Is Question Answering an Acquired Skill?

Ganesh Ramakrishnan
IIT Bombay

Soumen Chakrabarti*
IIT Bombay and CMU

Deepa Paranjpe
IIT Bombay

Pushpak Bhattacharyya
IIT Bombay

ABSTRACT

We present a question answering (QA) system which learns how to detect and rank answer passages by analyzing questions and their answers (QA pairs) provided as training data. We built our system in only a few person-months using off-the-shelf components: a part-of-speech tagger, a shallow parser, a lexical network, and a few well-known supervised learning algorithms. In contrast, many of the top TREC QA systems are large group efforts, using customized ontologies, question classifiers, and highly tuned ranking functions. Our ease of deployment arises from using generic, trainable algorithms that exploit simple feature extractors on QA pairs. With TREC QA data, our system achieves mean reciprocal rank (MRR) that compares favorably with the best scores in recent years, and generalizes from one corpus to another. Our key technique is to recover, from the question, fragments of what might have been posed as a structured query, had a suitable schema been available. One fragment comprises *selectors*: tokens that are likely to appear (almost) unchanged in an answer passage. The other fragment contains question tokens which give clues about the *answer type*, and are expected to be *replaced* in the answer passage by tokens which *specialize* or *instantiate* the desired answer type. Selectors are like constants in `where`-clauses in relational queries, and answer types are like column names. We present new algorithms for locating selectors and answer type clues and using them in scoring passages with respect to a question.

Categories and subject descriptors: [H.3.1 Content Analysis and Indexing; Linguistic processing] [H.3.3 Information Search and Retrieval; Query formulation, Retrieval models]. **General terms:** Algorithms, Experimentation. **Keywords:** Question answering, machine learning.

1. INTRODUCTION

A *QA System* responds to queries like *Who is the Greek God of the Sea?* with a precise answer like *Poseidon*. An important first step is to identify short *snippets* or passages of up to several words which contain the answer.

In this work we focus on open-domain systems that are of interest to the Information Retrieval (IR), Information Extraction, and Message Understanding communities. Falcon [14], Webclopedia [15], Mulder [18], AnswerBus [28] and AskMSR [11] are some well-known research systems, as are those built at the University of Waterloo [7, 8], and Ask Jeeves (`http://ask.com`).

* Contact author, email `soumen@cse.iitb.ac.in`.

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

Most QA systems are substantial team efforts, involving the design and maintenance of question taxonomies [14, 15], question classifiers, and passage-scoring heuristics. E.g., the Webclopedia team has compiled a taxonomy of 180 types of question targets[1] after analyzing over 27600 questions from `Answers.com`. The wisdom derived by QA experts from the corpus and task is embedded in rules like this [16]:

> Matching an upper-cased term adds a 60% bonus ... for multi-words terms and 30% for single words ... Matching a WordNet[2] synonym ... discounts by 10% (lower case) and 50% (upper case) ... Lower-case term matches after Porter stemming are discounted 30%; upper-case matches 70%.

Owing to these myriad details, it is very difficult to reproduce a well-tuned QA system from scratch, gauge the benefits of new algorithmic ideas, and generalize the tuning experience to new domains, new corpora, and new languages.

Compare the QA situation with IR engines, which are largely based on the now-standard vector-space model and TFIDF ranking, a *declarative* specification of *what* is a good matching document. The *procedural* details of *how* to retrieve the top-ranking documents have only efficiency implications. Consequently, off the shelf IR engines (Lucene, Glimpse, htDig, Verity) require essentially no tuning, and can be deployed in minutes.

In contrast, QA systems not only depend on complex building blocks like taggers and parsers (which we see as inevitable to an extent) but lash them together with customized "glue" and many crucial knobs that are best preset by QA specialists rather than the end-user. This might explain in part why off-the-shelf QA packages are rare.

1.1 Our goal

We seek to decompose the QA task cleanly into discovering features and learning to score potential answer snippets from a corpus and past question-and-answer examples. Our QA system

- performs fast, shallow processing of the corpus
- structures the scoring task using *features* and *learners*, which are cleanly separated
- trains its scoring algorithm from a past history of pairs of questions and vetted answers
- can include side information (WordNet, almanac) as a natural part of the learning process
- reuses expertise accumulated from one corpus to a new corpus
- can be reproduced easily from a complete yet concise description

[1] `http://www.isi.edu/natural-language/projects/webclopedia/Taxonomy/taxonomy_toplevel.html`
[2] A lexical network [22].

Our QA system is trained using QA pairs [2, 19]. A trainable QA system also eases the task of adapting to a new genre of corpus and even new languages. QA pairs can be acquired for training from a number of sources. FAQs abound on the Web; indeed, there is even a markup language, QAML[3], to represent FAQs. Search engines validate any new ranking tweak by regression tests across large query-response collections. Audit trails of paid services (like Google Answers[4]) can provide a wealth of QA-pair sources.

In this paper, we will be concerned with passage/snippet scoring. We will not discuss the final step of answer extraction from passages. Even without the final step, QA systems can reduce the searcher's burden from visiting dozens of URLs to browsing only a dozen lines of snippets.

1.2 Our contributions

We start in §2 with a novel **noisy simulation** perspective on the QA task. In a structured database with a suitable schema and a structured query language, information needs can be expressed precisely. We see QA as a *transformation* of this process by adding natural language from an unknown generative process, for both the data and the query.

Guided by this framework, we work backwards: given a question, we discover structured fragments in it, glimmerings of a structured query which would have been possible, had there been a schema. Specifically, we extract **selectors** which we are confident will appear (almost) unchanged in an answer passage, and we extract **atype clues**, which tell us what else to look for in a passage that has satisfied all selectors.

We learn the process of finding selectors and atype clues automatically from a training collection of question-and-answer pairs, where an "answer" is a passage with the **answer zone** specifically marked, as shown:

```
<q>Name Pittsburgh's  baseball team.</q>
<a>Johnson was president of the <zone>Pittsburgh
Pirates</zone> baseball team for a time.</a>
```

We can flag selectors with about 73–80% accuracy. We also describe a simple procedure to extract atype clues via a shallow parse of the question, such as can be performed readily by the Link Parser [25].

Next, we propose a simple and intuitive ranking procedure by combining information from the selectors, atype clues, and passage words. This procedure is tuned by automatic QA-training. All the important procedures of our system are described in §3.

Given infinite data, we may hope QA-pairs to suffice in training our ranking algorithm, but in reality, the extreme sparsity and high dimensionality of our learning problems makes it worthwhile to help the system through generic *biases* in the form of known relationships between words. We use WordNet [22], a lexical network.

We use data from the TREC QA competition to show that our basic approach and specific learning methods are effective (§4). From a baseline in which we index passages and return passages in the order returned by a typical IR engine, we get substantial increases in MRR (mean reciprocal rank, see §4.4) by our techniques. We also show that our system can learn on one year's worth of TREC data and generalize to data from a different year, where even the corpus is different.

[3] `http://www.ascc.net/xml/en/utf-8/qaml-index.html`
[4] `http://answers.google.com`

1.3 Related work

The broad architecture of QA systems [7, 14, 15, 18, 28, 8, 24] has become standard. The corpus is segmented into documents or passages of some suitable size. A named entity (NE) tagger marks people, places, organizations, etc. [1], and the corpus with annotations is indexed by an IR system. A taxonomy of (hundreds of) question types is built by hand, and question classifiers designed to label questions with types. Based on the label, a new question is first transformed into a keyword query for the IR system. Responding passages are reranked using a variety of strategies, usually involving some natural language analysis.

Clarke *et al.* [9] showed that a limited window of sentences is acceptable as the unit within which to look for exact answers. Tellex *et al.* [26] showed that 3 to 6 sentence-long passages gave the best accuracy, and claimed that simple Boolean queries with TFIDF style ranking (as provided by Lucene) are good enough to short-list passages for reranking. These numbers confirm earlier choices made by Abney *et al.* [1].

Our approach is closest in methodology to the work of Agichtein *et al.* [2] who learn from QA pairs how to *transform* a question into words and short phrases likely to appear in an answer. Frequent itemset and co-occurrence detection is at the core of their system. A related paper [19] explores how to pick one answer from a small set of candidate answers, as in the *Millionaire* TV show. While inspired by these ideas, we go further in our precise QA model and processing of the query as well as the trainable passage ranking procedure.

AskMSR and variants [11] use a seven-way classification of the question followed by a customized rewrite for each class. More importantly, AskMSR makes a strong case favoring data-intensive QA over knowledge-intensive QA. A search for passages satisfying the proximity query `bjorn borg NEAR wimbledon NEAR [numeric-literal]` is likely to retrieve the answer to the question "How many times did Bjorn Borg win Wimbledon?" and no deeper understanding of language may be needed in many cases. These observations form the launching-pad of our system.

We regard question-processing as extracting fragments of a structured query from among some natural language "noise." This approach is reminiscent of an area of database research [4] that seeks to translate natural language queries to SQL. Those systems can exploit the well-defined schema that underlies the data, which is missing in the QA scenario). With recent and notable exceptions [23], these systems are often based on rigid transformations of question parses into SQL syntax trees.

2. THE NOISY SIMULATION MODEL

If all natural language texts were replaced by relational or semi-structured databases, and people asked questions in structured query languages with precise semantics, there would be no need for QA systems.

Even with structured data, precise queries are hard on the searcher because s/he does not want to master the schema. E.g., a user may prefer to type `avg(salary (near physics))` in place of the structured query `select avg(salary) from payroll where dept='physics'`. In the RDBMS and XML communities, recent research has led to systems [13, 6, 3, 17, 23] that allow such schema-less access.

These systems have to infer that `physics` is a constant in

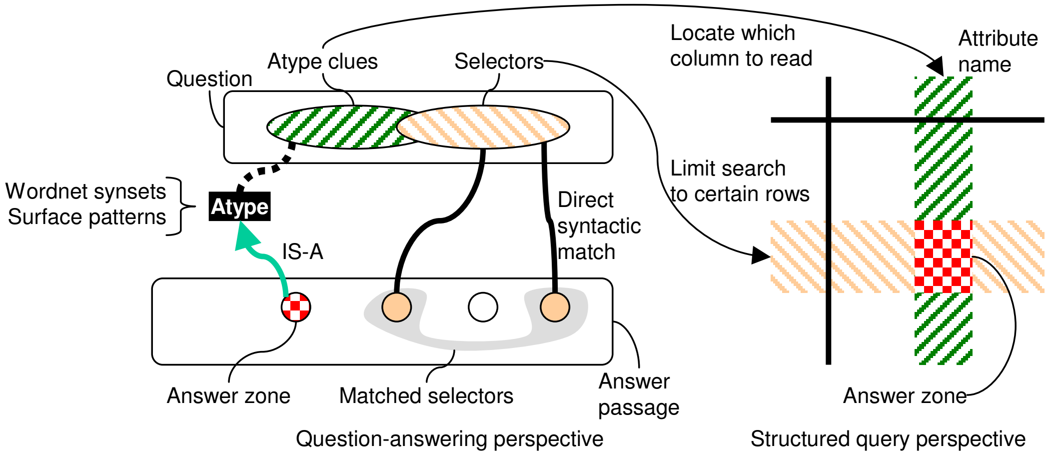

Figure 1: A pictorial view of the typical connections between a question and its answer, which helps us design our basic query processing strategy.

a select clause which will match the value of a column in a tuple (which we therefore call a **selector**), `salary` is the name of a column (which we therefore call an **answer type** or **atype** for short), and `avg` is an aggregator.

The loss of schema in the question can lead to some ambiguity: should we include adjunct and courtesy appointments? Should we include the stipend of a Chemistry student taking a Physics course? Yet, the fact that there *is* a schema helps making some sense of the query.

In QA, neither the question nor the corpus adheres to a fixed schema. However, given a factoid question, it is not hard to conceive of a simple relation with very few attributes whose rows can be matched with the question to generate the answer, if only such a table were populated on-demand by extracting information from the corpus (Figure 1). The trick, then, is to achieve the matching without the intermediation of the schema, assuming that all the information needed to populate a row in our phantom relation can be gleaned from a short passage.

A key step is to regard the question as a transformation of a structured query of the form `select...where...` to natural language, and try to recover the structured query by filling in the two blanks.

In the structured query `select...where...`, the contents of the second blank restrict our attention to some rows, and the first blank tells us which column to read off. So the first blank has to be filled with a *type* to be specialized to an *instance*, and the second blank has to be filled with one more more *constants*. Each word in a question can contribute information to filling in either (possibly both) blanks. In the remainder of this section, we discuss how to identify words of each type in the question. First we take a brief detour to describe WordNet, a lexical network.

2.1 WordNet primer

For our purposes, WordNet [22] is a graph where nodes are concepts and edges are relations between concepts. A concept is called a **synset** because it is described by a set of synonyms, also called **lemmas**. A synset may be described by many lemmas. Conversely, a lemma (like *match*) can describe many synsets, in which case it is highly **polysemous**. A lemma, a part-of-speech, and a (standardized) sense number together defines a synset uniquely, and is written as `match#n#1` (first noun sense of *match*). We only consider **hypernym** and **hyponym** edges in WordNet, which represent IS-A relations. E.g., in this chain of generalization:

> horse, Equus caballus → equine, equid → ⋯ → ungulate, hoofed mammal → placental, placental mammal, eutherian, eutherian mammal → mammal → ⋯ → animal, animate being, beast, brute, creature, fauna → ⋯ → entity

beast and *brute* are synonyms, *equid* is a **hyponym** of *ungulate*, *horse* is a **hyponym descendant** of *mammal*, *equid* is a **hypernym** of *horse*, and *entity* is a **hypernym ancestor** of *horse*.

2.2 Atype

Most factoid questions[5] have answers that are specific instances of broad classes of real-world entities, such as person, animal, time, duration, length, place, proper name, cardinal number, money amount, etc. A human can usually identify with ease a minimal subclass of entities which will answer a given question. We call this the **atype** of the question (and its answer).

To be able to compute with atypes, they must be represented somehow, and connected to all possible instances. We will consider two representations which are important to factoid QA: WordNet synsets and syntactic surface patterns.

Atype as synset: The imperative question *Name an animal that sleeps upright* is answered by *horse*. WordNet helps us recognize that *horse* is an instance (hyponym descendant) of *animal*, which appears in the question and is the atype of this question. If an atype and a passage token map to WordNet synsets with a suitably directed hypernym/hyponym chain, the token is a strong candidate for further evaluation. Most answers which are common nouns are assisted by this representation of atypes. WordNet is an obvious first resource to use for is-a information, but there is much research afoot [12] to extract ontologies from the Web.

Atype as surface patterns: Infinite or very large domains such as numbers, person names, place names, etc. cannot be covered by WordNet. However, we can logically augment WordNet to add connections from synsets to pattern

[5] Enumeration, aggregation, and true/false questions are beyond our current scope.

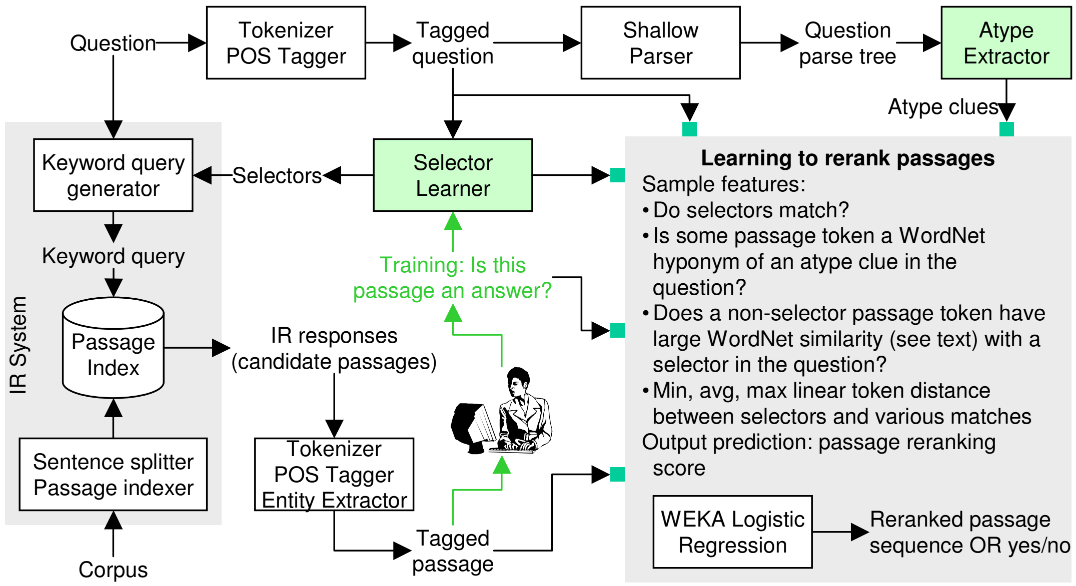

Figure 2: Overall architecture of our trainable QA system.

matchers (e.g., regular expressions) that work on the answer entity and/or its left and right lexical neighborhoods, such as "`at DD:DD`", "`in the 'DDs`", "`in DDDD`" or "`Xx+ said`", as in information extraction systems (here `D` is a digit and `X` is any uppercase letter, `x` is any lowercase letter). We can pretend they are like special synsets, and even connect them to appropriate synsets originally in WordNet. E.g., surface patterns for (calendar) dates can be associated with the synset `date#n#7`.

The accuracy of the system depends only on the lexical network, the choice of the surface features, and the learning algorithms used. Observe that we make no judgment about the relative importance of the two kinds of is-a relations, but let our learning algorithms figure this out based on the training corpus and QA pairs. If a third kind of atype specialization is devised later, it can be plugged in right away with no change in the algorithms.

The atype of a question is manifested through specific words (*Where*, *When*, *Who*) or by generic words (Which *city*, What is the *density*). We describe atype extraction in §3.1.

2.3 Selector

The second blank in the SQL query `select...where...` is the "where clause" which restricts the set of rows which will be considered by the query plan. In QA, the "where clause" is simply a set of words in the question that we expect will appear unchanged in a passage containing the answer. We call these words **selectors**. E.g., any passage that answers the question *Tokyo is the capital of which country?* is very likely to contain the word *Tokyo* in it, so *Tokyo* is a selector. The notion of a selector is "soft": in the question above, *capital* might also be a selector, but we are not as sure as with *Tokyo*; perhaps a passage writer might use "center of governance" with a small probability. *Country* is still softer: it is fairly likely that a synonym like *nation* may be used instead, or that neither *country* nor a synonym is used in the answer passage. (In fact, *country* is the atype.)

Adept users of text search systems, faced with the problem of satisfying a difficult-to-express information need, unconsciously determine selectors and then frame a series of keyword queries which share selectors but try out a variety of non-selecter alternatives.

There is an obvious trade-off between picking selectors eagerly and losing recall vs. picking selectors reticently and losing precision, given QA systems can afford to rerank only the top passages returned from the index, owing to high computational costs.

We mention in passing that schema-free keyword search in relational and XML databases [6, 17, 3, 23] face a similar problem of deciding whether a query keyword should map to metadata (name of a table, name of a column, element name, IDREF attributes) or data (text in relation cell or CDATA in element). Response-ranking in these settings is also an open area of research.

3. SUB-PROBLEMS AND SOLUTIONS

In this section we set up our problem more formally and propose approaches to identifying atype clues and selector words, connecting question and passage words through WordNet, and scoring passages. The overall architecture of our systems is shown in Figure 2. The broad outline is:

- The corpus is chopped into passages, and a shallow tagger such as GATE attaches annotations of surface patterns (§3.1). Passages are indexed by a standard IR system.
- QA pairs are provided. These help train a selector-finder (§3.2), an atype-finder (§2.2), and a passage reranking algorithm (§3.3). QA pairs also help build a **mapping** from atype clues to atypes (§3.1.2).
- When a new question arrives, the selector-finder proposes selectors which assist the initial keyword search, giving some number of top-ranking passages from the passage index.
- Each passage is combined with the question to form an unlabeled QA instance. The atype finder and the atype clue-to-atype mapping helps us extract a feature vector for the instance, and this is assigned a score by the passage reranking algorithm.

- Candidate passages are sorted by decreasing score and presented to the user.

3.1 From the question to an atype

Every language has a small set of common "wh-words" which express questions; for written English, these are *what*, *which*, *why*, *how*, *when*, *where*, *who*, *whom* and *whose* [21, Page 100]. Imperative questions can also begin with *name* and *define*.

Questions starting with *when*, *where* and *who* immediately reveal their expected atypes, e.g., *Where is Belize located?* is clearly asking for a place. *How*, by itself, gives scant clue about the atype, but the word after *how* is almost always a clue, as in *how many*, *how much*, *how long*, or *how often*[6]. Questions using *what* and *which* tend to mention the atype directly, as in *What is the capital of Japan?*.

As we describe next, atype identification for *what* and *which* questions is done through *shallow parsing* of the question, while for other questions it is done by *learning* the mapping of the atype clues in the question to atypes (which can be either synsets or surface patterns).

3.1.1 Shallow parsing to extract atype

Shallow parsing, also called *partial parsing* or *chunking* [21, Page 375] involves finding noun phrases, modifiers, and attachments between phrases based purely on part-of-speech (POS) tags and without a deeper understanding of the content. For our purposes, shallow parsing is almost as robust as POS tagging, because we need only local attachments between verbs and noun phrase (NP) chunks.

The parse tree that results from a shallow parse of the question *What is the capital of Japan?* is

```
(S (NP what)
     (S (VP is
            (NP (NP the capital)
                (PP of
                    (NP Japan))))))
```

Here *capital* is the "head" (the most important part) of the NP that is the sibling of the auxiliary verb *is*, and is the atype clue we need. The parse tree of another example from TREC 2002, *What American general is buried in Salzburg?*, is

```
(S (NP what American general)
     (VP is
         (VP buried
             (PP in
                 (NP Salzburg)))))
```

The atype word is *general*, it has an attribute *American*, and these are in the NP before the auxiliary verb *is*.

A study of English questions reveals that the atype words are embedded in the noun phrases in the neighborhood of either the auxiliary (e.g., *have* in "I have completed the homework" or *will* in "He *will* go to London") or the main verb (e.g., *completed* and *go* in these sentences).

This suggests a strategy for locating the atype clues from "what" and "which" questions:

1. If the head of the NP appearing *before* the auxiliary or main verb is not a wh-word, mark this word as an atype clue.
2. Otherwise, the head of the NP appearing *after* the auxiliary/main verb is an atype clue.

[6] *How* followed by a verb or infinitive asks about a *procedure* and is outside the scope of factoid QA, as are *why* questions.

Some atype clue words like *name*, *type* and *kind* are of little use and are discarded (similar to stopwords in IR). In §4.2.1 we will show that this tiny rule set is surprisingly effective.

Dealing with question syntax is the only place where we have to pay some attention to what language we are using, and its grammar rules (at a very superficial level). Porting our system to another language will require a POS tagger and partial parser; we see this as inevitable in analyzing a natural language question.

Training: Given a QA pair (q, a)

1. Connect tokens in a to one or more atype/s
2. Walk from these atypes up hypernym edges (see §2.1), collecting ancestor synsets in a set S_a
3. Locate the wh-word in the question
4. Walk up to k tokens right of the wh-word
5. Build a prefix tree of depth up to k using these tokens
6. In each node of the prefix tree, maintain a map from synsets to counters
7. Find the path in the prefix tree corresponding to the k-prefix of question q starting at the wh-word
8. For each node on the prefix path increment counters for all synsets in S_a

Significance tests:

1. Collect a background distribution of synset frequencies from the corpus
2. For each node n in the prefix tree and for each synset s at n, test the hypothesis that s appears more often at n than in the background distribution
3. Retain only those synsets that survive the test

Deployment:

1. Given a test question, locate its wh-word and walk to the right, trying to find the longest possible matched path in the prefix tree
2. At the last node to be matched, collect the surviving synsets

Figure 3: Generating mappings from wh-words to atypes.

3.1.2 Learning to map atype clues to atypes

Unlike *what* and *which* questions, *when*, *where*, *who* and *how X* questions do not directly use a term that describes a synset. E.g., *when* is only an adverb, meaning "as soon as", whereas we would like to establish a connection to the synset described as *time period*, among others. Moreover, WordNet does not know that 1947 is likely to be a year in a given context, and an augmented "synset" based on surface pattern (`DDDD`) may come handy.

Therefore there is a need to map (note the dotted line in Figure 1) atype clues in questions to a general atype, expressed either as a synset or as a surface pattern. In a significant departure from much of earlier work, we wish to discover and maintain this map automatically via learning techniques.

We devised another learning module to help us compile a table of mappings between short token sequences in questions to atypes. This module is also trained from QA pairs, and is described in Figure 3.

3.2 Identifying selectors

Determining selectors is a fundamental aspect of translating our information needs into queries suited for search engines. Millions of search engine users do this every day. A separate study of Web query log sessions that we conducted showed that a typical user is fairly sure that some words *must* appear on an answer page, but explores a larger neighborhood of other words in a more tentative way. The former are selectors, the latter are not. Our goal is to mimic this process to be able to generate a keyword query from a question.

3.2.1 Choice of features

It stands to reason that our judgment on whether a word is a selector depends on various features we observe about the word, both in the context of the specific question and in our prior knowledge about the language being used.

Features of a candidate word that are local to the question may include

- The part-of-speech (POS) assigned to a word
- The POS assigned to left and right neighbors of the word, up to some window width
- Whether the word starts with an uppercase letter

whereas global features may include

- Whether the word is a known stopword like *an*, *is* etc.
- The fraction of documents which contain the word (some version of IDF)
- How many senses the word has in isolation (available from WordNet)
- For a given sense described by this word, how many other words describe the sense

Words vary widely w.r.t. the number of senses, so for simplicity we reduce the last feature to one number by finding the *average* number of other words describing the various senses of the given word.

We call the last two items *ambiguity indicators*: they indicate the extent of aliasing of the candidate word with other words. Among other things, it is of interest to see how ambiguity indicators derived from knowledge bases like WordNet affect our ability to spot selectors.

Whether a question word has been flagged as an atype clue by the atype extractor (§3.1) is another feature that can be used. Being marked as an atype clue often, but not always, means that a word is likely not a selector.

3.2.2 Choice of learners

Once we put all the attributes discussed above into a feature vector, we can use a variety of learners. The hypothesis space is not very simple. There are discrete and continuous numeric attributes, and, based on world knowledge about languages, we expect the numeric attributes (ambiguity indicators) to follow different distributions conditioned on some of the discrete attributes (e.g. POS).

We were therefore predisposed to believe that a decision tree will work better than learners with simpler hypotheses, such as Logistic Regression. As we discuss in §4.3, this intuition turned out to be true. We compared two decision tree classifiers (C4.5 and J48) and a Logistic Regression classifier packaged in the WEKA data mining toolkit, and picked J48 as the consistent winner.

3.2.3 How to use selectors

Selectors can be used in two places: to pad the initial keyword query, and to rerank the candidate passages. A keyword search engine like Lucene has OR-semantics by default (i.e., passages matching at least one query word is eligible for scoring) but encourages AND-semantics (i.e., passages which match all/most query words get priority). In our experiments we insist that each response contains all selectors, and use Lucene's OR over other question words. Selectors can also be used in the reranking phase: one of the passage features we derive is the average distance from a candidate answer zone (see Figure 1) to a selector.

3.3 Learning to score passages

The algorithms described so far enable us to analyze the question q, probe the keyword index and retrieve a set R_q of promising passages; none, one, or at most a few of the passages in R_q contain the answer to question q. The ranking imposed by an IR engine on these passages is not the best possible for QA. Our noisy simulation model has set the stage for **reranking** the passages, based on features which appear in each candidate passage *in conjunction with* features in the question. The scenario is easily framed as a supervised learning or classification problem. During training, each (q, r) pair, for $r \in R_q$, forms an **instance**. The label is $+1$ if r contains an answer to q and -1 otherwise. During testing and deployment, the learner is only given (q, r) and must guess the label.

An important issue is the encoding of *features* from (q, r) which will best assist the reranking learner. Guided by work in information extraction and chunking, our inclination is to include any and all features which look useful to us, and let a sufficiently powerful learner deal with possible redundancy and correlation.

If (q, r) is a positive instance, we expect that:

- All selectors match between q and r.
- r has an answer zone a which does not contain selectors.
- The linear distances, in terms of number of tokens, between a and matched selectors in r, tend to be small.
- a has strong WordNet-based similarity with the atype of q (see §3.3.1).
- There is some relationship between the atype and a's part of speech tag and any entity tag (person, place, amount, etc.)

We will now design a route which turns (q, r) into a feature vector suitable for a conventional supervised learner. First we need a brief detour to design the WordNet-based similarity between the atype of q and a candidate answer zone a.

3.3.1 Strength of WordNet-based similarity

Let t be the atype of the question, and a a candidate answer zone. Let us overload t to mean the synset (original or augmented) in WordNet corresponding to t, and likewise with a. The simplest notion of the strength of a WordNet-based connection between t and a, as motivated in §2, is whether a is a hyponym descendant of t: a 0/1 feature.

We found some benefit in refining this boolean feature to a continuous feature to encode some idea of the significance of the connecting path. Prior work [20] suggests that simple graph measures, like the number of links, will not suffice. As one example, a three-hop path from *entity* (the most general noun) to *artifact* is not as strong a connection as the three-hop path between *mammal* and *elephant*.

This intuition is captured nicely by measuring the overlap of nodes on the path from each of t and a to all the noun roots. More specifically, we define the WordNet-based similarity feature (which we call **HyperPath**) between t and a as follows:

1. Unless t is a hypernym ancestor of a, the feature value is zero.
2. Otherwise, we collect the sets of hypernym synsets of t and a; let us call them H_t and H_a.
3. Compute the *Jaccard overlap* $|H_t \cap H_a|/|H_t \cup H_a|$.

We argue that (like POS tagging and shallow chunking in earlier sections), a lexical network with measure of specialization is an essential support structure for QA, and the experience generalizes easily to a WordNet-like network designed for a different language.

3.3.2 Feature design

We now complete the discussion of the conversion of a (q, r) instance into a feature vector through a short pseudocode.

1. Remove from consideration all terms in r that are matched selectors of q.
2. Map q to its atype t.
3. Consider each word a (which could be a compound token like *San Francisco*) in r as a potential answer zone.
4. Find the HyperPath similarity between t and a.
5. Retain the winning candidate answer zone a^* with the largest HyperPath score.
6. Collect a set L of linear distances from a^* to each matched selector.
7. Create a feature vector with these components:
   - The original IR rank of passage r
   - The HyperPath similarity between a and t
   - The maximum, average, and minimum of L
   - POS tag of a^*
   - Entity tag (person, place, date, etc.) of a^* if any
   - Wh-words (including compound wh-words like *how many*) from q

3.3.3 Choice of learner

During training, labels are directly available from QA-pairs. During testing, a suitable classifier can assign hard ± 1 labels to each $r \in R_q$, but, as it turns out, a hard elimination leads to an unacceptable level of false negatives. Instead, we used logistic regression (LR) for reranking. LR is a discriminative classifier which gives a continuous estimate between 0 and 1 of the probability that the label is $+1$. We rerank passages in order or their LR scores. Reranking does not rule out paying attention to the keyword search engine's ranking—that could be encoded as a feature for LR. Unlike naive Bayes, LR is good at handling features that are strongly correlated. Because we need to deal with a large number of training instances (20000 is common in production runs), we avoided SVM-based methods.

4. EXPERIMENTS

We experimented with a few years of TREC QA data to assess the merit of our model and approach. In this section we present data to argue that

- Atypes and selectors can be extracted from questions effectively
- Reranking passages by applying a trained learner significantly boosts the quality of ranking
- The benefits of learning to extract atypes and selectors and to rerank generalize from one corpus to another ("inductive transfer" in machine learning jargon)

4.1 Data preparation

We used data from the TREC QA track because it offers not only a standard corpus and a set of question, but also answer passages for each question. TREC QA made a major change in the corpus between 2001 and 2002. To study if our observations and approaches are sufficiently general, we initially focused on two years: 2000 and 2002.

Following Tellex *et al.*, [26] we picked sliding windows of three sentences as passages. For 2000, we got 978953 documents, 14721729 passages, and 693 questions. For 2002, we got 1031824 documents, 23266499 passages, and 500 questions. The passages were indexed by Lucene [5].

For each question, TREC provides a set of document identifiers which answer it, a regular expression which the participant has to match to score, and sometimes, a snippet from the document that contains the answer. From these, it was reasonably simple to assemble QA pairs (where the answer was marked in context).

Questions and passages were tokenized using GATE [10]. GATE also includes a part-of-speech (POS) tagger which tags each token with one of about 36 standard POS tags used in the PENN treebank corpus [21, Chapter 3] (also see `http://www.scs.leeds.ac.uk/amalgam/tagsets/brown.html`).

Atype clues were extracted from a shallow parse of the question by the Link Parser [25], as described in §3.1.1. For the learning tasks, we used the J48 decision tree and the logistic regression packages in WEKA [27].

Our code is entirely in Java, except for call-outs to the Link Parser. For each year of TREC data, our total data preparation and training time on a dual 1.3GHz Xeon server was several hours, most of it spent in tagging top passages returned by Lucene. WEKA itself took only several minutes. Typically, each query could be answered in under 30 seconds.

4.2 From questions to atypes

The question is first analyzed to find the atype we would expect to see in an answer passage. For *Which* and *What* queries, we attempt to extract the atype directly from a shallow parse (§3.1.1). Otherwise, we take the wh-words and phrases from the question and try mapping them to atypes (§3.1.2). Here we show that both these processes perform well.

4.2.1 Extracting atypes from shallow parses

Our algorithm (§3.1) to determine atype clues from questions is surprisingly effective. Manual inspection of hundreds of queries from TREC 2000 and TREC 2002 showed that we extracted correct atype clues in a very large fraction of cases. Figure 4 shows a break-up of our atype-extraction statistics for different question types (starting with What, Which, and Name).

4.2.2 Mapping question prefixes to atypes

As described in Figure 3 (§3.1.2), we grow a prefix tree with a virtual root and the wh-words as the children of the

Question type	Total questions	# correct
what	630	612
which	29	28
name	23	20

Figure 4: Atype statistics: we can correctly extract atype clues from a vast majority of questions of various types.

root. Recall that each node accumulates occurrence counts of atypes (which are WordNet synsets or surface pattern IDs) associated with the answer.

We ran the χ^2 test, the likelihood ratio test and the t-test (all with a rejection level of 0.0005) to identify statistically significant association of an atype to a prefix tree node.

Figure 5 shows some of the most significant atypes (only WordNet synsets in this specific run) that survive the test. The survivors are very intuitive, and change dramatically from node to node in the prefix tree. *How*, a very unspecific atype clue by itself, gets mapped to a very vague synset: the 6^{th} sense of the noun *abstraction*. But survivors at nodes corresponding to *how much* or *how many* are far more specific and useful.

```
How   => {abstraction#n#6}
  fast => {magnitude_relation#n#1,rate#n#2}
  much => {fundamental_quantity#n#1, time_period#n#1}
  many => {measure#n#3, definite_quantity#n#1}
What  => {region#n#3, entity#n#1,living_thing#n#1,
          casual_agent#n#1}
  city        => {area#n#1}
  achievement => {physical_phenomenon#n#1}
  university  => {geographical_area#n#1}
Where => {region#n#3}
Who   => {person#n#1}
When  => {calendar_month#n#1, happening#n#1,
          accident#n#2}
```

Figure 5: Sample frequent connections from atype clues to WordNet synsets. The WordNet synsets as shown are given in the form `synset#part-of-speech#sense`.

If we fail to extract an atype from the parse and we fail to match a new question to a precompiled prefix, we can always fall back on keyword-based ranking. Summarizing, at the end of this stage we are prepared with a set of synsets which we expect to connect to a suitable *hyponym* in the answer zone of an answer-bearing passage.

4.3 Spotting selectors in questions

We compared the J48 decision tree classifier in WEKA with the Logistic Regression classifier in WEKA. The results are shown in Figure 6. These numbers include all features discussed in §3.2 except whether the word has been flagged as an atype clue (which is negative evidence for being also a selector). The proximity window was chosen as ±1 and ±2 tokens, and the results were essentially the same. A window size of zero, i.e., not considering neighboring words, was slightly worse. As we expected, the decision tree is better than regression, and this is statistically significant. The higher precision of J48 is important to avoid eliminating passages early in our overall algorithm.

Classifier	Recall	Precision	F1	%correct
Logistic	0.79	0.71	0.75	74.5
J48	0.84	0.78	0.81	80.5

Figure 6: J48 (decision tree) is better than Logistic Regression at identifying selectors in the query.

In a different experiment, we compared the effect of including the atype clue flag as a feature. In that experiment, we did not use the ambiguity indicators from WordNet. Using the atype flag increased accuracy modestly, from 76% to 79% for TREC 2000 and from 75% to 76.3% for TREC 2002. On the other hand, adding ambiguity features was very beneficial: typically, it raised selector classification accuracy from 71-73% to over 80%, and this was again statistically significant. Figure 7 shows some snippets of the decision tree built for selectors, which are very intuitive in retrospect. Summarizing, we can flag selectors with about 75–80% accuracy.

```
POS@0=adj
  POS@-1=noun
    NumSense@0 <= 9
      NumLemma@0 <= 2.5: selector
      NumLemma@0 >  2.5: not-selector
    NumSense@0 >  9: not-selector
...
POS@0=verb
  NumLemma@0 <= 1.82
    POS@-1=noun
      POS@+1=noun: selector
    ...
  NumLemma@0 >  1.82: not-selector
```

Figure 7: The upper levels of a decision tree induced on the selector-learning data shows intuitive rules exploiting both local POS and ambiguity information. `NumSense` is the number of senses of the test word and `NumLemma` is the number of other words sharing a sense with the test word.

4.4 Passage reranking performance

Once we set up a Logistic Regression (LR) classifier for the reranking step, a natural first measurement to make is its accuracy. Sampling training instances and validating on held-out labeled data show that the classifier learns quickly, settling near its best accuracy within only 2–4% of the training data available.

TREC 2000

	−1	+1
−1	51228	308
+1	5017	3359

$P = 0.918 \pm 0.013$

$R = 0.4 \pm 0.004$

$F_1 = 0.56 \pm 0.002$

TREC 2002

	−1	+1
−1	38396	51
+1	2169	550

$P = 0.934 \pm 0.023$

$R = 0.2 \pm 0.0038$

$F_1 = 0.33 \pm 0.004$

Figure 8: The reranking classifier can reject negative (non-answer) passages well, but suffers from low recall owing partly to the large number of training instances labeled −1. Label +1 means the passage has an answer, −1 means it does not. We show representative confusion matrices and mean and standard deviation of recall, precision and F_1 scores over five runs, each using 40% of the labeled QA instances sampled u.a.r. as training data.

Figure 8 shows that accuracy (the fraction of instances classified correctly) is high (over 80%) but this is largely because of the large fraction of negative instances. Recall (R), precision (P), and $F_1 = 2RP/(R+P)$ are more modest. The large negative training bias makes recall low, risking false negatives if we use the LR classifier to simply eliminate candidate passages. Luckily, in our application we need not make a hard decision[7]; we simply sort by the score returned by Logistic Regression.

Reranking greatly improves the rank of the correct passages. We show in Figure 9 a histogram of the number of

[7]This is not quite true; a very small number of TREC questions test if the system can return an empty set. We can do a recall/precision trade-off and cross-validate in that case.

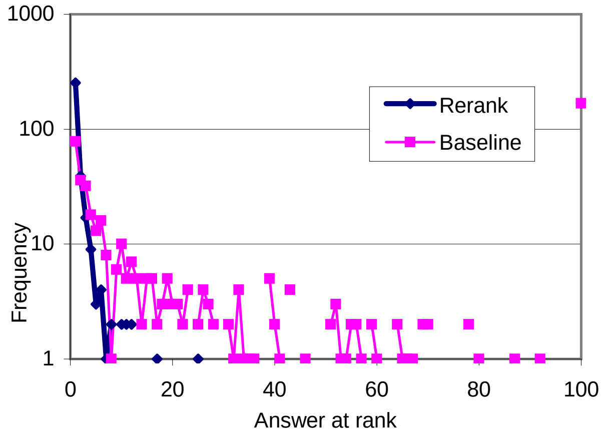

Figure 9: Reranking significantly improves the rank of correct passages. The x-axis is the rank at which the (first) answer is found, the y-axis counts questions for which the first answer was found at the given rank. Note that the y-axis is log-scale, and reranking increases precision at rank 1 from 78 to 253.

answer passages found at specific ranks by a baseline system (before reranking) and after passage reranking. It is immediately obvious that reranking eliminates many non-answers, pushing answers to better ranks.

In the QA literature, rank histograms are aggregated into a single figure of merit: the **mean reciprocal rank (MRR)**. Suppose $n_q \geq 1$ is the earliest rank of the passage at which the answer to question $q \in Q$ is found. Then MRR is defined as $(1/|Q|) \sum_q (1/n_q)$. MRR is between 0 and 1, and a higher MRR is better.

Passage ranked by	TREC 2000	TREC 2002
IR score (Lucene)	0.377	0.249
LR score	0.71 ± 0.001	0.565 ± 0.001

Figure 10: Mean and standard deviation of MRRs after reranking by logistic regression (LR), calculated over five runs, each using a random 40% sample of labeled data for training. MRR based on LR reranking exceeds baseline MRR based on IR ranking by 86% and 127%.

Our MRR scores are shown in Figure 10, For comparison, some of the top MRRs from the actual TREC 2000 competition are shown in Figure 11. Our TREC 2000 score compares favorably. (In later years, TREC replaced MRR with a direct assessment of the pinpointed answer zone, which is beyond our scope.)

System	Corpus	MRR
FALCON	TREC 2000	**0.76**
U. Waterloo	TREC 2000	0.46
Queens College, CUNY	TREC 2000	0.46
Webclopedia	TREC 2000	0.31

Figure 11: Recent best MRRs at TREC, showing that our system compares favorably.

Does reranking benefit all kinds of queries equally? To study this, we divided queries into categories based on their starting words (*what*, *which*, *how*..., which are good indicators of question type), and plotted their group MRRs before and after reranking in Figure 12. MRR improvement via reranking is widespread across question types, but is particularly high for *what* and *which* queries.

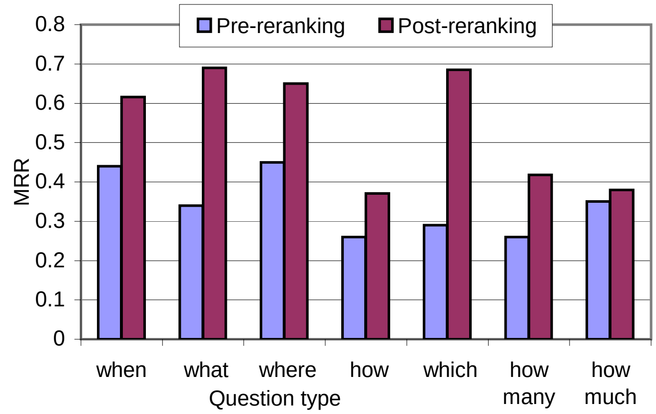

Figure 12: Sample MRR improvement via reranking separated into question categories.

Finally, we wanted to verify that our learning approach picks up generic patterns useful for QA, not patterns specific to a particular corpus and QA set. TREC has published QA pairs over several years, and the corpus changed completely starting 2002. This gave us a good opportunity to test the generalization power of our system.

Trained on	Tested on	Reranked MRR
2002	2002	**0.565**
2000	2002	0.539
2000+2001	2002	0.534
2000+2001+1999	2002	0.541
2000	2000	**0.710**
2002	2000	0.705
2002+2001	2000	0.627
2002+2001+1999	2000	0.693

Figure 13: MRR improvements if we train on one TREC year and test on another.

The basic idea was to collect QA instances from some years set apart as "training years" and test it on a fixed "testing year." The results are shown in Figure 13. The MRR improvement, while smaller than with cross-validated reranking within each year (as discussed before), are still quite significant. It is also clear that very little training data is sufficient, and in fact, some protection against overfitting to patterns in specific years would help.

5. CONCLUSION

We have presented a QA system that is built by wrapping a small and simple logic around text indexers, taggers, shallow parsers, and classifiers. The system is trained with QA-pairs. A new model of QA match and proximity is used to make the system trainable, transparent and modular. Experiments with past TREC QA data show that our technique is effective. Our study in this paper clearly exposes what factors matter, and by how much, rather than report overall system performance.

Our claim is not that our building blocks (POS and entity tagger, indexer, shallow parser, and classifier) are simple or free of expert tuning. Our central claim is that *the way we* *assemble them together is simple* (only about 4000 lines of Java code), and that the resulting assembly can be reproduced easily by other researchers and trained mechanically by QA pairs by people having no QA expertise.

In ongoing work, we are exploring how to get away from our single-shot selector model using backoff queries on Lucene, and cautious forms of query expansion, again, guided by QA-mining alone. The selector learner can perhaps be improved by exploiting word usage statistics from WordNet and the corpus. Like Dumais and others [8, 11], we would like to include redundancy into our scoring model. Performance improvement in our graph reachability algorithms will also let us consider more candidate passages within a limited response time.

Acknowledgments: This research was supported in part by TCS, IBM, and NEC.

6. REFERENCES

[1] S. Abney, M. Collins, and A. Singhal. Answer extraction. In *Applied Natural Language Processing*, pages 296–301, 2000.

[2] E. Agichtein, S. Lawrence, and L. Gravano. Learning search engine specific query transformations for question answering. In *Proceedings of the 10th World Wide Web Conference (WWW10)*, pages 169–178, 2001.

[3] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, San Jose, CA, 2002. IEEE.

[4] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Journal of Language Engineering*, 1(1):29–81, 1995.

[5] Apache Software Group. Jakarta Lucene text search engine. GPL Library, 2002.

[6] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, San Jose, CA, 2002. IEEE.

[7] C. Clarke, G. Cormack, D. Kisman, and T. Lynam. Question answering by passage selection (MultiText experiments for TREC9). In E. Voorhees and D. Herman, editors, *The Ninth Text REtrieval Conference (TREC 9)*, NIST Special Publication 500-249, pages 673–683. NIST, 2000.

[8] C. L. A. Clarke, G. V. Cormack, and T. R. Lynam. Exploiting redundancy in question answering. In *Research and Development in Information Retrieval*, pages 358–365, 2001.

[9] C. L. A. Clarke and E. L. Terra. Passage retrieval vs. document retrieval for factoid question answering. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 427–428. ACM Press, 2003.

[10] H. Cunningham, K. Humphreys, R. Gaizauskas, and Y. Wilks. GATE: A TIPSTER-based general architecture for text engineering. In *Proceedings of the TIPSTER Text Program (Phase III) 6 Month Workshop*. Morgan-Kaufmann, may 1997.

[11] S. Dumais, M. Banko, E. Brill, J. Lin, and A. Ng. Web question answering: Is more always better? In *SIGIR*, pages 291–298, aug 2002.

[12] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll. In *WWW*, New York, may 2004. ACM.

[13] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. In *VLDB*, volume 24, pages 26–37, New York, Sept. 1998. Online at `http://www-db.stanford.edu/pub/papers/proximity-vldb98.ps`.

[14] S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus, and P. Morarescu. FALCON: Boosting knowledge for answer engines. In *Proc. of Ninth Text REtrieval Conference (TREC 9)*, pages 479–488, 2000.

[15] E. . Hovy, L. Gerber, U. Hermjakob, M. Junk, and C.-Y. Lin. Question answering in webclopedia. In *Proceedings of the TREC-9 Conference. NIST, Gaithersburg, MD*, 2000.

[16] E. H. Hovy, U. Hermjakob, C.-Y. Lin, and D. Ravichandran. Using knowledge to facilitate pinpointing of factoid answers. In *COLING*, Taipei, aug 2002.

[17] V. Hristidis, Y. Papakonstantinon, and A. Balmin. Keyword proximity search on xml graphs. In *Proceedings of 18th International Conference on Data Engineering, San Jose, USA*, 2002.

[18] C. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the Web. In *WWW*, volume 10, pages 150–161, Hong Kong, may 2001. IW3C2 and ACM. See `http://www10.org/cdrom/papers/120/`.

[19] S. K. Lam, D. M. Pennock, D. Cosley, and S. Lawrence. 1 billion pages = 1 million dollars? mining the web to play "who wants to be a millionaire?". In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence, UAI 2003*, Acapulco, Mexico, August 8–10 2003.

[20] D. Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304, San Francisco, CA, 1998. Morgan Kaufmann.

[21] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

[22] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to WordNet: An online lexical database. *International Journal of Lexicography*, 1993.

[23] A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Intelligent User Interfaces*, pages 149–157, Miami, jan 2003. ACM.

[24] D. R. Radev, H. Qi, Z. Zheng, S. Blair-Goldensohn, Z. Zhang, W. Fan, and J. Prager. Mining the Web for answers to natural language questions. In *Proceeding of the Tenth International Conference on Information and Knowledge Management (CIKM)*. ACM, 2001.

[25] D. D. Sleator and D. Temperley. Parsing English with a link grammar. In *Third International Workshop on Parsing Technologies*, 1993.

[26] S. Tellex, B. Katz, J. Lin, A. Fernandes, and G. Marton. Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 41–47. ACM Press, 2003.

[27] I. H. Witten and E. Frank. *Data mining: Practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, San Francisco, oct 1999.

[28] Z. Zheng. Answerbus question answering system. In *Human Language Technology Conference (HLT)*, 2002.