22 Question Answering

BONNIE WEBBER AND NICK WEBB

1 What is Question Answering?

Questions are asked and answered every day. Question answering (QA) technology aims to deliver the same facility online. It goes further than the more familiar search based on *keywords* (as in Google, Yahoo, and other search engines), in attempting to recognize what a question expresses and to respond with an actual answer. This simplifies things for users in two ways. First, questions do not often translate into a simple list of keywords. For example, the question

(1) Which countries did the pope visit in the 1960s?

does not simply translate to the keywords 'countries,' 'pope,' 'visit,' '1960s' because a search on those keywords will only find documents (web pages) that contain the *words* 'countries' (or 'country,' if the search engine recognizes plurals), 'pope,' and '1960s,' and not words or phrases that denote particular countries (such as 'United Kingdom,' or the 'United States'), or the pope ('head of the Catholic church,' for example), or a date within the 10-year time span between '1960' and '1970.' A much more complex set of keywords is needed in order to get anywhere close to the intended result, and experience shows that people will not learn how to formulate and use such sets.

Second, QA takes responsibility for providing answers, rather than a searchable list of links to potentially relevant documents (web pages), highlighted by *snippets* of text that show how the query matched the documents. While this is not much of a burden when the answer appears in a snippet and further document access is unnecessary, QA technology aims to move this from being an accidental property of search to its focus.

In keyword search and in much work to date on QA technology, the information seeking process has been seen as a one-shot affair: the user asks a question, and the system provides a satisfactory response. However, early work on QA (Section 1.1) did not make this assumption, and newly targeted applications are hindered by it: while a user may try to formulate a question whose answer is the information they want, they will not know whether they have succeeded until something has been returned for examination. If what is returned is unsatisfactory or, while not the answer, is still of interest, a user needs to be able to ask further questions that are understood in the context of the previous ones. For these target applications, QA must be part of a collaborative search process (Section 3.3).

In the rest of this section, we give some historical background on QA systems (Section 1.1), on dialogue systems in which QA has played a significant role (Section 1.2), and on a particular QA task that has been a major driver of the field over the past 8 years (Section 1.3). Section 2 describes the current state of the art in QA systems, organized around the de facto architecture of such systems. Section 3 discusses some current directions in which QA is moving, including the development of *interactive QA*. We close with some pointers to further reading.

1.1 Early question answering systems

Early QA systems were developed to enable users to ask interesting questions about well-structured data sets such as baseball statistics, personnel data, or chemical analyses of lunar rock and soil samples (Simmons 1965 provides an early survey). These early QA systems essentially attached a front end and a back end to a database system. The front end performed parsing and interpretation, mapping questions phrased in everyday terms onto a form that specified a *computation* to be carried out over the database – for example, the question

(2) What is the average concentration of aluminum in high-alkali rocks?

would be mapped to a computation that identifies the high-alkali rocks in the database, finds the aluminum concentration in each, and then computes an average over those values.

Given the potential complexity of such queries, differences between the system's and the user's underlying models of the data, as well as users' frequent lack of awareness of what information is actually in the database, early QA development focused on such issues as:

- mapping user questions to computable database queries;
- handling questions that could not be parsed or that could not be interpreted as a valid query;
- resolving syntactic and referential ambiguities detected in questions;
- handling differences in how user and system conceptualized the domain (e.g., user queries about the age of lunar rocks versus system data on potassium/rubidium and uranium isotope ratios, as well as differences between what user and system believed to be true in the domain (Kaplan 1982; Mays et al., 1982; Pollack 1986; Webber 1986);
- identifying, in the case of distributed databases, what information needed to be imported from where, in order to answer the user's question (Hendrix et al., 1978).

User–system interactions designed to resolve ambiguities or reconcile mismatches between user and system beliefs about the domain showed that satisfying a user's information needs required the user to do more than just ask questions and the system to do more than just answer them. But systems were still viewed as question answerers, carrying out other types of interactions on an 'as needed' basis.

This first foray into database QA was essentially abandoned in the late 1980s for two reasons – one technical, and one social. Technically, considerable effort was needed to guarantee an effective and reliable mapping between user questions and database queries. Not only did the correct mapping depend on the structure of the particular database, but many disparately phrased user questions needed to be mapped onto the same database query. Even worse, questions that differed only minimally needed to be mapped onto very different database queries. The only solution to these problems available at the time was more and more mapping rules that had to be written by hand by system experts. As a solution, this was neither scalable nor portable. The social problem involved the lack of a significant audience for the technology: ordinary people lacked access to large data sets, and managers whose companies maintained large data sets lacked sufficient interest in accessing the data themselves. Companies such as Symantic which developed state-of-the-art software for database QA (Hendrix 1986) ended up abandoning it.

With the advent of the web, this social problem disappeared, and machine learning techniques that have proved so useful in other areas of language technology are beginning to be applied to the problem of learning (complex) mappings between user questions and database queries (Mooney 2007; Zettlemoyer & Collins 2007). While it is still early days, this does mean that the growing number of databases containing rich and useful information may again be primed for access through natural language questions.

1.2 Question answering in dialogue systems

QA was also a feature of early systems whose main purpose in interacting with users in natural language was something other than answering their questions. In one of the earliest of such *dialogue systems*, SHRDLU (Winograd 1973), users could converse with an animated robot (with a visible arm) that could both act (in response to user requests) and reflect on its actions (in response to user questions). More generally, users could question SHRDLU about the state of its world (e.g., where objects were, what objects were where) or about its previous actions or its plans (e.g., why or when it performed some action), which SHRDLU could answer in terms of the history it maintained of its goals and actions.

Interactions with SHRDLU, and with another animated robot system called the Basic Agent (Vere & Bickmore 1990), were through typed text. Later systems supported limited speech interaction (Allen et al., 1996; Lemon et al., 2001; Eliasson 2007). Because these robots did not have any goals of their own, apart from those adopted in response to user requests/commands, and because no mechanism was provided for user–robot collaboration, the dialogic capability of these systems did not extend to asking questions of the user or to making requests themselves.

Other more recent dialogue systems, which take on other roles with respect to the user, also include QA capabilities in their repertoire. In *intelligent tutoring systems*, the system has goals in its role of tutor – e.g., assessing the student’s knowledge, correcting the student’s errors, and imparting information that the student is missing. Dialogues can thus involve the system introducing and describing a topic for tutoring, after which it might ask the student a question about some aspect of the problem, explain why the student’s response is correct or incorrect, and/or remind the student of something already said previously during the interaction. Here, it is the student who answers questions or says he/she does not know, but it is still the system’s job to determine if an answer is correct. Again, the earliest tutoring systems, like SOPHIE (Brown & Burton 1975), interacted through typed text, while later systems such as ITSPOKE (Litman & Forbes-Riley 2006) allow for spoken interaction.

The most industrially relevant role played by dialogue systems has been in information provision and user assistance, such as in helping users to plan travel (Goddeau et al., 1994), book flights (Seneff 2002), or find an appropriate restaurant (Walker et al., 2004), or in routing user telephone calls (Chu-Carroll & Carpenter 1999) or handling directory inquiries (De Roeck et al., 2000). All such tasks involve the system getting sufficient information from the user to fully or partially instantiate some form (often called a *frame*) which the system can evaluate (just like a database query) and present the results to the user as a basis for further interaction. In such cases, the user’s information needs may be anywhere from completely formed to vague. They may or may not be able to be satisfied given the underlying data, and queries may need to be reformulated on the basis of additional knowlege and relaxed constraints. Dialogues can thus involve the system asking the user questions related to values of frame elements; the user specifying such values (either precisely or vaguely); the system listing and/or describing the results (when too numerous to list); the user choosing some item from among the results or modifying or replacing some already specified values; and the system requesting confirmation of its understanding.

A key emerging element of dialogue approaches is their inherent generality – the potential for subdialogue structures independent of task or application (such as for error correction or clarification) that will, in the future, allow them to be seamlessly integrated with QA systems (Section 3.3). For more detailed discussion of the issues and technology underlying dialogue systems, see Chapter 16, COMPUTATIONAL MODELS OF DIALOGUE.

1.3 Question answering in TREC

Returning to straight QA systems, the advent of the web has made increasing amounts of information accessible to people. To find the bits of interest, people are using search methods pioneered in text retrieval (a field revitalized by the web), but scaled up to nearly instant response to vast numbers of users. In this context, QA too has found a second life, with the idea that users would find not just relevant documents (web content), but the particular information they are seeking.

This vision of QA differs significantly from QA over databases, where (as noted in Section 1.1) questions map onto *computations* carried out over a database of known structure. Instead, in what has been called *open domain question answering*, the answer to a question must be *found* and *extracted* rather than *computed*, as a natural extension to text retrieval. Advances in open domain QA have been accelerated by its adoption within the Text Retrieval Conference (TREC).

Initially, TREC QA focused on 'factoid' questions – questions of *who*, *what*, *where*, and, to some extent, *when*, that can be answered by a short word or phrase. From 1999 to 2007, TREC QA advanced on several fronts, to address increasingly large document collections, increasingly complex questions, and increasingly complex evaluation strategies. While the basic approach to factoid QA is now well understood, challenges remain. This basic approach and the current state of the art are described in Section 2, and some challenges that are now beginning to be addressed, in Section 3.

2 Current State of the Art in Open Domain QA

A basic QA system involves a cascade of processes that takes a user's question as input and responds in the end with an answer or rank-ordered list of top answer candidates, along with an indication of the source of the information (see Figure 22.1). This embodies the de facto paradigm for QA characterized by Paşca (2007) as:

- retrieve potentially relevant documents;
- extract potential answers (called here *answer candidates*);
- return top answer(s).

We discuss *question typing* in Section 2.1, *query construction* and *text retrieval* in Section 2.2, *text processing for answer candidates* (including weighting, ranking, and filtering candidates) in Section 2.3, and *answer rendering* in Section 3.3. Performance evaluation is briefly discussed in Sections 2.4 and 3.4, and at greater length in Chapter 11, EVALUATION OF NLP SYSTEMS.

2.1 *Question typing*

Questions generally undergo two initial processes to identify what type of information is being sought (*question typing*) and in what piece of text it is likely to be found (*query construction*). Although these processes can be carried out in parallel, *query construction* is so intimately tied up with *text retrieval*, that we will discuss them together in Section 2.2. *Question typing* aims to associate a label (QType) with

“9781405155816_4_022” — 2010/3/5 — 20:24 — page 635 — #6

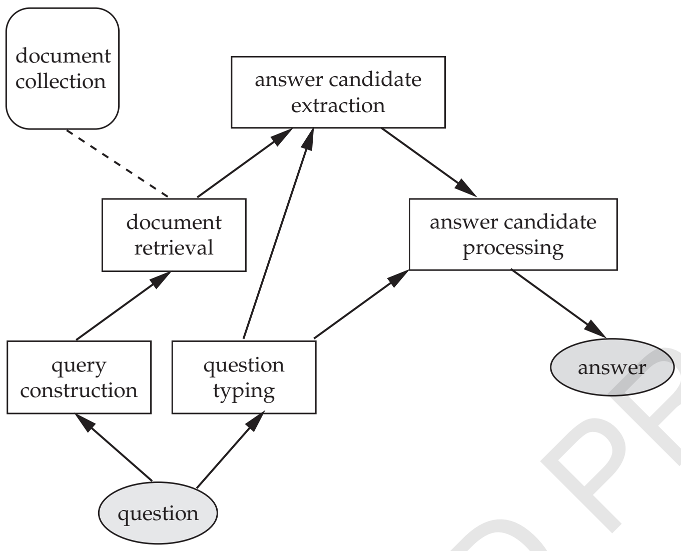


Figure 22.1 Basic QA system architecture.

a question, indicating the kind of information being sought – e.g., the meaning of an abbreviation (ABBREV) or of a word or phrase (DEFINITION), the name of the person who has or had some particular property or set of properties (PERSON), etc. For more on the location of specific entities in text, see Section 3 on name extraction in Chapter 18, INFORMATION EXTRACTION. These labels provide testable semantic constraints on possible answer candidates. Labels assigned by *question typing* have also been used to support *text retrieval* through *predictive annotation* (Section 2.2), as well as to support the processing involved with identifying and ranking answer candidates. For example, a system that has done text retrieval on short passages may filter out ones that lack anything of the appropriate type as an answer candidate (Section 2.3).

While manually constructed rules have been used for *question typing* – for example,

- If the question starts with *Who* or *Whom*, QType is PERSON.
- If the question starts with *Where*, QType is LOCATION.

state-of-the art systems (Li & Roth 2006; Moschitti et al., 2007) use probabilistic classifiers, P(QType | Q), where the question Q is represented as a set of features. The features used have become more sophisticated over time, now extending to syntactic and semantic, as well as lexical features.

Using syntactic features alone, Li and Roth (2006) achieve state-of-the-art performance, using the SNoW classifier (Carlson et al., 1999) over the question types in the widely available TREC 10 and 11 question sets (Voorhees 2002; available from http://l2r.cs.uiuc.edu/∼cogcomp/). This corpus of 1,000 manually annotated questions has both coarse-grain (i.e., ABBREVIATION, ENTITY, DESCRIPTION, HUMAN, LOCATION, NUMERIC) question types, and a further 50 refinements of those categories (so for example the coarse-grain category HUMAN can be represented by four fine-grain categories: GROUP, INDIVIDUAL, TITLE, and DESCRIPTION). For the coarse-grain question types, Li and Roth (2006) achieve 92.5 percent accuracy, and 85 percent on the fine grain. Using semantic information, including named entities and some manually extracted word class information, did not significantly improve classification accuracy over the coarse-grain question types, but boosted accuracy over the fine-grain categories to 89.3 percent.

Such accuracy figures disguise the fact that a question can often have more than one type of answer. For example, is *What is an SME?* an ABBREV question for which an appropriate strategy would involve patterns commonly used to relate abbreviations with their full-text forms, or is it a DESCRIPTION question, for which an appropriate strategy would involve patterns commonly used in defining terms? A *who* question may be answered by a PERSON (*Who won the Masters tournament in 1985?*), or an ORGANIZATION (*Who won the Nobel Peace Prize in 1999?*), or a COUNTRY (*Who won the World Cup in 2006?*). When a question is asked, a system may know neither which kind of information the user is seeking nor what information the corpus contains. Allowing questions to have more than one possible type permits the system to first see what kind of answers the corpus supports and, if more than one, provide the user with answers that cover the different alternatives. It is only because TREC has required systems to produce a single minimal answer to a question that work on QA has by and large ignored the possibilities of such helpful responses.

2.2 *Query construction and text retrieval*

In open domain QA, questions are always answered with respect to a corpus of texts. This can be as vast and diverse as the web or more specialized, such as the collection of biomedical abstracts in MedLine (www.ncbi.nlm.nih.gov/PubMed) or the collection of news articles in the AQUAINT corpus (Voorhees & Tice 2000). Because both the types of queries that one can construct and their success in retrieving text with good answer candidates reflect characteristics of the corpus and its access methods, it makes sense to discuss *query construction* and *text retrieval* together. More specifically, the kind of query a system constructs depends on which of two forms of retrieval is used to find text that might contain an answer to a user's question: *relevance-based retrieval* and *pattern-based retrieval*. These are discussed in the next subsections.

2.2.1 Relevance-based retrieval In *relevance-based retrieval*, queries are interpreted as requests for texts *relevant to* a topic. Relevance may be assessed in terms of matching a Boolean combination of terms or proximity to a weighted vector of terms or language model, just as in standard text retrieval (Manning et al., 2008). The problem, as already noted, is that QA demands answers, rather than the texts that contain them, and such answers are generally expressed very *locally* in a text. (With complex questions, the evidence supporting an answer may actually be distributed across one or even several texts. This is discussed in Section 3.2.) Although standard word-based indexing of texts supports very fast and efficient retrieval, such indexing characterizes texts in terms of their gross lexical properties, not by local information. For example, a news article on the 1987 *Herald of Free Enterprise* ferry disaster may mention in passing 'It was the worst peacetime disaster involving a British ship since the *Titanic* sank in 1912.' While this sentence contains an answer to the question

(3) When did the *Titanic* sink?

this local information might simply be noise with respect to more prominent lexical properties of the article, such that it might not be retrieved on a relevance-based search for '*Titanic*' and 'sink,' or it might not be ranked high enough for the answer candidate extraction process to ever get to it (Section 2.3). So, with respect to a given question, *text retrieval* using relevance-based methods is not guaranteed to return texts with a high likelihood of having an answer somewhere within them. And if text retrieval fails to return texts containing answers, any subsequent processing might as well not happen. Attempts to improve this situation are usually found under the rubric *information retrieval for question answering (IR4QA)*.

A simple and widely adopted solution to the locality problem in relevance-based retrieval is simply to break texts into a set of separate passages, each of which is separately indexed for text retrieval (*passage retrieval*), assuming that there is a better correlation between standard metrics of *relevance* and *answers* in short texts.

Other text retrieval techniques used instead of, or in addition to, segmentation into smaller passages involve extensions to what is indexed. In *predictive annotation* (Prager et al., 2000), texts are indexed not just by the position of each (non-stop) word in the text, but also by the position of each of 20 types of named entities (called here *QA-tokens*), each of which could answer a question of a given type. For example, predictive annotation of the sentence

(4) Sri Lanka boasts the highest per capita income in South Asia.

would index the QA-token COUNTRY$ as occuring over the first two words of the sentence, and the QA-token PLACE$ as occuring over the last two. (Techniques used here are similar to those used in information extraction, as discussed in Chapter 18, INFORMATION EXTRACTION.)

QA-tokens are used as well in *query construction*. When user questions are mapped to queries, not just keywords from the question are included in the query, but also an appropriate QA-token (or disjunction thereof) for the question type. For example, the *where* question

(5) Where is the capital of Sri Lanka?

would be mapped to a query comprising the keywords 'capital,' 'Sri,' and 'Lanka,' along with a disjunct of the QA-tokens (PLACE$, COUNTRY$, STATE$, NAME$), all of which can potentially answer a *where* question. Such a query would cause to be retrieved any passage containing the sentence in (4), although, as Prager notes, it is important to discard the match between COUNTRY$ and the annotation of 'Sri Lanka' as COUNTRY$, in order to answer the question correctly. (Even where QA-tokens, or answer types, are not themselves indexed, they may still be used to filter out retrieved texts or passages that lack any instance of the QA-token/answer type.)

While *predictive annotation* reduces 'false positives' in text retrieval by demanding passages that contain candidate answers of particular types, they can also be reduced by indexing not just words and their positions but how the words are used. This is illustrated in the *Joost* system (Bouma et al., 2005), which indexes each word along various linguistic dimensions (e.g., part of speech, dependency relations, etc.), as well as indexing the type and span of each named entity. Using a genetic algorithm to optimize various weightings, Bouma and his colleagues found a 10 percent improvement in passage retrieval on a test set of 250 questions, measured by MRR (*mean reciprocal ranking*) over passages containing a correct answer. Moreover, Morton (2005) found that when third-person pronouns were resolved and indexed by the full NPs with which they corefer, the frequency of both 'false positives' and 'false negatives' could be reduced by narrowing the distance between potential answer candidates and terms from the question (all treated as full NPs through coreference resolution).[1]

In all this work, the basic idea is that the more frequently a system can rely on top-ranked passages to contain a correct answer, the fewer passages need to be examined in the next stage of processing. It should be remembered, however, that any ranking in text retrieval is a ranking on texts, not on the answers they may contain. A separate process of answer candidate ranking is carried out at a later stage (Section 2.3).

A survey of passage retrieval techniques for QA can be found in Tellex et al. (2003).

2.2.2 Pattern-based retrieval The other type of text retrieval used in QA is *pattern-based retrieval*. This relies on the *quoted string* search capabilities of search engines, reflecting the assumption that, although natural language usually provides multiple ways to express the same information, it is expressed in the same or a similar way to the question somewhere in the corpus. Pattern-based retrieval also differs from relevance-based retrieval in taking the result of retrieval to be the *snippet* returned as evidence for the match (cf. Section 1) rather than a pointer to the text that contains it. Thus pattern-based retrieval does not derive any benefit from breaking texts into smaller passages, as does relevance-based retrieval, because it targets snippets rather than their source.

Pattern-based retrieval systems differ from one another in the kinds of string patterns they use in performing the match. The well-known AskMSR system

(Brill et al., 2002) used different sets of string patterns for different types of questions. For *where is* questions, for example, retrieval patterns were generated with 'is' in every possible position – e.g.

(6) Where is the Vale of Tears?
is the Vale of Tears
the is Vale of Tears
the Vale is of Tears
the Vale of is Tears
the Vale of Tears is

Patterns usually reflect a direct relationship between questions and their answers. For example, since wh-questions in English (where, when, who, what, which, why) usually involve *fronting* the question phrase (and, in some cases, an auxiliary as well), as in:

(7) When was the telephone invented?
(8) In which American state is Iron Mountain located?
(9) What is the largest whale?

their answers are likely to be found non-fronted, directly to the right of the strings:

- *the telephone was invented in* <answer>
- *Iron Mountain is located in* <answer>
- *the largest whale is* <answer>

Other syntactic relationships predict other string patterns, such as:

- *invented the telephone in* <answer>
- *the telephone, invented in* <answer>
- *in* <answer> *the telephone was invented*
- *Iron Mountain, located in* <answer>
- <answer> *is the largest whale*
- <answer> *is the largest of the whales*

As Lin (2007) notes, a quoted string search does not guarantee that the snippet returned from a search engine as evidence will actually contain a filler for the question phrase (i.e., an answer). There are several reasons for this. First, the answer material may be outside the boundaries of the snippet. Secondly, since search engines (as of this writing) ignore punctuation and case, a quoted string can also match over successive sentences and/or clauses (i.e., across final punctuation), producing false positive matches that have to be filtered out later on, as in

(10) He was six when *the telephone was invented. In 1940*, he ...

which a search engine would find as a match for the string *the telephone was invented in*.

More recently, researchers have shown how lexical resources such as WordNet (Miller et al., 1990) and FrameNet (Baker et al., 1998) can be used to construct additional string patterns that can be used in pattern-based retrieval (Kaisser & Webber 2007). Again, one must be aware that any ranking here is the search engine's ranking on the documents, not on any answer candidates contained in the snippets. The ranking of answer candidates themselves is discussed in Section 2.3, after answer candidate extraction.

2.3 *Processing answer candidates*

Once a set of passages or snippets has been retrieved, a system must determine what, if anything, in each separate passage or snippet might serve as an answer to the question (*answer candidate extraction*) and how good an answer it might be (*answer candidate evaluation*). The latter requires assessing the candidate and its textual context, possibly comparing it as well with the other, separately extracted candidates. Candidates are then ranked based on this assessment, with the top-scoring candidate (or top N scoring candidates) presented to the user. Techniques for these different aspects of answer candidate processing are described below.

Two kinds of patterns are used for extracting answer candidates from passages or snippets. They can be extracted directly, using *string patterns* that identify the contents of a particular bounded span as an answer candidate, or they can be extracted using *structured patterns*, from the output of parsing, semantic role labeling, and/or interpreting the passages or snippets.

String patterns are the simplest. They can be derived directly from the user's question (exactly as in pattern-based retrieval, Section 2.2.2), authored manually, or computed by bootstrapping from questions with known answers. To illustrate the latter, known [entity, location] pairs such as [Taj Mahal, Agra], [Grant's Tomb, New York City], etc., can be used to retrieve texts that suggest the following patterns for identifying the answer to location questions:

<NAME> [is | are] located in <ANSWER>.
<NAME> in <ANSWER>,
<NAME> [lies | lie] on <ANSWER>.

In each case, the answer candidate is bounded on its left and right by an identifiable word or symbol. Such patterns can also be characterized by their reliability – how often they pick up a candidate of the right type, as opposed to a random string. Because patterns are so simple, they are rarely completely reliable, as in example 11, where 'the background' is identified as an answer candidate. Because parentheticals, adverbial modifiers, and adjuncts can occur so freely within a sentence, no set of patterns can reliably capture all desired answer candidates, as in example 12, where the adverbials 'all the way up' and 'all the way down' block the 'located in' pattern from matching.

(11) 'Where are the Rocky Mountains?'
<NAME> in <ANSWER>

'Denver's new airport, topped with white fiberglass cones in imitation of the Rocky Mountains in **the background**, continues to lie empty.'

(12) 'Where are the Rocky Mountains?'
<NAME> [is | are] located in <ANSWER>.
'The Rocky Mountains are located all the way up to Alaska and all the way down to Mexico.'

To reduce false positives (as in example 11), answer candidates found by string patterns are commonly filtered by tests derived from the question type (Section 2.1). For example, for a *where* question, <ANSWER> must be classifiable as a LOCATION. For a *who* question, it must be classifiable as a PERSON, or in some cases, a PERSON or a COMPANY. Named entity recognition, often in conjunction with WordNet and/or Wikipedia categories, has been used in this filtering.

Answer candidate extraction rarely produces a single candidate answer, so one or more best candidates need to be selected and displayed to the questioner. To produce a ranking of such candidates so as to be able to select the best, their quality needs to be assessed. The simplest and most common method of doing so involves computing their frequency of occurrence within the set of answer candidates (Brill et al., 2002). To reflect true frequency, this requires recognizing answer candidates that should be treated as equivalent, such as 'Clinton,' 'Bill Clinton,' 'William Jefferson Clinton,' and 'ex-President Clinton,' rather than as distinct candidates. Online resources such as Wikipedia are useful in this regard.

A more complex method can be used when correct answers to questions occur only infrequently in the corpus. This method involves assessing the probability that an answer candidate is the correct answer to the question (Xu et al., 2002; Ko et al., 2007). Since lack of sufficient data prevents such probabilities from being computed precisely, they are approximated using such features as the ways in which the context of answer A_i matches question Q, whether A_i is correctly typed for the argument role that the question word plays in Q, whether A_i is supported as an answer by online resources such as gazetteers, WordNet, Wikipedia and/or the snippets returned from web search. Probabilities based on these approximations have been shown to indeed increase the ranking of good answers, thereby improving system performance.

Result of answer candidate extraction and evaluation is a rank-ordered list of answer candidates, of which the top (or top N) is/are presented as the answer(s), with each supported by either a document ID or a specific piece of text meant to serve as evidence. In the case of answer candidates that occur multiple times, each with different support, there has as yet been no attempt to assess the quality of that evidence – whether one piece of text might provide either stronger or clearer support for an answer.

2.4 *Evaluating performance in QA*

Factoid QA in TREC treats every question as having a single correct answer, although it may be described in different ways (e.g., a distance question with an answer given in miles or in kilometers, with a precise value or an approximate one, etc.). In early QA tracks, groups were allowed to return a list of five possible answers for each question, in rank-order, so that a question could be scored by the *reciprocal rank* of the first correct answer: if it ranks first, it gets a score of 1; if it ranks second, it gets a score of 1/2; etc. If only the fifth answer is correct, it gets a score of 1/5. Otherwise it scores a 0. Over a set of questions, then, a system could be assessed in terms of *mean reciprocal rank* (MRR) – i.e., the average of the reciprocal rank score of the question set. MRR is still often used to measure system performance, as it is more lenient than assessing only a single answer.

Although *recall* and *precision* are also used in assessing system performance, because they do not take account of the ranked position of answers and because only the top-ranked correct answer to a given question actually matters (not how many correct answers have been returned for the question), *coverage* and *answer redundancy* are sometimes used as alternatives to recall and precision for assessing retrieval in QA (Roberts & Gaizauskas 2004).

- *Coverage* is the proportion of the question set for which a correct answer can be found within the top *n* passages retrieved for each question.
- *Answer redundancy* is the average number, per question, of passages within the top *n* retrieved that contain a correct answer.

Coverage is preferable to *recall* because, since factoid questions are taken to have a single answer, it does not make sense to worry about *recall* (i.e., the proportion of documents containing that answer).

These metrics all assume that every question has a correct answer (or different descriptions of the correct answer), so at issue is where judgments of correctness come from. When evaluating QA systems, one must keep the document collection fixed, so that differences in performance do not simply reflect differences in the corpus. Because it is expensive to find all possible answers (or possible ways to phrase every answer) in a large corpus, a method called *pooling* was developed.

As used in large, multi-system QA evaluations, pooling involves collecting the rank-ordered results from all runs of all the participating systems; selecting N runs per system; taking the top X documents from each of those N runs and merging them into a judgment pool for that query, eliminating duplicates, and, finally, manually assessing correctness judgments on only these pooled answers. Importantly, pooling has been found not to be biased towards systems that contribute to the pool – that is, there is no performance benefit to be gained by participating in the pooling system. On the other hand, pooling makes it harder to assess new techniques which may produce answers that may be correct but do not occur in the pool formed from earlier results.

To allow for the development of such techniques, Lin and Katz (2006) attempted to find all possible answers in the AQUAINT corpus for a subset of questions from TREC, while, more recently, Kaisser and Lowe (2008) have used the *Mechanical Turk* to create an even larger set of 8,107 [question, document id, answer, answer source sentence] tuples for the over 1,700 TREC questions from 2002 to 2006.

More discussion of evaluation methods used in QA can be found in Chapter 11, EVALUATION OF NLP SYSTEMS.

3 Current Directions

QA technology is moving in several different directions, but here we will focus on the three that we find most important: (1) extending the relationship between question and corpus; (2) broadening the range of questions that can be answered; and (3) deepening the relationship between user and system. We briefly discuss each in turn, and then follow with a brief discussion of the consequences for evaluation.

3.1 Extending the relation between question and corpus

We have mentioned several times that open domain QA involves *finding* answers in text, while database QA involves *computing* them, usually from an assortment of other, simpler facts. Section 2.2 showed that evidence that enabled an answer to be located could take the form of words from the user's question, possibly augmented by the syntactic relations between them, or the form of lexical and/or syntactic variations on the question that could be predicted from handmade resources such as WordNet and FrameNet or from patterns found through text mining. As Kaisser and Webber (2007) showed, however, this is not enough: while a corpus as large as the web might yield an answer phrased similarly to a given question, especially if the answer is widely discussed, this is much less likely with a less extensive corpus or a popular topic. The corpus may still be able to serve as the source of an answer, but only through inference.

Textual entailment captures the intuition that one piece of text – in this case, text containing a correct answer to a given question – can be inferred from another. To date, progress has been gauged by the PASCAL Recognising Textual Entailment (RTE) Challenge (Bar-Haim et al., 2006; Dagan et al., 2008b).

Formally, the textual entailment task is to determine, given some text, *T*, and hypothesis, *H*, whether the meaning of *H* can be inferred from the meaning of *T*. For QA, *H* would be derived from a question such as:

(13) How many inhabitants does Slovenia have?

and *T* would be an answer passage such as:

In other words, with its 2 million inhabitants, Slovenia has only 5.5 thousand professional soldiers.

Recognizing the value of RTE to QA, the RTE Challenge includes manually annotated *T–H* data for training and testing, based on question–answer passage pairs from TREC (http://trec.nist.gov) and CLEF QA (http://clef-qa.itc.it) (see Chapter 10, LINGUISTIC ANNOTATION, for a discussion on annotating language resources). To create the *T–H* pairs, annotators extract from the answer passages answers of the correct type – but not necessarily the correct answer. The original question with the selected answer term included becomes *H*, and the answer passage becomes *T*. If, from example 13, annotators chose the answer '2 million inhabitants' from *T*, they could create a positive entailment pair, with 'Slovenia has 2 million inhabitants' as *H*. Alternatively, if they chose '5.5 thousand' from *T*, they could create a negative entailment pair with 'Slovenia has only 5.5 thousand inhabitants' as *H*.

The RTE challenge has tested systems on a 50–50 mix of positive and negative *T–H* pairs, where 50 percent would be the baseline accuracy. For the second RTE challenge, most systems obtained between 55 percent and 61 percent (Bar-Haim et al., 2006), where the majority of approaches drew on some combination of lexical overlap (often using WordNet; Miller et al., 1990), semantic role labeling, using FrameNet (Baker et al., 1998), and extensive use of background knowledge. The top-performing system was LCC's GROUNDHOG (Hickl et al., 2006), which achieved an accuracy of 75.4 percent. Around 10 percent of this score was obtained by expanding the training set of positive and negative entailment examples by following Burger and Ferro (2005), collecting around 100,000 positive examples consisting of the headline and first sentence of newswire texts. To create negative examples, they extracted pairs of sequential sentences that included mentions of the same named entity from a newswire corpus. Sample testing with human annotators determined that both of these example sets were accurate to the ninetieth percentile.

Although, intuitively, we might expect a deeper level of analysis to be required to achieve high accuracy in the RTE task, systems that have employed such analysis have so far failed to improve over the 60 percent baseline performance achieved by simple lexical matching, a technique already exploited in QA. Any future breakthrough in RTE is likely to be quickly and widely adopted in QA and elsewhere.

3.2 *Broadening the range of answerable questions*

We noted in Section 1.3 that open domain QA has primarily addressed *factoid* questions of *who*, *what*, *when*, *how many*, and *where* – in particular, ones that can be answered with a word or short phrase from the corpus. Little effort has gone into discovering systematic ways of answering *why* and *how* questions, which usually require longer answers. This may primarily have to do with the problem of evaluating such answers. And completely ignored are polar ('yes'/'no') questions such as

(14) Is pinotage another name for pinot noir?
(15) Is calcium citrate better absorbed and a more effective treatment for osteoporosis than calcium carbonate?

Scenario 1: The al-Qaida Terrorist Group
Your division chief has ordered a detailed report on the al-Qaida Terrorist Group due in three weeks. This report should present information regarding the most essential concerns, including who are the key figures involved with al-Qaida along with other organisations, countries, and members that are affiliated, any trades that al-Qaida has made with organisations or countries, what facilities they possess, where they receive their financial support, what capabilities they have (CBW program, other weapons, etc.) and how have they acquired them, what is their possible future activity, how their training program operates, who their new members are.

Figure 22.2 An ARDA scenario (from Small & Strzalkowski 2009).

because finding and presenting good evidence for an answer is more important than the answer itself and because the problem of assessing the quality of evidence has not really been addressed.

Nevertheless, there are now efforts to move beyond factoid QA in order to address questions that require information from more than a single source text or from multiple paragraphs within a text, possibly coupled with inference as above. These are called *complex questions*, the simplest form of which Bouma et al. (2005) have called 'which questions,' as in

(16) Which ferry sank southeast of the island Utö?

Answering this requires combining evidence that some entity sank southeast of the island Utö with evidence that the entity is a ferry. Complex QA often requires additional knowledge to process (from the data, from a model of the world, or from the user), and is used to accumulate evidence to support a particular position or opinion. This moves QA significantly away from the text retrieval dominated paradigm of factoid QA. The US ARDA AQUAINT program has been a major driver in the development of systems to address complex questions, often in the context of an explicit task or *scenario*, such as in Figure 22.2.

Complex QA has been approached in different ways, two of which we describe here: (1) decomposing the problem into subproblems and then dealing with each one in turn; and (2) more complex document/passage indexing.

In question decomposition, a complex question is reduced to a set of subquestions, using linguistic and/or domain-specific knowledge. The subquestions are meant to be ones that can be answered by existing factoid QA technology. For example, the START system (Katz 1997) syntactically decomposes the question

(17) Who was the third Republican president?

into the sequence of questions Q1 (*Who were the Republican presidents?*), followed by Q2 (*Who is/was the third of them?*), where 'them' stands for the list of answers to Q1. Saquete et al. (2004) perform similar syntactic decomposition in answering temporal questions such as

(18) Where did Bill Clinton study before going to Oxford University?

which gets split into the pair of questions Q1 (*Where did Bill Clinton study?*) and Q2 (*When did Bill Clinton go to Oxford University?*), with a constraint that the answer to Q1 fall in the period *before* the time associated with Q2.

The problem with syntactic decomposition is that it quickly becomes computationally expensive to generate and evaluate all syntactically legal decompositions of a complex question. For the question

(19) What American companies have introduced a generic drug in a European country?

possible subquestions include *What companies have introduced a drug in a European country?*, *What American companies have introduced a generic drug?*, and so on. One really only wants to generate those which have answers in the corpus, and that is the goal of the second way of addressing complex questions – complex indexing.

In Section 2.2, we mentioned various ways in which text is indexed for QA – by words (or stems), by named entities, by part-of-speech tags, and by dependency relations. Even more complex indexing is used, either alone or with question decomposition, in the *parameterized annotations* used in START (Katz et al., 2005) and elsewhere, as a way of answering complex questions. Parameters take the form of domain-specific templates with parameterized slots – such as the following, from Katz et al. (2005):

In <year>, <group type> <group name> carried out a <event type> in <country>, involving <agent type> <agent name>.

When used to index a passage, slots would be replaced by appropriate named entities. Berrios et al. (2002) use similar templates to index paragraphs in medical textbooks, in order to answer common clinical questions, including:

(20) What is the <Pathophysiology/Etiology> of <Manifestation/Pathology>?
(21) How can <Pharmacotherapy> be used to treat <Disease/Syndrome>?

Here the parameters are disjunctions of Unified Medical Language System (UMLS) categories (www.nlm.nih.gov/research/umls/) that can be filled by a term or phrase that belongs to any one of them.

Templates such as these function in a similar way to syntactic decomposition, in that they can collect lists of responses for each subquery, then perform constraint propagation to find responses that satisfy all parts of the original query. However, these templates need to be built for each domain or data resource under consideration by the QA system – a significant overhead.

The LCC system (Lacatusu et al., 2005) achieves question decomposition in a similar way. Top-down decomposition is driven by syntactic information. Bottom-up decomposition is achieved by using predictive question–answer pairs (stored in a question–answer base, and referred to as QUABs) to expand the answer space. These QUABs are selected from a database of 10,000 question–answer pairs created offline by human annotators. So for a question such as *What has been the impact of job outsourcing programs on India's relationship with the US?*, those QUABs which closely match the question, and are therefore adjudged to anticipate the user's information need, are presented to the user. Both methods of decomposition are powerful approaches, but require a high degree of manual knowledge construction – creating templates or the construction of question–answer pairs – in order to be effective. Some initial work has been undertaken to incorporate existing large-scale knowledge sources, such as CYC (Curtis et al., 2005), into the complex QA scenario, without yielding significant improvements in either domain-specific or open domain QA, possibly due to the incompleteness of the information represented in these knowledge sources.

3.3 Relation between user and system

Open domain QA has basically assumed a single correct answer to every factoid question, very much a reflection of its origins in IR, which abstracts individual relevancy judgments away to those of an average user, to serve as a *gold standard*. But it should be clear that different correct answers will be appropriate for different users. The most obvious example of this comes in the degree of specificity desirable in answering *where is* questions, such as:

(22) Where is the Verrazano-Narrows Bridge?
(23) Where is the Three Gorges Dam?

where someone in North America might appreciate a specific answer to the first (e.g., New York City, or between Brooklyn and Staten Island) and a general answer to the second (e.g., China), while for someone in Asia, the reverse might be true (e.g., New York, and Western Hubei Province).

Addressing another aspect of tailoring responses to users, Quarteroni and Manandhar (2009) show how a user's age and assumed reading level can be used to choose the most appropriate among possible correct answers to a question. Alternatively, rather than trying to choose the most appropriate among possible correct answers, a different approach would be to offer all of them, allowing users to choose for themselves, as in the answer model for the question *Where is Glasgow?* given in Figure 22.3.

Dalmas and Webber (2007) show that a similar approach can be taken in presenting answers to ambiguous questions, such as the famous question *Where is the Taj Mahal?* (the tomb? the casino/hotel? the local Indian restaurant? etc.)

But the main way of enriching the relationship between user and QA system involves the use of interaction. In Section 1.2 we described early systems that could engage in dialogue in the process of providing answers to user questions. Such systems relied on complete descriptions of the target domain in order to choose the next dialogue move, and so could only work in conceptually simple domains

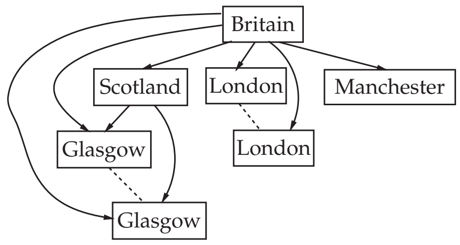


Figure 22.3 An answer model for the question: *Where is Glasgow?* (Dalmas & Webber 2007), showing both Scotland and Britain as possible answers. Here → conveys a part-of relation, and a dashed line, an equivalence relation.

such as travel planning (Goddeau et al., 1994; Seneff 2002), finding an appropriate restaurant (Walker et al., 2004), automated banking (Hardy et al., 2005), etc. In more complex applications, a complete description of the domain is unlikely, and systems must be able to ask questions of their own in order to work towards providing the user with the information he/she is after. On the other hand, as user questions become more complex, it becomes more difficult to anticipate all possible ways of answering them, and systems require dialogue to help guide them to what it is that the user wants. This is the issue that we focus on, under the rubric *interactive question answering* or *IQA*.

IQA can be seen as a process in which the user is a continual part of the information loop – as originator of the query, arbitrator over information relevance, and consumer of the final product. This mode of operation is useful for both factoid and complex QA, but perhaps provides greatest benefit in those cases where the user's information need is still vague, or involves complex multifaceted concepts, or reflects misconceptions or opinions – all cases where the expected information content returned is complex, with a degree of variability not present in factoid QA. Interactive QA systems borrow from dialogue systems their interaction, their emphasis on completion of user task, their handling of incomplete or underspecified (as well as overspecified) user input and the constraint and relaxation phases of the query process, while remaining focused on large or open domains, such as law, biomedicine, or international politics.

Rather than attempting to resolve complex question ambiguity independently of either the user or the context, both can be used to generate follow-up queries to the user that can serve as significant directional tools to guide the information seeking process. To achieve this requires an understanding of the context of the user's query and some knowledge about the domain of inquiry, or at least the domain as represented by results returned by the specific query.

In the previous section, we described the use of predictive question–answer pairs (QUABs) to expand the range of answerable questions. They are also used in the interactive QA system FERRET (Harabagiu et al., 2005) to support interaction. When the user asks a question, those QUABs that relate to the query are offered to the user, as a way of expanding the search space. These QUABs can address not only subquestions decomposed from the original question, but also other information that may be tangentially related to the question. However, this data is often created manually, and thus is an expensive overhead. HITIQA (Small & Strzalkowski, 2009) exemplifies an IQA system that does not rely on *a priori* knowlege of the domain. Rather, HITIQA creates just-in-time semantic representations of retrieved passages, using domain-independent 'frames' – approximations of the underlying relationships between entities. While less precise or targeted than the parameterized annotations used in the START system (Katz et al., 2005) described in Section 3.2, they are sufficient for capturing the essence of a paragraph.

A similar framing process is applied to the questions posed to the system, creating a goal frame. This allows for systematic comparison between the analyst's question and the retrieved text passages and for selection of instantiated frames for answer passages. HITIQA uses interactive dialogue with the analyst to negotiate the scope of the answer, and experiments with several groups of analysts showed that users accept system suggestions regarding the current question scope as well as those that relate to other aspects of their task – thus speeding up their work. Dialogue is enabled through the frame-based representation of content and more specifically through the mismatches between the goal frame and the data frames in the answer space. This is best illustrated through the example interaction taken from a live demonstration to the ARDA AQUAINT community in 2005 in Figure 22.4.

This interaction exploits the goal frame that was created for the question (Figure 22.5) and the frames built for each of the retrieved passages (Figures 22.6 and 22.7).

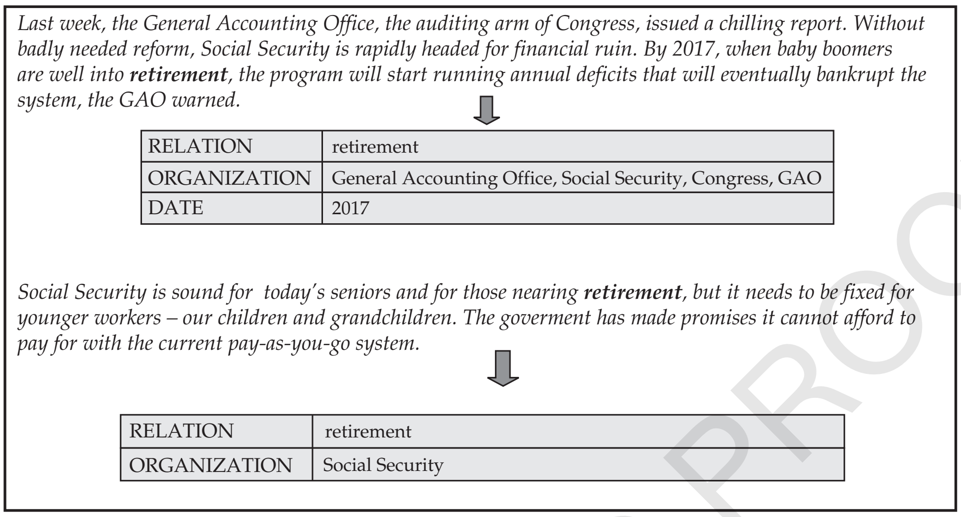
The passages in Figures 22.6 and 22.7 are represented by one-conflict data frames in the analyst's answer space (i.e., where there is one conflict between the data frame and the question frame). The conflicts on the relation attribute were, respectively: status vs. retirement and status vs. private accounts. These relations

Analyst: *"What is the status of the Social Security System?"*
[exact match answer passages are displayed]
HITIQA: *"Would you be interested in information on **retirement** relative to your question?"*
Analyst: *"Yes"*
HITIQA: *"Do you want to see information on **private accounts**?"*
Analyst: *"Yes"*
HITIQA: *"Thank you, Please view your answer"*
[final selection of passages are displayed]

Figure 22.4 Example interaction taken from a live demonstration to the ARDA AQUAINT community in 2005.

RELATIONS	status
ORGANIZATION	Social Security System

Figure 22.5 Goal frame for the question: *What is the status of the Social Security system?*



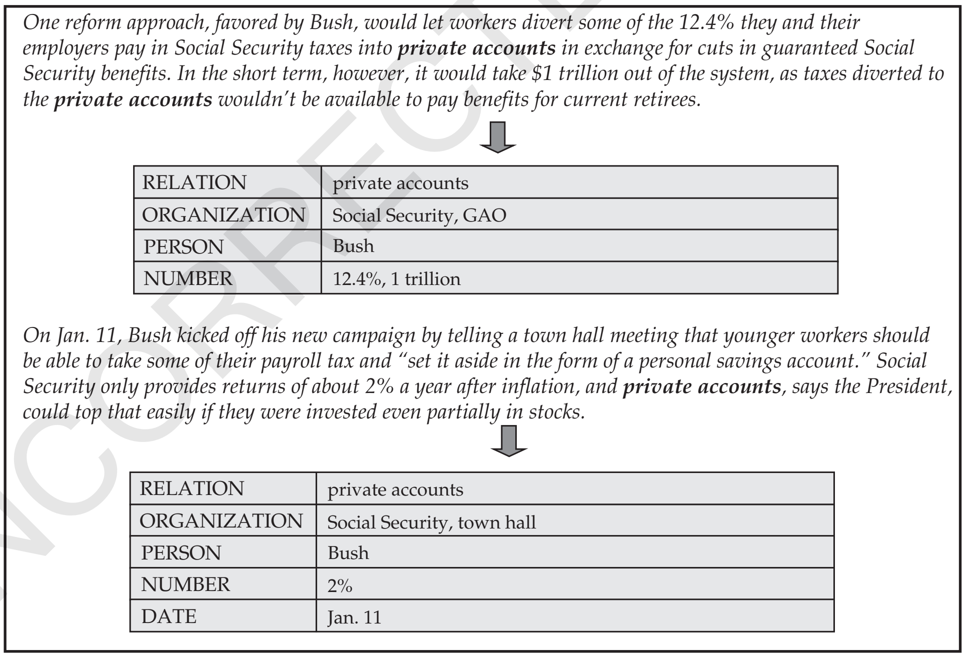
*Last week, the General Accounting Office, the auditing arm of Congress, issued a chilling report. Without badly needed reform, Social Security is rapidly headed for financial ruin. By 2017, when baby boomers are well into **retirement**, the program will start running annual deficits that will eventually bankrupt the system, the GAO warned.*

RELATION	retirement
ORGANIZATION	General Accounting Office, Social Security, Congress, GAO
DATE	2017

*Social Security is sound for today's seniors and for those nearing **retirement**, but it needs to be fixed for younger workers – our children and grandchildren. The goverment has made promises it cannot afford to pay for with the current pay-as-you-go system.*

RELATION	retirement
ORGANIZATION	Social Security

Figure 22.6 Two cluster seed passages and their corresponding frames relative to the retirement clarification question.



*One reform approach, favored by Bush, would let workers divert some of the 12.4% they and their employers pay in Social Security taxes into **private accounts** in exchange for cuts in guaranteed Social Security benefits. In the short term, however, it would take $1 trillion out of the system, as taxes diverted to the **private accounts** wouldn't be available to pay benefits for current retirees.*

RELATION	private accounts
ORGANIZATION	Social Security, GAO
PERSON	Bush
NUMBER	12.4%, 1 trillion

*On Jan. 11, Bush kicked off his new campaign by telling a town hall meeting that younger workers should be able to take some of their payroll tax and "set it aside in the form of a personal savings account." Social Security only provides returns of about 2% a year after inflation, and **private accounts**, says the President, could top that easily if they were invested even partially in stocks.*

RELATION	private accounts
ORGANIZATION	Social Security, town hall
PERSON	Bush
NUMBER	2%
DATE	Jan. 11

Figure 22.7 Two cluster passages and their corresponding frames relative to the private accounts clarification question.

were obtained from the passage's cluster relations: one cluster containing retirement (Figure 22.6) and the other cluster containing private accounts (Figure 22.7). From the system's viewpoint each of these clusters represents an alternative interpretation of the user's question about the status of Social Security, and so each is offered to the user through automatically generated clarification questions. This data-driven approach allows the system to work effectively in any domain and with any text genre, without necessitating costly knowledge engineering.

3.4 *Evaluating extended QA capabilities*

As QA broadens and deepens to include the extensions described in this section, so evaluation methods need to develop so that we can continue to assess how well a method performs and/or how much better it is than the alternative(s).

In Section 2.4, we described some of the metrics used and the issues involved in evaluating system performance on factoid questions, where it is assumed that specific *gold standard* answers can be identified and agreed upon prior to evaluation. (See also Chapter 11, EVALUATION OF NLP SYTEMS.) Here, for each way of extending QA from where it is today, one can consider whether existing evaluation methods suffice or whether new ones are needed.

Extending the relation between question and corpus is a matter of *textual entailment* (Section 3.1), so methods used in assessing correctness in that task should apply here as well, provided that QA systems identify the minimal set of sentences within a document that entails the provided answer (rather than simply a pointer to the document).

For answering *why* and *how* questions, nugget methods (Dang & Lin 2007) should suffice initially, though one might also want to assess temporal, causal, and/or subsumption relations that should be seen to hold between the nuggets. Evaluating performance on answering *polar questions* could also be taken to be a matter of textual entailment, where providing the minimal set of evidential sentences would again be crucial for evaluation, not least for user acceptance. For what Bouma et al. (2005) have called *which* questions, standard QA metrics should suffice, though, as with textual entailment, systems should probably also identify the set of sentences (within a document or across multiple documents) that were used in computing the answer.

It is in the more complex relation between user and system that one finds a challenge for QA evaluation. But even here, there is a gradient. Quarteroni and Manandhar (2009) assess a QA system's ability to tailor answers to a user's age and reading level by asking outside assessors to provide judgments on the answers. For simple information needs, such as a list of the names of the N upcoming conferences on language technology, along with their locations and acceptance rates (Bertomeu et al., 2006) – i.e., the sort of information that one might be able to get from a database if such a database existed – evaluation metrics developed in dialogue system technology such as task completion, time to task completion, and user satisfaction (Walker et al., 2000) would probably be equally effective here. But unlike the setup where assessors are considered the final consumers of answers, as they are in a gold standard model, here motivated users must become system assessors.

This is even more true as user information needs become more complex and users may not even know what information might be relevant. Here there has been an attempt to develop objective evaluation criteria – in the ciQA track at TREC (www.umiacs.umd.edu/∼jimmylin/ciqa/), but see also iCLEF (Gonzalo & Oard 2004; Gonzalo et al., 2006, and ACLIA (Advanced Cross-lingual Information Access) track (http://aclia.lti.cs.cmu.edu/wiki/moin.cgi/Home) in NTCIR. We want to argue, however, that ciQA and its evaluation criteria do not reflect the true character of users attempting to solve complex information needs.

A key element in a successful evaluation paradigm is the definition of the central concepts. Evaluation metrics have gained increased prominence in recent years, as they often influence the research agenda. Unfortunately, ciQA's view of complex interactive QA reflects TREC's IR bias, and consequently fails to showcase the true potential of the interaction process. In particular, it assumes that the user's information need does not have to be expressed via natural language queries: it can be expressed in a template that represents the question in canonical form. Only additional context (the *narrative*) is provided in natural language. For example:

> Template: What evidence is there for transport of <drugs> from <Bonaire> to <the United States>?
> Narrative: *The analyst would like to know of efforts made to discourage narco traffickers from using Bonaire as a transit point for drugs to the United States. Specifically, the analyst would like to know of any efforts by local authorities as well as the international community.*

This narrative elaborates on the information need, providing context or a finer-grained statement of interest, or a focus on particular topical aspects. While this narrative could be used automatically by systems that adopt a question decomposition approach, or could be exploited by a motivated user, to drive an interaction around tangential issues surrounding the central concepts, ciQA evaluation has not been designed to discriminate between any attention being paid to the narrative or not.

This template approach seems similar to the use of such templates in some of the decomposition approaches discussed in Section 3.2, but with a fundamental difference. Whereas those approaches attempt to process large amounts of data into a form that has the potential to match a range of queries, here the templates are known *a priori*, greatly reducing the problem space. Within IR, when evaluation was fixed on a known data set, and fixed set of associated queries, performance increased over time, but again many approaches failed to scale when the data set increased, or substantially changed in form and content.

A second problem with ciQA with respect to showcasing the true potential of interaction is its limited definition of the term. In particular, it allowed only a single interaction (2006) or single five-minute period of interaction (2007). This single interaction or period of interaction reflects the mistaken view (deriving from an idealization adopted in IR) that interaction involves no more than refining the original query to better reflect the available data. However, as Wizard-of-Oz experiments into the use of interaction in QA continue to show (cf. Bertomeu 2007), success and satisfaction can depend on support for a variety of different interchanges and answer strategies, including clarification subdialogues, explanation subdialogues, responses that *overanswer* a question (providing more information and more kinds of information than the user actually requested), being unaware of their relevance when the question was originally asked.

Since it is clear that neither the type of interaction supported in ciQA nor its evaluation criteria (in terms of the ‘information nuggets’ contained in the system responses) are a fair representation of the capabilities of a true interactive system, it should be no surprise that the results of the first ciQA event were largely negative – that standard IR approaches performed as well as complex QA approaches. Of the 12 individual runs in 2006, only two scored higher than the manual baseline sentence retrieval system. Thus Kelly and Lin (2007) note that ‘interaction doesn’t appear to help much (at present),’ although he acknowledges both that the evaluation design could be flawed (in limiting interaction to a single turn, and limiting the time for that interaction), and that the types of interaction deployed (a variant of relevance feedback) are not appropriate for this kind of task. Indeed, Lin (2007) points to a truism that should be very revealing. Current IR paradigms maximize recall – by returning *more of the same*. Complex QA, in contrast, values novelty – what information the user has not seen, that may present another aspect of the scenario. Some systems, such as HITIQA (Small & Strzalkowski 2009), explicitly encode this assumption, remembering in the dialogue history both what users have seen, and what they have indicated they like, versus what does not interest them. This does not have to be explicit – interest can be gauged by a user copying information into a report, or dismissing a window after just a few seconds, for instance.

Perhaps a more useful indicator of evaluation is related work by Kelly et al. (2009), which addresses the issue of instability of traditional evaluation metrics in multi-user environments, and describes the use of questionnaires to evaluate a range of IQA systems, as a method of garnering effective user feedback about the systems themselves, and involving users in a subjective evaluation process. Key is the ability to discriminate between the resulting systems on the basis of several hypotheses, such as effort and efficiency. However, the assumption here, reflecting a fact that the field will have to face fairly soon, is that effective evaluation of complex interactive QA systems can only truly be achieved by subjective evaluations by motivated users. This is certainly more expensive, and less statistically clear cut than previous evaluation paradigms for QA, but may lead us to real discoveries of strategies for complex interactive question answering.

4 Further Reading

A survey of work on database QA can be found in Webber (1986), while the most comprehensive description of a database QA system remains Woods (1978). Prager (2007) provides the best introduction to open domain QA, while an incisive component-by-component performance analysis of an open domain QA system can be found in Moldovan et al. (2003). For collections of papers on open domain QA, the reader is referred to Maybury (2003) and Strzalkowski and Harabagiu (2006), and to special issues of the *Journal of Applied Logic* (on *Questions and Answers: Theoretical and Applied Perspectives* 5:1, March 2007) and of the *Journal of Natural Language Engineering* (cf. Webb & Webber 2009, a special issue on interactive question answering). Papers on QA appear regularly in conferences sponsored by the Association for Computational Linguistics (ACL) and by the Special Interest Group on Information Retrieval (SIGIR).

NOTE

1 Even when pronoun resolution is not used to support enhanced indexing, but only answer candidate extraction and/or evaluation, it has been shown to be of benefit and worth the cost (Vicedo & Ferrández 2000).