsimple query templates disabled, qa42 with answer-models based scoring disabled, and the baseline with both of these features disabled. When the nonsimple query templates are disabled, the only query sent to Google is the original verbatim question. When the answer models are disabled, the scorer uses only the counts (i.e. s=1 rather than s=r).

For each configuration, we have two sets of results: one on all questions from TREC-8 [5] and another only on questions that we classified correctly. The second set of results is important since misclassification of the question type, which is a very significant source of error, is not the focus of our project. Thus it is reasonable to evaluate the system separately for those questions where the classification is correct.

5.2. Results

The results presented here are accurate as of June 7, 2007. Since Google is dynamic, subsequent rerun of this data may have slightly differing results.

Table 2a shows the accuracy and MRR per TREC-8 classification, while Table 2b presents the same results for those questions that we classified correctly. qa42 performs best on questions with *who* and *where* pronouns, followed by *when* pronouns, finally by *how* pronouns. This is directly related to NER accuracy on each pronoun type, so we conclude that the augmented NER module has significant impact on performance. On the pronoun types where the NER module is strong, the MRR is fairly high, around 0.6 in general.

Question misclassification is the primary source of error. To evaluate the system without misclassification, we ran an experiment where the correct question type was given to the question. The results are presented in Table 3. As expected, this improved both the MRR and accuracy metrics, especially for questions with *how* pronouns, which are the most difficult to classify.

The next step in analysis was to determine the degree to which the query templates and the answer-model based scoring helped. To this end, we experimented with the configurations described above (§5.1). As can be seen from Tables 4-6, query templates improve performance by approximately 2-3%.

In contrast, answer-model based scoring on its own is deleterious to performance, but is beneficial when combined with query templates. Since the lack of templates results in fewer candidate answers, we conclude that simple counting works better with sparse summary sets. Additionally, since the answer models are generated from the templates, summaries returned by Google are not as likely to contain appropriate matches to the models.

5.3. Analysis and Conclusion

We identified three specific areas where significant improvement is possible: *quantity* entity types, the NER, and the parser. However, we further conclude that the mechanism of question parsing does improve the performance of qa42.

The results show that questions in the *quantity* entity type perform most poorly. We surmise that this effect is due to the following factors: (i) The NER is the weakest for the *quantity* type and is critical for accuracy in our system. (ii) Clustering does only basic string comparisons and is therefore extremely weak for quantities. (iii) Shallow analysis of the pronoun subtype for this entity type weakens the value of the query template and answer models.

Some errors would be resolved if the augmented NER were more fine-grained. For *who* pronoun types, we look both for *person* and *organization* entity types, which leads to errors in examples such as *Which company created the Internet browser Mosaic?*. In this case, the correct answers are *University of Illinois National Centre for Supercomputing Applications* and *NCSA / Netscape Communications*, according to [5]. Because qa42 does not distinguish persons from organizations, it answers *Marc Andreessen*, the person who created Mosaic.

The accuracy of qa42 depends on good parse analysis and in certain cases this process goes awry. The parse analyzer depends on a reasonably valid parse of the question and an error in parsing throws it off. This is especially noticeable in the case of imperatives as discussed above (§2.1). This is expected since the training data [3] used for the Stanford Parser [2] contains few imperatives.

The question parse analysis in qa42 performs well for questions such as *Who killed John Lennon?*. According to our experiments, without parse analysis, qa42 returns the answer *Fenton Bressler*, who has written a book titled *Who killed John Lennon?* [16]. Since qa42 with parse analysis submits *John Lennon killed by* as a query and *John Lennon, killed by*, *pronoun* as an answer model, *Mark David Chapman*, the correct answer, scores higher.

Thus we conclude that our basic concept is valid and that parse analysis is key to identifying text phrases from the search-engine summaries which contain likely answers.

5.4. Future Work

Future work on qa42 would include (i) more aggressive interpretation of quantities to improve clustering (§3.3), (ii) deeper analysis of pronoun types and subtypes (§2.1), and (iii) handling additional question classifications (§5.1).

An relatively straight-forward but significant enhancement to the augmented NER would improve resultant clustering by standardizing quantities in much the same way as dates. That is, the unit of measure would always follow the quantity. For example, *$ 400* would be rewritten as *400 dollars*.

Additional performance gains may also be realized by interpreting words such as *million* and translating units, such as feet to meters. Thus *4 million* becomes *4000000*, *12 in* becomes *1 foot*, and *3.28 feet* becomes *1 meter*. This also implies the interpretation of common abbreviations and varied unit notations, and pluralization or singularization as needed.

The current version of qa42 employs a minimal analysis of pronoun types and subtypes. Better analysis could potentially improve the generation of query templates and answer models. For example, *How far is Yaroslavl from Moscow?* results in a template of *pronoun*, *far*, *Yaroslavl*, *Moscow* since the parse analysis blindly inserts the query subtype *far* into the template. Logic to apply context to determine that this question is about distance would allow qa42 to use templates that include appropriate units of measure.

qa42 currently can only return answers that fall into a handful of entity types. The extension we plan for this would work as follows: If the answer type as determined from the parse analysis module is not any one of the known entities, the qa42 would simply mine *n*-grams and return them as candidates, which could then be tiled using a straight-forward greedy tiling algorithm similar to AskMSR [15]. In this way, qa42 would benefit from its intelligent approach when possible and fall back upon a more basic approach as needed.

References

[1] Vincenzo Di Nicola and Jyotika Prasad. 42: A Web Based Question Answering System, 2006.

[2] Dan Klein and Christopher Manning. The Stanford Parser: A Statistical Parser. http://nlp.stanford.edu/downloads/lex-parser.shtml.

[3] George A. Miller, et al. WordNet: a lexical database for the English language. http://wordnet.princeton.edu.

[4] Jenny Finkel, Dan Klein, and Christopher Manning. Stanford Named Entity Recognizer. http://nlp.stanford.edu/software/CRF-NER.shtml

[5] Text Retrieval Conference (TREC-8). National Institute of Standards and Technology. 1999. http://trec.nist.gov/data/qa/t8_qadata.html.

[6] Roget's New Millennium Thesaurus, First Edition (v 1.3.1). Lexico Publishing Group. 02 Jun. 2007. http://thesaurus.reference.com.

[7] Wikipedia, http://en.wikipedia.org.

[8] Douglas Adams, The Hitchhiker's Guide to the Galaxy, BBC Radio 4, 1978.

[9] S. Dumais, M. Banko, E. Brill, J. Lin, and A. Ng. Web Question Qnswering: Is More Always Better? *Proceedings of the 25th Annual international ACM SIGIR Conference on Research and Development in information Retrieval*, pp. 291–298, ACM Press, New York, August 2002, http://doi.acm.org/10.1145/564376.564428.

[10] Google search engine. http://www.google.com.

[11] Mitchell P. Marcus, Mary Ann Marcinkiewicz, Beatrice Santorini. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19 (1993), 313-330.

[12] Message Understanding Conference (MUC) 7 Corpus. Linguistic Data Consortium. Philadelphia. 2001. LDC2001T02.

[13] ACE 2004 Multilingual Training Corpus. Linguistic Data Consortium. Philadelphia. 2004. LDC2005T09.

[14] Conference on Computational Natural Language Learning, CoNLL. Linguistic Data Consortium. Philadelphia. 2005. LDC2005E43.

[15] Eric Brill, Susan Dumais, and Michele Banko. An analysis of the AskMSR question-answering system. *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing – Volume 10*. Association for Computational Linguistics, Morristown, NJ, 257-264. 2002. http://dx.doi.org/10.3115/1118693.1118726

[16] Fenton Bresler. Who killed John Lennon? St. Martin's Press. New York. 1989.

Entity Type	MRR Score	Accuracy
quantity	0.0811	0.0541
date	0.4321	0.3333
location	0.3984	0.2927
person/organization	0.5464	0.4754
Average	0.3876	0.3133

Table 2a. Results from all TREC-8 questions with full qa42 system.

Entity Type	MRR Score	Accuracy
quantity	0.1111	0.0741
date	0.5833	0.4500
location	0.6049	0.4444
person/organization	0.6736	0.5833

Table 2b. Results from selected TREC-8 questions with full qa42 system.

Entity Type	MRR Score	Accuracy
quantity	0.1982	0.1892
date	0.5062	0.4074
location	0.5732	0.3902
person/organization	0.5956	0.5082
Average	0.4869	0.3916

Table 3. Results from all TREC-8 questions with full qa42 system plus entity-type oracle.

Entity Type	MRR Score	Accuracy
quantity	0.0811	0.0811
date	0.3827	0.2963
location	0.3537	0.2927
person/organization	0.4508	0.3934
Average	0.3333	0.2831

Table 4a. Results from all TREC-8 questions with query templates disabled.

Entity Type	MRR Score	Accuracy
quantity	0.1111	0.1111
date	0.5167	0.4000
location	0.5370	0.4444
person/organization	0.5729	0.5000

Table 4b. Results from selected TREC-8 questions with query templates disabled.

Entity Type	MRR Score	Accuracy
quantity	0.0921	0.0789
date	0.4383	0.3704
location	0.3618	0.2683
person/organization	0.4973	0.4262
Average	0.3584	0.2952

Table 5a. Results from all TREC-8 questions with answer-model scoring disabled.

Entity Type	MRR Score	Accuracy
quantity	0.1250	0.1071
date	0.5917	0.5000
location	0.5494	0.4074
person/organization	0.6319	0.5417

Table 5b. Results from selected TREC-8 questions with answer-model scoring disabled.

Entity Type	MRR Score	Accuracy
quantity	0.0857	0.0857
date	0.3889	0.3333
location	0.3618	0.2927
person/organization	0.4917	0.4167
Average	0.3505	0.2970

Table 6a. Results from all TREC-8 questions with both query templates and answer-model scoring disabled.

Entity Type	MRR Score	Accuracy
quantity	0.1154	0.1154
date	0.5250	0.4500
location	0.5494	0.4444
person/organization	0.6146	0.5208

Table 6b. Results from selected TREC-8 questions with both query templates and answer-model scoring disabled.

```
question: Who did write Hamlet?
pronounType: WHO
mainVerb: write
mainVerbRevised: wrote
mainVerbPassive: written
mainVerbForm: PAST
objectNoun: ( (NP [27.133] (NNP [23.177] Hamlet)))
parsePCFG: (ROOT [86.546] (S [86.441] (SBAR [25.134] (WHNP [9.763] (WP
[8.615] Who)) (S [11.803] (VP [11.463] (VBD [7.860] did)))) (VP [42.518]
(VB [13.309] write) (NP [27.133] (NNP [23.177] Hamlet))) (. [8.916] ?)))

getBestPCFGParse:
(ROOT [86.546]
  (S [86.441]
    (SBAR [25.134]
      (WHNP [9.763] (WP [8.615] Who))
      (S [11.803]
        (VP [11.463] (VBD [7.860] did))))
    (VP [42.518] (VB [13.309] write)
      (NP [27.133] (NNP [23.177] Hamlet)))
    (. [8.916] ?)))

queryTemplateList: [[Who did write Hamlet?], [<PRONOUN>, wrote, hamlet],
[hamlet, written by, <PRONOUN>]]

queryList:
Who did write Hamlet?
wrote hamlet
hamlet written by

answerModelList:
[<PRONOUN>, wrote, hamlet] r=1.0
[hamlet, written by, <PRONOUN>] r=1.0
[<PRONOUN>, hamlet] r=0.6666666666666666
[<PRONOUN>, wrote] r=0.6666666666666666
[hamlet, <PRONOUN>] r=0.6666666666666666
[written by, <PRONOUN>] r=0.6666666666666666

correct:  William Shakespeare
qa42 answer 1: mr. william shakespeare   s=12.0
qa42 answer 2: queen elizabeth   s=2.0
qa42 answer 3: christopher marlowe   s=1.0
```

Figure 2. Example.

```
question: Where is Microsoft's corporate headquarters located?
pronounType: WHERE
mainVerb: <BE>
mainVerbRevised: <BE>
mainVerbPassive: <BE>
objectNoun: ( (NP [48.744] (NP [18.375] (NNP [17.089] Microsoft) (POS
[0.146] 's)) (JJ [11.013] corporate) (NN [15.197] headquarters)))
parsePCFG: (ROOT [98.602] (SBARQ [93.184] (WHADVP [9.392] (WRB
[9.327] Where)) (SQ [80.755] (VBZ [2.273] is) (NP [48.744] (NP [18.375]
(NNP [17.089] Microsoft) (POS [0.146] 's)) (JJ [11.013] corporate) (NN
[15.197] headquarters)) (VP [15.440] (VBN [14.105] located)) (. [8.916]
?))))

queryTemplateList: [[Where is Microsoft's corporate headquarters located?],
[<PRONOUN>, corporate headquarters, microsoft's], [corporate headquarters,
microsoft's, <PRONOUN>]]

queryList:
Where is Microsoft's corporate headquarters located?
corporate headquarters microsoft's

answerModelList:
[<PRONOUN>, corporate headquarters, microsoft's] priority=1.0
[corporate headquarters, microsoft's, <PRONOUN>] priority=1.0
[<PRONOUN>, corporate headquarters] priority=0.6666666666666666
[<PRONOUN>, microsoft's] priority=0.6666666666666666
[corporate headquarters, <PRONOUN>] priority=0.6666666666666666
[microsoft's, <PRONOUN>] priority=0.6666666666666666

correct: Redmond, Washington
qa42 answer 1: redmond   r=1.2
qa42 answer 2: washington   r=0.6
qa42 answer 3: phoenix   r=0.4
```

Figure 3. Example.

```
question: When did the original Howdy Doody show go off the air?
pronounType: WHEN
mainVerb: go
mainVerbRevised: went
mainVerbPassive: gone
mainVerbForm: PAST
subjectNoun: ( (NP [84.054] (DT [1.380] the) (JJ [13.813] original) (NNP
[23.163] Howdy) (NNP [23.163] Doody) (NN [14.678] show)))
objectNoun: ( (NP [18.379] (DT [1.380] the) (NN [15.002] air)))
parsePCFG: (ROOT [159.520] (SBARQ [154.102] (WHADVP [4.435] (WRB
[4.370] When)) (SQ [139.291] (VP [137.327] (VBD [7.860] did) (NP
[84.054] (DT [1.380] the) (JJ [13.813] original) (NNP [23.163] Howdy) (NNP
[23.163] Doody) (NN [14.678] show)) (S [41.043] (VP [37.015] (VB [9.585]
go) (PRT [4.415] (RP [4.348] off)) (NP [18.379] (DT [1.380] the) (NN
[15.002] air)))))) (. [8.916] ?)))
queryTemplateList: [[When did the original Howdy Doody show go off the air?],
[the original howdy doody show, went, the air, on, <PRONOUN>], [the original
howdy doody show, went, the air, in, <PRONOUN>], [the original howdy doody
show, went, the air, at, <PRONOUN>], [the air, gone by, the original howdy
doody show, on, <PRONOUN>], [the air, gone by, the original howdy doody
show, in, <PRONOUN>], [the air, gone by, the original howdy doody show, at,
<PRONOUN>]]

QueryList:
When did the original Howdy Doody show go off the air?
the original howdy doody show went the air on
the original howdy doody show went the air in
the original howdy doody show went the air at
the air gone by the original howdy doody show on
the air gone by the original howdy doody show in
the air gone by the original howdy doody show at

answerModelList:
[the air, gone by, the original howdy doody show, at, <PRONOUN>] r=1.0
[the air, gone by, the original howdy doody show, in, <PRONOUN>] r=1.0
[the air, gone by, the original howdy doody show, on, <PRONOUN>] r=1.0
[the original howdy doody show, went, the air, at, <PRONOUN>] r=1.0
[the original howdy doody show, went, the air, in, <PRONOUN>] r=1.0
[the original howdy doody show, went, the air, on, <PRONOUN>] r=1.0
[gone by, the original howdy doody show, at, <PRONOUN>] r=0.8
[gone by, the original howdy doody show, in, <PRONOUN>] r=0.8
[gone by, the original howdy doody show, on, <PRONOUN>] r=0.8
[the air, gone by, at, <PRONOUN>] r=0.8
[the air, gone by, in, <PRONOUN>] r=0.8
[the air, gone by, on, <PRONOUN>] r=0.8
[the air, gone by, the original howdy doody show, <PRONOUN>] r=0.8
[the air, the original howdy doody show, at, <PRONOUN>] r=0.8
[the air, the original howdy doody show, in, <PRONOUN>] r=0.8
[the air, the original howdy doody show, on, <PRONOUN>] r=0.8
[the original howdy doody show, the air, at, <PRONOUN>] r=0.8
[the original howdy doody show, the air, in, <PRONOUN>] r=0.8
[the original howdy doody show, the air, on, <PRONOUN>] r=0.8
[the original howdy doody show, went, at, <PRONOUN>] r=0.8
[the original howdy doody show, went, in, <PRONOUN>] r=0.8
[the original howdy doody show, went, on, <PRONOUN>] r=0.8
[the original howdy doody show, went, the air, <PRONOUN>] r=0.8
[went, the air, at, <PRONOUN>] r=0.8
[went, the air, in, <PRONOUN>] r=0.8
[went, the air, on, <PRONOUN>] r=0.8
[gone by, at, <PRONOUN>] r=0.6
[gone by, in, <PRONOUN>] r=0.6
[gone by, on, <PRONOUN>] r=0.6
[gone by, the original howdy doody show, <PRONOUN>] r=0.6
[the air, at, <PRONOUN>] r=0.6
[the air, gone by, <PRONOUN>] r=0.6
[the air, in, <PRONOUN>] r=0.6
[the air, on, <PRONOUN>] r=0.6
[the air, the original howdy doody show, <PRONOUN>] r=0.6
[the original howdy doody show, at, <PRONOUN>] r=0.6
[the original howdy doody show, in, <PRONOUN>] r=0.6
[the original howdy doody show, on, <PRONOUN>] r=0.6
[the original howdy doody show, the air, <PRONOUN>] r=0.6
[the original howdy doody show, went, <PRONOUN>] r=0.6
[went, at, <PRONOUN>] r=0.6
[went, in, <PRONOUN>] r=0.6
[went, on, <PRONOUN>] r=0.6
[went, the air, <PRONOUN>] r=0.6
[at, <PRONOUN>] r=0.4
[gone by, <PRONOUN>] r=0.4
[in, <PRONOUN>] r=0.4
[on, <PRONOUN>] r=0.4
[the air, <PRONOUN>] r=0.4
[the original howdy doody show, <PRONOUN>] r=0.4
[went, <PRONOUN>] r=0.4

correct: 1960
qa42 answer 1: 1960   s=2.6
qa42 answer 2: 1954   s=0.8
qa42 answer 3: 1941   s=0.6
```

Figure 4. Example.

Query Template Format	Verb	Subject	Object	Type	Example Summary
subject, verb, pronoun	*req*	*req*			Charles Robb represents Virginia.
pronoun, passiveverb, subject	*req*	*req*			Virginia is represented by Charlse Robb.
subject, verb, pronounsubtype, pronoun	*req*	*req*			Charles Robb represents the state of Virginia.
pronounsubtype, pronoun, passiveverb, subject	*req*	*req*			The state of Virginia is represented by Charlse Robb.
subject, verb, pronoun, pronounsubtype	*req*	*req*			Charles Robb represents Virginia state.
pronoun, pronounsubtype, passiveverb, subject	*req*	*req*			Virginia state is represented by Charlse Robb.
pronounsubtype, verb, object	*req*		*req*		
object, passiveverb+by, *pronoun*	*req*		*req*	*who*	John Lennon was killed by Mark Chapman.
object, passiveverb+at, *pronoun*	*req*		*req*	*where*	John Lennon was killed at The Dakota.
object, passiveverb+in, *pronoun*	*req*		*req*	*where*	John Lennon was killed in The Dakota.
object, passiveverb+on, *pronoun*	*req*		*req*	*when*	John Lennon was killed on December 8. 1980.
object, passiveverb+at, *pronoun*	*req*		*req*	*when*	John Lennon was killed at 10.50pm on December 8. 1980.
object, passiveverb+in, *pronoun*	*req*		*req*	*when*	John Lennon was killed in 1980 on December 8.
subject, verb, object, in, *pronoun*	*req*	*req*	*req*	*where*	Mark Chapman killed John Lennon in The Dakota.
subject, verb, object, at, *pronoun*	*req*	*req*	*req*	*where*	Mark Chapman killed John Lennon at The Dakota.
object, passiveverb+by, *subject,* in, *pronoun*	*req*	*req*	*req*	*where*	John Lennon was killed by Mark Chapman in The Dakota.
object, passiveverb+by, *subject,* at, *pronoun*	*req*	*req*	*req*	*where*	John Lennon was killed by Mark Chapman at The Dakota.
subject, verb, object, on, *pronoun*	*req*	*req*	*req*	*when*	French revolutionaries stormed the Bastille on July 14, 1789.
subject, verb, object, in, *pronoun*	*req*	*req*	*req*	*when*	French revolutionaries stormed the Bastille in 1789.
subject, verb, object, at, *pronoun*	*req*	*req*	*req*	*when*	French revolutionaries stormed the Bastille at sunrise on July 14, 1789.
object, passiveverb+by, *subject,* on, *pronoun*	*req*	*req*	*req*	*when*	The Bastille was stormed by French revolutionaries on July 14, 1789.
object, passiveverb+by, *subject,* in, *pronoun*	*req*	*req*	*req*	*when*	The Bastille was stormed by French revolutionaries in 1789.
object, passiveverb+by, *subject,* at, *pronoun*	*req*	*req*	*req*	*when*	The Bastille was stormed by French revolutionaries at sunrise on July 14, 1789.
subject, verb, object, pronoun	*req*	*req*	*req*		Mark Chapman killed John Lennon with a gun.
object, passiveverb+by, *subject, pronoun*	*req*	*req*	*req*		John Lennon was killed by Mark Chapman with a gun.
subject, verb, object, pronounsubtype, pronoun	*req*	*req*	*req*		Mark Chapman killed John Lennon in the year 1980.
pronounsubtype, pronoun, subject, verb, object	*req*	*req*	*req*		In the year 1980, Mark Chapman killed John Lennon.
object, passiveverb+by, *subject, pronounsubtype, pronoun*	*req*	*req*	*req*		John Lennon was killed by Mark Chapman in the year 1980.
pronounsubtype, pronoun, object, passiveverb+by, *subject*	*req*	*req*	*req*		In the year 1980, John Lennon was killed by Mark Chapman.
pronounsubtype, subject, verb, object, pronoun	*req*	*req*	*req*		The year Mark Chapman killed John Lennon was 1980.
pronoun, pronounsubtype, subject, verb, object	*req*	*req*	*req*		1980 was the year that Mark Chapman killed John Lennon.
pronounsubtype, object, passiveverb+by, *subject, pronoun*	*req*	*req*	*req*		The year that John Lennon was killed by Mark Chapman 1980.
pronoun, pronounsubtype, object, passiveverb+by, *subject*	*req*	*req*	*req*		1980 was the year that John Lennon was killed by Mark Chapman.
pronoun, verb	*req*				Bob procrastinates.
pronounsubtype, verb, pronoun	*req*				The metal that burns is magnesium.
pronoun, pronounsubtype, verb	*req*				Magnesium is the metal that burns.
pronounsubtype, pronoun, verb	*req*				The metal magnesium burns.

Table 7a. Different formats for query templates. In the example, the pronoun is underlined. This table is continued on the following page.

Query Template Format	Verb	Subject	Object	Type	Example Summary
verb, pronounsubtype, pronoun	*req*				The spiffiest substance that burns is the metal magnesium.
subject, object, pronoun, pronounsubtype		*opt*	*opt*	*how*	
pronounsubtype, pronoun, subject, object		*opt*	*opt*	*which*	
pronoun, pronounsubtype, subject, object		*opt*	*opt*		
pronounsubtype, subject, object, pronoun		*opt*	*opt*		
pronoun, subject, object					
subject, object, pronoun					
subject, pronoun, object		*req*	*req*		

Table 7b. Different formats for query templates. In the example, the pronoun is underlined. This table was continued from the previous page.

Java Class	Summary
`qa42.query.Model`	Model for the expected answer expressed as a `List` of `String` objects. Most items in the list represent phrases that are to be matched in the search-engine summary. Each `Model` object contains one reference to the symbol `pronoun` to represent the position in which the answer is expected to be found.
`qa42.query.ModelList`	`List` of `Model` objects for a particular `Question`. Sorted by model priority when constructed by `TemplateList.queryModelList`.
`qa42.query.QuerySet`	`Set` of `String` objects that are sent as queries to the search engine. Each `String` element represents one query.
`qa42.query.Question`	Question to be answered. An object of this class encapsulates several attributes including the question text, an arbitrary identifier, a list of gold-standard answers, and analysis results including the parse results and query templates.
`qa42.query.QuestionList`	`List` of `Question` objects. This class includes methods for reading questions from specific plain text and XML file formats used by [5].
`qa42.query.SimpleTemplate`	Special form of a `Template` object produces no `Model` objects. This class is used by `Question` methods to insert a copy of the original question as a query template since it contains no `pronoun` element.
`qa42.query.Template`	Query template represented as a `List` of `String` objects. Each object is sent to the search engine as a query; each element represents a phrase used to generate several `Model` objects. The special symbol pronoun is included in all Model objects but excluded from the search engine query.
`qa42.query.TemplateList`	`List` of `Template` objects. This class contains logic to generate a `QuerySet` object and a sorted `ModelList` object.
`qa42.word.AreaList`	`Set` of words that indicate units of measure for area.
`qa42.word.BeList`	`Set` of words that are forms of the verb "to be".
`qa42.word.ByteList`	`Set` of words that indicate units of measure for data storage.
`qa42.word.DistanceList`	`Set` of words that indicate units of measure for distance.
`qa42.word.HelperVerbList`	`Set` of words that are forms of the verb "to do" and "to have".
`qa42.word.InterrogativePronoun`	`Enum` indicating the question pronoun type, i.e. who/whom, where, when, why, how, which, what, and other.
`qa42.word.LocationList`	`Set` of words that indicate location. Words in this list, when combined with interrogative pronouns "which" or "what", are taken to be idiomatically equivalent to "where".
`qa42.word.MoneyList`	`Set` of words that indicate units of measure for money including the names of most currencies through modern history.
`qa42.word.PostFixList`	`Set` of words that can be used as a postfix.
`qa42.word.PrefixList`	`Set` of words that can be used as a prefix, such as currency symbols.
`qa42.word.TimeList`	`Set` of words that indicate units of measure for time.
`qa42.word.VerbForm`	`Enum` indicating the voice and tense of the main verb, i.e. past, present singular, present plural.
`qa42.word.VerbFormConverter`	Logic for converting present tense to past perfect and past participle. Member `Map` objects handle irregular forms.
`DateMatcher`	NER for date entities.
`HTMLPreprocessor`	Logic for cleaning up search-engine summaries.

Table 8. Summary of Java classes.