Learning to Find Answers to Questions on the Web

EUGENE AGICHTEIN
Columbia University
STEVE LAWRENCE
NEC Research Institute
and
LUIS GRAVANO
Columbia University

We introduce a method for learning to find documents on the Web that contain answers to a given natural language question. In our approach, questions are transformed into new queries aimed at maximizing the probability of retrieving answers from existing information retrieval systems. The method involves automatically learning phrase features for classifying questions into different types, automatically generating candidate query transformations from a training set of question/answer pairs, and automatically evaluating the candidate transformations on target information retrieval systems such as real-world general purpose search engines. At run-time, questions are transformed into a set of queries, and reranking is performed on the documents retrieved. We present a prototype search engine, *Tritus*, that applies the method to Web search engines. Blind evaluation on a set of real queries from a Web search engine log shows that the method significantly outperforms the underlying search engines, and outperforms a commercial search engine specializing in question answering. Our methodology cleanly supports combining documents retrieved from different search engines, resulting in additional improvement with a system that combines search results from multiple Web search engines.

Categories and Subject Descriptors: H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*linguistic processing*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*query formulation, retrieval models, search process*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*performance evaluation (efficiency and effectiveness), information networks*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*web-based services*

General Terms: Algorithms, Experimentation, Performance

---

1. INTRODUCTION

Many natural language questions (e.g., *"What is a hard disk?"*) are submitted to search engines on the Web every day, and an increasing number of search services on the Web specifically target natural language questions. For example, AskJeeves (www.ask.com) uses databases of precompiled information, metasearching, and other proprietary methods, while services such as AskMe (www.askme.com) and Google Answers (answers.google.com) facilitate interaction with human experts.

Web search engines such as AltaVista (www.altavista.com) and Google (www.google.com) typically treat natural language questions as lists of terms and retrieve documents similar to the original query. However, documents with the best answers may contain few of the terms from the original query and may be ranked low by the search engine. These queries could be answered more precisely if a search engine recognized them as questions.

Consider the question *"What is a hard disk?"*. The best documents for this query are probably not the company Web sites of disk storage manufacturers, which may be returned by a general-purpose search engine, but rather hardware tutorials or glossary pages with definitions or descriptions of hard disks. A good response might contain an answer such as: *"Hard Disk: One or more rigid magnetic disks rotating about a central axle with associated read/write heads and electronics, used to store data..."*. This definition can be retrieved by transforming the original question into a query {*hard disk* NEAR *"used to"*}. Intuitively, by requiring the phrase *"used to"*, we can bias search engines towards retrieving this answer as one of the top-ranked documents.

We present a new system, *Tritus*, that automatically learns to transform natural language questions into queries expected to retrieve answers to the question using a given search engine (e.g., a specific Web search engine such as Google). An example *Tritus* search for the question *"what is a hard disk?"* is shown in Figure 1. *Tritus* has determined the best 15 transforms for the *"what is a"* type of question (e.g., {*hard disk "is usually"*}, {*hard disk called*}) for the specific underlying search engine (in this case, for Google). *Tritus* learns these effective transformations automatically during training by analyzing a collection of question-answer pairs, and recognizing the indicative *answer* phrases for each question type (e.g., *Tritus* learns that a phrase *"is usually"* is a good transform for *"what is a"* questions). We describe the *Tritus* training process in more detail in Section 3.

At run-time, *Tritus* starts with a natural language question submitted by the user (e.g., "what is a hard disk?"), which is *transformed* into a set of new, effective queries for the search engine of interest. *Tritus* then retrieves and reranks the documents returned by the underlying search engine. In our prototype, *Tritus* returns the documents to the user directly (see Figure 1); alternatively, the documents returned by *Tritus* can be used as input to a traditional question
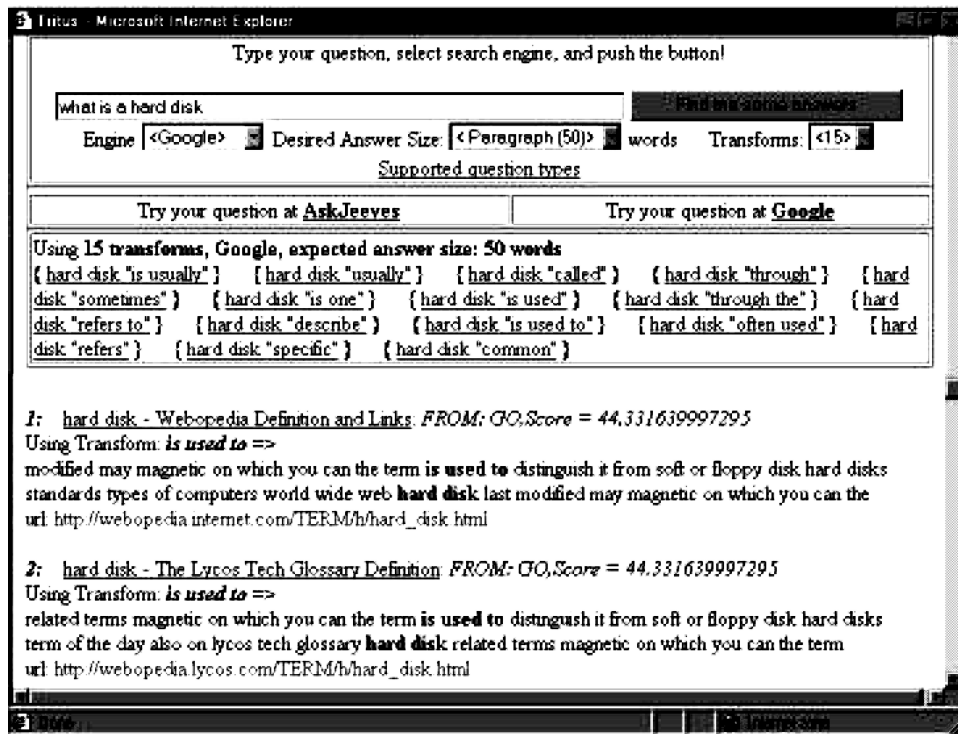
Fig. 1. An example *Tritus* search for the question "What is a hard disk".

answering system in order to extract the actual answers from the retrieved documents.

The rest of the article is organized as follows. We review the related work in the next section. Then we present our method for automatically learning to transform natural language questions into queries containing terms and phrases expected to appear in documents containing answers to the questions (Section 3). As part of our evaluation, we compare the quality of the documents retrieved by *Tritus* with documents retrieved by other state-of-the-art systems (Section 4) in a blind evaluation (Section 5) over a set of questions chosen randomly from the query logs of a public Web search engine.

2. RELATED WORK

Question Answering (QA) has been an area of active research. Recently, the state-of-the-art in QA research has been represented in the Text Retrieval Evaluation Conference (TREC) question answering track evaluations [Voorhees 2001], which involve retrieving a short (50 bytes long) answer to a set of test questions. In our work we address an aspect of question answering that was not a direct focus of the TREC QA track. We also consider a more general class of questions, where the answers may not be short, precise facts, and the user might be interested in multiple answers (e.g., consider the question "What are ways people can be motivated?").

More specifically, we focus on the first part of the question answering task, namely, retrieving promising documents that are likely to contain an answer to a given question. Retrieving documents to match an information need has long been an active topic of information retrieval (IR) research. The body of the IR research most closely related to our work focuses on automatically expanding queries in order to retrieve additional relevant documents (e.g., Rocchio [1971] and Mitra et al. [1998]). An interesting approach presented by Xu and Croft [2000] describes how to automatically expand a query based on the co-occurrence of query terms with other terms in the top-ranked documents for the original query. In general, automatic query expansion systems modify queries at run-time on a query-by-query basis, using documents returned in response to the original query. In contrast, we will show how to derive a set of query transformations that work best for each *type* of question.

Once the initial set of documents has been retrieved, typical QA systems attempt to extract the answers from these documents. For example, Abney et al. [2000] describe a system in which answers are extracted from documents returned by the SMART information retrieval system. Questions are classified into "question types" that identify the type of entity that is appropriate for the answer. Documents are tagged to recognize entities, and passages surrounding entities of the correct type for a given question are ranked using a set of heuristics. Moldovan et al. [1999] and Aliod et al. [1998] present systems that rerank and postprocess the results of regular information retrieval systems with the goal of returning the best document passages. Cardie et al. [2000] describe a system that combines statistical and linguistic knowledge for question answering and employs sophisticated linguistic filters to postprocess the retrieved documents and extract the most promising passages to answer a question.

Most of the question answering systems presented in the TREC 8 and 9 QA track evaluations used the general approach of retrieving documents or passages that are similar to the original question with variations of standard TF-IDF term weighting schemes [Salton 1989]. The most promising passages are chosen using heuristics and/or hand-crafted regular expressions. This approach is not optimal because documents that are similar to the *question* are initially retrieved. However, the user is actually looking for documents containing an *answer* and these documents may contain few of the terms used to ask the original question. Using effective queries is particularly important when document retrieval is expensive or limited to a certain number of documents, as is the case with Web search engines.

More recently, some of the traditional QA systems have begun to modify queries to improve the chance of retrieving answers. Harabagiu et al. [2000] describe a system that transforms questions using a hierarchy of question types. The hierarchy is built semi-automatically using a bootstrapping technique. Other systems (e.g., Hovy et al. [2000]) used WordNet [Miller 1995] to expand queries with word synonyms and hypernyms to retrieve additional relevant documents. However, this approach has not been shown to significantly improve the quality of the retrieved document set. Additionally, the AskMSR system [Brill et al. 2001] used manually-crafted, question-to-query transformations to focus on promising documents. In a question answering document

retrieval approach closer to our work, Ittycheriah et al. [2000] used LCA [Xu and Croft 2000], a form of automatic query expansion, to expand the initial query with terms selected from encyclopedia passages matching the initial question.

Recent work has been done on *reranking* the candidate set of documents, or phrases, with the goal of locating those most likely to contain answers to a question. In contrast, we focus on learning how to generate good queries to create a quality candidate set of documents, which can then be postprocessed by such reranking techniques. For example, a machine-learning and term-correlation based approach for reranking candidate answer phrases, or documents, is described in Berger et al. [2000] and is similar in spirit to our work. However, this work does not address learning effective *queries* for specific Web search engines. Similarly, Mann [2002] describes a method for learning to locate an exact answer in a passage by using co-occurrence statistics derived from question-answer pairs used in trivia games. We use a similar training set (FAQ question-answer pairs in our case) for the different purpose of deriving effective queries to retrieve promising documents. Prager et al. [2002] show how to automatically identify the semantic type of the expected answer to a question. All these methods can be used to process the set of documents retrieved by *Tritus* to extract short and precise answers from them.

In a study more closely related to our work, Lawrence and Giles [1998] introduced *Specific Expressive Forms* for Web search, where questions are transformed into specific phrases that may be contained in answers. For example, the question "what is x" may be transformed into phrases such as "x is" or "x refers to". The main difference from our current work is that in Lawrence and Giles [1998], the transforms are handcrafted (hard coded) and the same set of queries is submitted to all search engines used, except for the differences in query syntax between search engines. Joho and Sanderson [2000] use a set of handcrafted query transformations in order to retrieve documents containing descriptive phrases of proper nouns. Schiffman and McKeown [2000] describe experiments in automatically building a lexicon of phrases from a collection of documents with the goal of building an index of the collection that is better suited for question answering.

In the research area of specialized search engines, Glover et al. [2001] present a method for learning topic-specific query modifications. The system starts with a set of example documents on the topic of interest, and trains a classifier to recognize such documents. Then, salient features are selected from the example documents and tested on the search engine of interest by retrieving documents using the candidate queries. Our method, which we will describe in the next section, uses a similar approach for a different setting of question answering.

Recently, Radev et al. [2001] independently presented a method for automatically weighting query reformulation operators (e.g., insertions and deletions of terms in the original question), with the goal of transforming a natural language question into one "best" search engine query. Our approach, methodology, and evaluation are substantially different. For example, while the training of the system in Radev et al. [2001] assumes a small controlled collection of documents, we address the problem of learning effective queries for general purpose search engines over the Web at large. The system for answering questions over

the Web presented by Radev et al. [2002] does not apply any query reformulation strategies, but rather uses the original question as the query. Another recently presented system, *Mulder* [Kwok et al. 2001], also transforms questions into queries and extracts answers from the returned documents. The presented method uses a semi-automatically constructed taxonomy of questions in order to match a new question to a known question type, and to use predefined transformations to convert the question to the query.

In contrast to the previous research in question answering and query expansion, we present a system that *automatically learns* multiple query transformations, *optimized specifically* for each search engine, with the goal of maximizing the probability of an information retrieval system returning documents that contain answers to a given question. We exploit the inherent regularity and power of natural language by transforming *natural language questions* into sets of *effective search engine queries*.

3. THE TRITUS SYSTEM

Submitting natural language questions (e.g., *"How do I tie shoelaces?"*) to search engines in their original form often does not work very well. Search engines typically retrieve documents similar to the original queries. Unfortunately, the documents with the best answers may contain only one or two terms from the original queries. Such useful documents may then be ranked low by the search engine, and will never be examined by typical users who do not look beyond the first page of results. To answer a natural language question, a promising approach is to automatically reformulate the question into a query that contains terms and phrases that are expected to appear in documents containing answers to the original question.

3.1 Problem Statement

We focus on the first step of the question answering process: retrieving a set of documents likely to contain an answer to a given question. These documents are then returned as the output of the system. The returned documents can be examined by a human user directly, or passed on to sophisticated answer extraction modules of a question answering system (e.g., Abney et al. [2000], Mann [2002], Prager et al. [2002], and Radev et al. [2002]). Thus, it is crucial that the answer to a question of interest be present in this set of initially retrieved documents. At the same time, the set of retrieved documents cannot be so large that it overwhelms the user or the subsequent (typically computationally expensive) answer extraction components. Therefore, our goal is to return a reasonable-sized set of documents that, at the same time, must contain an answer to the question. We now formally state the problem that we are addressing.

Problem Statement: We are given a general purpose search engine *SE* (e.g., Google) that operates over a document collection (e.g., the Web), and a natural language question Q_{NL}. Our goal is to retrieve a set of *documents* via *SE* that are likely to contain an answer to Q_{NL}. For this, we *transform* Q_{NL} into a set of *queries*, $q_1, \ldots, q_m$, such that the top K documents returned by *SE* for each of the queries are likely to contain an answer to the original question Q_{NL}.

```
(1)    Generate Question Phrases from Questions in Training Data
          (Section 3.2.1)
(2)    Generate Candidate Transforms from Answers in Training Data
          (Section 3.2.2)
(3)    Evaluate Candidate Transforms for each Search Engine
          (Section 3.2.3)
(4)    Output Best Transforms for each Search Engine
```

Fig. 2.   Outline of the process used to train the *Tritus* system.

In the rest of this section, we describe our solution to this problem. First, we define a strategy for transforming natural language questions into candidate search engine queries (Section 3.2). This strategy relies on a set of question-answer pairs for training, derived from a parsed FAQ collection (which we will describe in detail in Section 4.1). In this section, we also describe our method for automatically detecting the most effective of the candidate transformations for a search engine of interest. Finally, we show how *Tritus* evaluates a question at run-time by applying these transformations (Section 3.3).

3.2 Learning to Transform Questions into Effective Queries

We attempt to find transformations from natural language questions into effective queries that contain terms or phrases expected to appear in documents that contain answers to the question. Our learning process is shown in Figure 2.

3.2.1 *Selecting Question Phrases.*   In the first stage of the learning process (Step (1) in Figure 2), we generate a set of phrases that identify different categories of questions where the questions in each category have a similar goal. For example, the question *"What is a hard disk?"* implies that the user is looking for definitions or descriptions of a hard disk. The goal of the question can be inferred from the *question phrase "what is a"*.[1]

The input to this stage is a set of questions. These questions and their corresponding answers constitute the training data. As we will describe in Section 4.1, we use a collection of FAQ question and answer pairs for this stage of training. We generate potential question phrases by computing the frequency of all n-grams (phrases) of length *minQtokens* to *maxQtokens* words, with all n-grams anchored at the beginning of the questions. We use all resulting n-grams that occur at least *minQPhrCount* times.

The output of this stage is a set of question phrases that can be used to quickly classify questions into respective question types. Sample question phrases, automatically generated from questions in the training collection described later, are shown in Table I.

This method for selecting question phrases can produce many phrases, potentially including a significant number of phrases that are too specific to be widely applicable. Because the following stages of the training process are relatively expensive and we have limited resources for training, we chose to limit

---

[1]We use the term *phrase* to refer to two or more consecutive words, separated by space, and not in the technical linguistic sense.

Table I. Question Type Phrases Used for Evaluation
(Section 4)

Question Type	Question Phrase(s)
Who	"who was", "who is"
How	"how do i", "how can i"
Where	"where is", "where can i"
What	"what are", "what is", "what is a"

```
(^what (is|are)\s)|(^who (is|was)\s)
|(^how (do|can)\s)|(^where (is|can)\s)
```

Fig. 3. The regular expression used to filter the automatically-generated question phrases that are candidates for transformation.

the training for the results reported here to phrases that match the regular expressions shown in Figure 3. The regular expressions match common questions and allow us to concentrate our resources on the most useful phrases. The actual phrases generated as a result of this step are the frequent *superstrings* that match these regular expressions. For example, some generated phrases that match the prefix "*^what is*" are *"what is a"*, *"what is the"*, and so on. Feature selection techniques, part-of-speech tagging, and other natural language processing techniques may be used to fine-tune the filtering of generated question phrases. Note that we do not attempt to predict the type of the expected *answer* to the question (which is one of the main uses of question classification in traditional QA systems). In contrast, we will only use the question classification to select the set of automatically learned queries into which we will transform the question.

Although alternative approaches can be used to identify categories of questions, our *n*-gram approach has a number of advantages. This approach is relatively inexpensive computationally, allowing the processing of large training sets. The approach is also domain independent and will work for many languages with only minor modifications. Additionally, when evaluating a question at run-time (Section 3.3), categorizing a question using phrase matching can be incorporated with negligible overhead in the overall processing time of queries.

3.2.2 *Generating and Filtering Candidate Transforms.* In the second stage of the learning algorithm (Step (2) in Figure 2), we generate candidate terms and phrases that may be useful for reformulating questions. We apply a filtering procedure to reduce the computational requirements for the following stage (evaluating the *candidate transforms* for search engine effectiveness, Step (3) in Figure 2). Candidate transforms are generated for each of the *question phrases* from the previous learning stage. The procedure for generating candidate transforms for each question phrase *QP* consists of a number of steps, namely, generating initial candidate transform phrases, filtering these phrases by minimum co-occurrence count, and weighting and further filtering the remaining phrases. Each step is described below in detail.

For this stage of the learning process, we use a collection of <*Question, Answer*> pairs. Each pair is also assigned a *FAQCategory*, which corresponds to

Question	Answer
What is a Lisp Machine (LISPM)?	A Lisp machine (or LISPM) is a computer which has been optimized to run lisp efficiently and provide a good environment for programming in it. . . .
What is a near-field monitor?	A near field monitor is one that is designed to be listened to in the near field. Simple, eh? The "near field" of a loudspeaker is the area where the direct, unreflected sound from the speaker dominates significantly over the indirect and reflected sound, sound bouncing off walls, floors, ceilings, the console. . . .

Fig. 4.   Sample question-answer pairs from the training collection.

Table II. A Sample of Candidate Transforms
Generated Without Disregarding Phrases
Containing a Noun

Question Phrase	Candidate Transforms
"what is a"	"the term" "component" "ans" "a computer" "telephone" "collection of" "stands for" "unit"

the category of the original newsgroup from which this pair was extracted (e.g., `comp.lang.C`). A sample of the original collection is given in Figure 4. As we will describe next, this stage of the learning process operates over a collection that has been *tagged* with a part-of-speech tagger, which assigns a syntactic part of speech (e.g., noun, verb) to each word in the text. We use Brill's part-of-speech tagger [Brill 1992], which is widely used in the natural language processing community.[2]

For each *<Question, Answer>* pair in the training collection where a prefix of *Question* matches *QP*, we generate all possible potential answer phrases from all of the words in the prefix of *Answer*. For this we use *n*-grams of length *minAtokens* to *maxAtokens* words, starting at every word boundary in the first *maxLen* bytes of the *Answer* text. A sample of answer phrases generated after this step is shown in Table II. These phrases are heavily biased towards electronics or the computer domain. They were generated because a large portion of the documents in the training collection were on technology-related topics. If we used these phrases in transforms, we might change the intended topic of a query. Recall that the transformations that we are trying to learn should improve accuracy of the retrieved set of documents, yet preserve the topic of the original query.[3]

---

[2]Available at `http://www.cs.jhu.edu/~brill/`.

[3]By *topic* we mean a set of concepts that describe the user's information need—roughly related to the *answer focus* of the question answering community.

We address this problem by using a simple heuristic that worked well in our training experiments. We filter out candidate transform phrases containing nouns, because we observed that in most of the queries the nouns are *content* words (i.e., words expressing the topic of the query). Since the transformations will ultimately be applied to questions, our goal is to maximize the query effectiveness while preserving the topic of the original query. We found that if transformations including nouns are used, the resulting query is more likely to match documents not relevant for answering the original question. For example, consider a real question found in one of the search engine logs that we have examined, *"what is a rainbow?"* When we automatically computed a set of candidate transformations for the "what is a" question phrase (Table II) without using our heuristic, one of the high-ranking candidate transformations is the single word *"telephone"*. Were we to use "telephone" as a transformation for the original question (resulting in the query {*rainbow* AND *telephone*}), the documents matching this query will likely not contain an answer to this question. Thus, we filter candidate transform phrases by checking if a generated answer phrase contains a noun, and if it does, the phrase is discarded. There is some empirical support for our decision to use this heuristic. Klavans and Kan [1998] show that verbs are particularly useful for text filtering—namely, that they can provide important clues to selecting relevant documents. By allowing verbs, adverbs, and adjectives in the transformations and discarding nouns, the heuristic attempts to utilize this notion, while preserving the topic of the original question.

Additionally we require that candidate transform phrases have some minimum support, *catSupport*, defined as the number of distinct FAQ categories in which the answer phrase should appear. Filtering by FAQ category allows us to avoid phrases that are too specific. If we were interested in domain-specific question answering, we could use the originating categories to restrict transformations to ones that help questions on a particular topic. In contrast, our goal is to generate queries that are *generally* useful and can be applied to questions in any domain.

Of the remaining *n*-grams, we keep the top *maxPhrCount* most-frequent answer phrases that co-occur at least *minAPhrCount* times with the corresponding question phrase. We then apply IR techniques for *term weighting* to rank these candidate transforms. The initial term weights are assigned to each candidate transform phrase, t_i, by applying the term weighting scheme described in Robertson and Walker [1997]. These term weights were used in the Okapi BM25 document ranking formula (used by the state-of-the-art Okapi information retrieval system, participating in TREC conferences since TREC-3). Many information retrieval systems use the *vector space* model [Salton 1989] to compute similarity between documents, where similarity is computed as a dot product between vectors representing each document. The elements of each vector are calculated as a combination of the *term weight* and *term frequency* of each term in the document. The BM25 metric [Robertson et al. 1998] uses a similar idea. In the original definition of BM25, each term t_i in the document is assigned the Robertson-Sparck Jones term weight $w_i^{(1)}$ [Robertson and

Table III. Sample Candidate Transforms Along with Their Frequency
Count qtf_i, BM25 Term Weight $w_i^{(1)}$, and the Resulting Term Selection
Weight wtr_i

Question Phrase	Candidate Transform	qtf_i	$w_i^{(1)}$	wtr_i
	"refers to"	30	2.71	81.3
	"refers"	30	2.67	80.1
	"meets"	12	3.21	38.52
"what is a"	"driven"	14	2.72	38.08
	"named after"	10	3.63	36.3
	"often used"	12	3.00	36
	"to describe"	13	2.70	35.1

Sparck-Jones 1976] with respect to a specific *query topic* and is calculated as:

$$w_i^{(1)} = \log \frac{(r + 0.5)/(R - r + 0.5)}{(n - r + 0.5)/(N - n - R + r + 0.5)}, \tag{1}$$

where r is the number of relevant documents containing t_i, N is the number of
documents in the collection, R is the number of relevant documents, and n is
the number of documents containing t_i. Intuitively, this weight is designed to
be high for terms that tend to occur in many relevant documents and few non-
relevant documents, and is smoothed and normalized to account for potential
sparseness of relevance information in the training data.

In the original definition of BM25, term weight $w_i^{(1)}$ is specific to each query
topic. We apply this metric to our task of weighting candidate transforms by
incorporating two modifications. First, we interpret *query topic* as question type.
In this interpretation, a relevant document is one of the answers in the training
collection that corresponds to the question phrase (question type). Therefore
$w_i^{(1)}$ is an estimate of the selectivity of a candidate transform t_i with respect
to the specific question type. Second, we extend the term weighting scheme
to *phrases*. We apply the same consistent weighting scheme to phrases, and
treat them in the same way as we treat the single word terms. We compute this
weight for each candidate transform tr_i by computing the count of <*Question,
Answer*> pairs where tr_i appears in the *Answer* to a question matching *QP* as
the number of relevant documents, and consider the number of the remaining
<*Question, Answer*> pairs where tr_i appears in the *Answer* as nonrelevant, and
apply the formula in Equation (1).

We then compute the *term selection weights*, wtr_i, for each candidate trans-
form tr_i, as described in Robertson [1990] in the context of selecting terms for
automatic query expansion as:

$$wtr_i = qtf_i \cdot w_i^{(1)}, \tag{2}$$

where qtf_i is the co-occurrence count of tr_i with *QP*, and $w_i^{(1)}$ is the relevance-
based *term weight* of tr_i computed with respect to *QP*. This term-ranking strat-
egy exploits both co-occurrence statistics and relevance weights with the aim of
filtering out noise. While $w_i^{(1)}$ assigns higher weight to terms and phrases with
high discriminatory power, *qtf* is a measure of how often a phrase occurs in
answers to relevant question types. For example, while in Table III the phrase

Table IV. A Sample of Candidate Transforms Grouped
into Buckets According to the Transform Length for the
Question Phrase *"what is a"*

Transform Length	Candidate Transform tr_i	wtr_i
3	"is used to"	32.89
	"according to the"	23.49
	"to use a"	21.43
2	"is a"	298.89
	"of a"	94.34
	"refers to"	81.3
1	"usually"	128.23
	"used"	110.39
	"refers"	80.1

"named after" is a better discriminator than "refers to" for the question phrase
"what is a", it does not occur as often, and "refers to" is ultimately ranked higher.
This tradeoff between discrimination and frequency of occurrence, or expected
precision and recall, may be explored in future work. Sample output of this
stage is shown in Table III.

Finally, the candidate transforms are sorted into buckets according to the
number of words in the transform phrase, and up to *maxBucket* transforms
with the highest values of wtr_i are kept from each bucket. In general, we ex-
pect that longer phrases may be processed differently by the search engines.
This step was done in order to include longer, potentially higher precision trans-
forms in the set of candidate transforms, whereas primarily shorter transforms
with higher frequency counts might be chosen otherwise. In Table IV, we show
a sample of phrases with the highest selection weights from each candidate
transform bucket.

3.2.3 *Weighting and Reranking Transforms Using Search Engines.*   In the
third and final stage of training, we evaluate the performance of each candi-
date transform, tr_i, on Web search engines. Figure 5 shows the algorithm for
ranking a set of candidate transforms for a single question phrase and search
engine. The procedure is repeated for all question phrases and search engines
of interest.

In Step (1) of the algorithm we retrieve a set of *<Question, Answer>* pairs
to be used as training examples. Recall that our goal is to select the trans-
forms for each specific question type, indicated by the corresponding question
phrase, for example, "what is a". In order for the transforms to be general
and useful, we need to provide a representative sample to the training al-
gorithm. This training set is created by selecting the question-answer pairs
uniformly from each of the FAQ categories where the specific question phrase
occurs.

For each of the example *<Question, Answer>* pairs and the set of candidate
transforms generated in the previous stage of the process, we apply each trans-
form $\{tr_i\}$ to the *Question* one at a time (Step (2)). Consider *Question* $= \{QP\ C\}$,
where *QP* is the question phrase, and *C* are the remaining terms in the question.
Using transform tr_i we remove the question phrase *QP* and rewrite *Question* as

```
procedure EvaluateTransforms(QP)

(1)     Examples = RetrieveExamples(QP, numExamples)

        for each <Question, Answer> in Examples
          for each candidate transform tr_i
(2)           Query = ApplyTransform(Question, tr_i)
(3)           Results = SubmitQuery(Query, SE)
              for each Document in Results
                docScore = ∅
(4a)            SubDocuments = getSubDocuments(Document, subDocLen)
                for each SD_i in SubDocuments
(4b)                tmpScore = DocumentSimilarity(Answer, SD_i)
                    if (tmpScore > docScore) docScore = tmpScore

(4c)            updateTransformScores(tr_i, docScore)
                updateTransformCounts(tr_i)

(5)     AssignTransformWeights(TransformScores, TransformCounts)
```

Fig. 5. Automatically evaluating the effectiveness of candidate transforms.

$Query = \{$C AND $tr_i\ \}$. For example, consider the candidate transform "refers to" for the question phrase *"what is a"*, and the *<Question, Answer>* pair *<"what is a lisp machine (lispm)", "A Lisp Machine (lispm) is a computer optimized for running Lisp programs, . . ."*>. Applying the transform to the *Question* we obtain a rewritten *Query* = {*lisp machine lispm* AND *"refers to"*}.

The syntax of the querying interface varies for each search engine. For AltaVista we use the NEAR operator instead of AND because we found the NEAR operator to produce significantly better results in preliminary experiments. In the example above, the actual query submitted to AltaVista would be {*lisp machine lispm* NEAR *"refers to"*}. Google treats all the terms submitted in a query with implicit AND semantics in the absence of an explicit OR operator. Note that Google incorporates the proximity of query terms in the document ranking [Brin and Page 1998] and may discard some words that appear in its stopword list.

Unfortunately, stopwords are significant in the context of question answering. A useful transformation for a question "what is X" would be "X is a", which contains stopwords "is" and "a". If submitted as is, the stopwords in this query may be discarded by a search engine. Search engines may provide a mechanism for specifying that a particular stopword in the query is significant, and should not be ignored. For example, Google's syntax to specify that "is" and "a" in the transform above should not be ignored is currently "+is +a". Unfortunately, simply putting a "+" in front of every term in the query does not always work. For example, if a query is submitted with a "+" in front of what Google considers *not* a stopword, at the time of our experiments, the query was ignored completely and only a helpful error message was returned.

Therefore, to use the stopwords properly, we need to know which terms the search engine considers to be stopwords and would discard if submitted

normally. We use a simple algorithm that creates a list of stopwords for a search engine automatically by *probing* the search engine with each term in the candidate transform. More specifically, we create a *dictionary* of all terms used in the transforms, and probe the search engine for each term. If the search engine responds that the term is a stopword and will be ignored, this term is added to the list of stopwords for that search engine. Subsequently, if a transform with that term is used, it will be specified as important using the appropriate syntax for that search engine. For example, a question "what is a binturong" would be transformed for Google into {binturong "refers +to"}.

In Step (3) of Figure 5 the rewritten query *Query* is submitted to the search engine *SE*. At most 10 of the top results returned by *SE* are retrieved. Each of the returned documents *D* is analyzed in Steps (4a), (4b), and (4c). In Step (4a), *SubDocuments* of *D* are generated. In Step (4b), the subdocument SD_i in *SubDocuments* that is most similar to *Answer* is found. In Step (4c), the scores and counts for tr_i are updated based on the similarity of *D* with respect to *Answer*. More details on these steps follow.

In Step (4a) we generate *SubDocuments* from a document to calculate a more accurate similarity measure. Consider original answer *A* and a document *D*, one of the documents retrieved using the transformed query. We make an assumption that answers are *localized*, that is, that the key set of terms or phrases will appear in close proximity of each other—within subdocuments of length *subDocLen*. The subdocuments overlap by *subDocLen*/2 words in order to minimize the possibility that an answer will not be entirely within one of the subdocuments. In other words, given query *Q*, document *D*, and *subDocLen = N*, we break *D* into overlapping subdocuments $SD_1, SD_2, SD_3, SD_4, \ldots$, each starting at successive positions 0, $N/2$, N, $3 \cdot N/2, \ldots$

In Step (4b) we calculate the score of document *D* with respect to *Answer*. We define *docScore(Answer, D)* as the maximum of the similarities of each of the subdocuments SD_i in *D*. More formally, *docScore(Answer, D) = $Max_i(BM25_{phrase}(Answer, SD_i))$*, where $BM25_{phrase}$ is an extension of the *BM25* metric [Robertson and Walker 1997] modified to incorporate phrase weights, calculated as in Equation (1).

The original $BM25$ metric uses relevance weights $w_i^{(1)}$ and topic frequencies as described previously, and is defined for a generic query *Q* and a document as:

$$BM25 = \sum_{i=0}^{|Q|} w_i^{(1)} \frac{(k_1 + 1)tf_i(k_3 + 1)qtf_i}{(K + tf_i)(k_3 + qtf_i)}, \tag{3}$$

where $k_1 = 1.2, k_3 = 1000, K = k_1((1 - b) + b \cdot dl/avdl), b = 0.5, dl$ is the document length in terms, $avdl$ is the average document length in terms, tf_i is the term frequency in the document, and $w_i^{(1)}$ and qtf_i are the relevance weight and query topic frequency as described previously.[4]

In the $BM25_{phrase}$ metric, the terms in the summation (Equation (3)) include phrases with weights learned over the training data, as in Section 3.2.2.

---

[4]We use the simplified version of the metric that was used in the TREC evaluation, where $k_2 = 0$.

Table V. Some of the Top Ranked Transforms for the Question Phrase
"what is a", Automatically Optimized for AltaVista and Google

Search Engine	Transform	WT_i	Search Engine	Transform	WT_i
AltaVista	"is usually"	377.03	Google	"is usually"	280.68
	"refers to"	373.22		"usually"	275.68
	"usually"	371.55		"called"	256.64
	"refers"	370.14		"sometimes"	253.53
	"is used"	360.07		"is one"	253.24

The weight for a term or phrase t is calculated as follows.

$$w = \begin{cases} w_t^{(1)} & \text{if } w_t^{(1)} \text{ is defined for } t \\ \log IDF(t) & \text{if } w_t^{(1)} \text{ is not defined for } t, \text{ but } IDF(t) \text{ is} \\ NumTerms(t) \cdot \sum_{t_i \in t} \log IDF(t_i) & \text{otherwise} \end{cases}$$

This multi-step assignment procedure is used because terms encountered may not be present in the training collection. We use *IDF* weights (Inverse Document Frequency, which is high for rare terms, and low for common terms), derived from a much larger sample (one million Web pages, obtained from the collection of pages used in the TREC Web Track [Hawking et al. 1999]). The last, fallback case is necessary in order to handle phrases not present in the training data. Intuitively, the function assigns the weight of phrase t inversely proportionally to the probability that all the terms in t appear together, scaled to weight occurrences of multi-word phrases higher. This heuristic worked well in our preliminary experiments, but clearly can be improved on with further work. While document ranking is important during run-time operation of the system as described in Section 3.3, reranking of the result set of documents was not the focus of this work.

The overall goal of ranking candidate transforms is to weight highly the transforms that tend to return many relevant documents (similar to the original *Answer*s) and few nonrelevant documents. In Step (5) we calculate weight WT_i of a transform tr_i as the average similarity between the original training answers and the documents returned in response to the transformed query:

$$WT_i = \frac{\sum_{<Q,A>} docScore(A, D_{tr_i})}{Count(D_{tr_i})}, \tag{4}$$

where the sum is calculated over all of the *<Question, Answer>* pairs in the set of examples.

The result of this final stage of training is a set of transforms, automatically ranked with respect to their effectiveness in retrieving answers for questions matching *QP* from search engine *SE*. Two samples of highly ranked transforms for $QP = $ *"what is a"*, the first optimized for the AltaVista search engine and the second for the Google search engine, are shown in Table V.

```
procedure EvaluateQuestion(Question, K)

(1a)  QP = matchQuestionPhrase(Question)
(1b)  (tr,WT) = retrieveTransforms(QP, numTransforms)
(1c)  Results=∅, Documents=∅, Scores=∅

      for each tr_i in tr
(2)       Query = ApplyTransform(Question, tr_i)
(3a)      Results_i = SubmitQuery(Query, SE)
(3b)      Documents += Results_i, Results += Results_i

      for each Results_i in Results
          for each document d_j in Results_i
(4a)          SubDocuments = getSubDocuments(d_j, subDocLen)
              for each SD_k in SubDocuments
(4b)              tmpScore_k = CommonTerms(Query, SD_k)
(4c)          Scores_j += Max_k(tmpScore_k)·WT_i

(5)   RankedDocuments = Sort Documents in decreasing order of Scores
(6)   Return the K RankedDocuments with highest Scores
```

Fig. 6.   Evaluating questions at run-time.

3.3 Run-Time Query Reformulation

Once the set of the best transformations is automatically trained for each question phrase, they are stored as transformation rules. *Tritus* then evaluates a given question using the procedure in Figure 6.

In Step (1a), the system determines if it can reformulate the question by matching known question phrases, with preference for longer (more specific) phrases. For example, "what is a" would be preferred over "what is". In Step (1b), the corresponding sets of transforms and their weights are retrieved. Only the top *numTransforms* transforms are used. In Step (2) each transform is used to rewrite the original question, one transform at a time, resulting in a new query. In Step (3a) the transformed queries are submitted to the search engine and the matching documents are retrieved and stored in Step (3b) as the current result set $Results_i$, and are also appended to the complete set of retrieved documents *Documents*.

In Steps (4a), (4b), and (4c), the returned documents are analyzed and scored based on the similarity of the documents with respect to the *transformed* query. The *CommonTerms* function returns the number of non-stopword terms common between the transformed query and the subdocument. The maximum of this value over all the subdocuments is multiplied by the *weight* of each of the transforms that retrieved the document, resulting in the current *incremental* score for the document. Intuitively, if a document is retrieved by multiple transformed queries with high weight, it will be assigned a high score. Thus, the final score for the document is the sum of the incremental scores computed over the set of transformed queries that helped retrieve the document.

In our original system, described in Agichtein et al. [2001], we used a different reranking method that used the BM25 similarity measure instead of the *CommonTerms* function. This similarity measure weights terms according to

their importance, as determined by the frequency of occurrence of the terms in the training collection. Unfortunately, determining meaningful weights for a large variety of terms requires a prohibitively expensive training phase, hence our choice of the simpler, more robust *CommonTerms* function above.

An additional benefit of using a weighted-sum model to compute the document score becomes evident when we combine output from multiple search engines. Since transform weights are specific for each search engine, each initial document score is then *scaled* with respect to how that search engine performed using that specific transform during training. Thus, the final score of each document is the sum of all scores for the document, over all the transformed queries that retrieved the document, over all search engines that retrieved it. Since each individual score for the document contains the *weight* for the transform that retrieved it, the final ranking of the documents is equivalent to the *weighted average* of individual scores. This simple weighted average approach of combining ranked results from IR systems has been shown to perform best among various combination methods [Croft 2000].

Once all the document scores are computed, the retrieved documents are ranked (Step (5)) with respect to their final scores, and in Step (6), the *K* top-ranked documents are returned as the final result produced by the *Tritus* system. An example search for the question "What is a hard disk" on our working prototype is shown in Figure 1.

4. EXPERIMENTAL SETTING

Tritus was designed to retrieve documents from the Web that are likely to contain answers to a given natural language question. As such, *Tritus* will be trained and evaluated over the *Web at large*. Furthermore, since the goal of *Tritus* is to retrieve a good set of *documents* (as opposed to extracting an exact answer), we evaluate *Tritus* on the document level. Finally, since we do not restrict the type of questions that users can ask, we will use real human judges to evaluate the quality of documents retrieved by *Tritus*. In this section, we first present the details of training *Tritus* for the evaluation (Section 4.1). Then, Section 4.2 lists the retrieval systems that we use in our comparison. Section 4.3 introduces the evaluation metrics for the performance of the retrieval systems, and details of the queries evaluated and relevance judgments are reported in Section 4.4.

4.1 Training Tritus

We used a collection of approximately 30,000 question-answer pairs for training, obtained from more than 270 Frequently Asked Question (FAQ) files on various subjects. Figure 4 shows a sample of the question-answer pairs. We obtained these FAQ files from the FAQFinder project [Burke et al. 1995]. All of the FAQ files used for evaluation are publicly available in parsed form.[5] We evaluated four question types. The number of question-answer training pairs in the collection for each of the question types is shown in Table VI.

---

[5]Available from `http://tritus.cs.columbia.edu/`.

Table VI. The Number of Training Questions for Each Question
Type

Type	Phrase(s)	Question-Answer Pairs in Collection
Where	*"where can i"*	1035
	"where is"	139
	"what is"	2865
What	*"what are"*	1143
	"what is a"	443
How	*"how do i"*	2417
	"how can i"	1371
Who	*"who is"*	225
	"who was"	34

Table VII. *Tritus* Training Parameters

Parameter	Value	Description
minQPhrCount	30	Min. frequency for generating question phrases
minAPhrCount	3	Min. frequency for generating candidate transforms
catSupport	5	Min. number of supporting FAQ categories to generate transforms
maxPhrCount	500	Max. number of most frequent candidate transforms to consider
maxQtokens	4	Max. length of question phrases (in words)
maxAtokens	5	Max. length of answer phrases (in words)
minQtokens	2	Min. length of question phrases (in words)
minAtokens	1	Min. length of answer phrases (in words)
maxLen	4096	Max. length of the prefix of answers from which candidate transforms are generated
subDocLen	10,000	Length (in words) of the subdocuments for document similarity calculation. Set high to include complete example answers in the similarity calculation.
maxBucket	25	Max. number of highest ranked candidate transforms of each length for the final search-engine weighting stage.
numExamples	100	Number of example <*Question, Answer*> pairs used to evaluate candidate transforms for each question phrase
Timeout (sec)	30	Individual page timeout

Tritus uses a number of parameters in the training process. We performed some experimentation with the different values of these parameters resulting in the parameters shown in Table VII. We did not use any of these tuning questions for the actual evaluation of our techniques. We did not test parameters exhaustively and further fine-tuning may improve the performance of the system.

4.2 Retrieval Systems Compared

Recall that *Tritus* starts with a natural language question submitted by a user, and transforms the question into a set of new effective queries for the search engine of interest. The output of *Tritus* is a set of documents, which can either be shown to the user directly, or used as input to a traditional question answering system. Hence, the TREC QA evaluation [Voorhees 1999b, 2000, 2001] is not appropriate for *Tritus*. The first and most important reason is that

Tritus returns documents as its output, whereas the question answering systems evaluated as part of the TREC QA track were required to return a short 50- or 250-byte answer. Additionally, the TREC QA evaluation considered a set of *specific*, *factoid* questions, whereas we consider a more general class of questions where there may be multiple different valuable answers (e.g., "What are ways people can be motivated"), and answers may be lengthy, potentially spanning entire documents. Finally, while the TREC QA evaluation was done over a relatively small controlled collection of documents, *Tritus* is developed to find answer documents on the Web, over real, Web-specific search engines such as Google and AltaVista. Consequently, to evaluate *Tritus* rigorously we proceed with an alternative experimental setting.

Tritus learns query transformations that are specifically tailored for a given search engine. Our experimental evaluation focuses on two popular "general purpose" search engines, Google and AltaVista. We compare the results produced by each of these systems against those of *Tritus* produced by using the corresponding search engine-specific transformations. Additionally, we explored combining the results retrieved by *Tritus* over multiple search engines (AltaVista and Google). We also evaluated AskJeeves for comparison. The six systems evaluated are:

—Google **(GO)**: The Google search engine as is.
—*Tritus* optimized for Google **(TR-GO)**: The retrieval system that results from transforming user questions into multiple queries using transformations specifically learned for Google, and combining the query results from Google as in Section 3.3.
—AltaVista **(AV)**: The AltaVista search engine as is.
—*Tritus* optimized for AltaVista **(TR-AV)**: The retrieval system that results from transforming user questions into multiple queries using transformations specifically learned for AltaVista, and combining the query results from AltaVista as in Section 3.3.
—*Tritus* over both AltaVista and Google **(TR-ALL)**: The retrieval system that results from obtaining documents using both **TR-GO** and **TR-AV** simultaneously, and combining the results with the document reranking scheme of Section 3.3.
—AskJeeves (**AJ**): The results of the AskJeeves search engine, which specializes in answering natural language questions.

4.3 Evaluation Metrics

Information retrieval systems are usually compared based on the quality of the retrieved document sets. This quality is traditionally quantified using two metrics, *recall* and *precision* [Salton 1989]. Each document in the collection at hand is judged to be either *relevant* or *nonrelevant* for a query. *Precision* is calculated as the fraction of relevant documents among the documents retrieved, and *recall* measures the coverage of the system as the fraction of *all* relevant documents in the collection that the system retrieved.

To measure recall over a collection, we need to mark every document in the collection as either relevant or nonrelevant for each evaluation query. This, of course, is a daunting task for any large document collection, and is essentially impossible for the Web, which contains billions of documents. Researchers have addressed this problem by developing standard document collections with queries and associated relevance judgments, and by limiting the domain of documents that are judged [Voorhees 1999a].

Recently, TREC has incorporated a *Web Track* [Hawking et al. 1999] that employs a collection of Web documents (small relative to the size of the Web). This collection of documents is a valuable resource to evaluate information retrieval algorithms over Web data. However, the collection is not well suited to evaluate a system like *Tritus* where we aim to transform user queries to obtain improved results from existing search engines like Google and AltaVista that operate over the Web at large.

To evaluate *Tritus*, we inspect the K pages returned by the various systems that we compare (Section 4.2) for each query that we consider (Section 4.4). We describe how we computed relevance judgments for these documents in Section 4.4. Using these relevance judgments, we evaluate the answers that the systems produce using the *precision*, *helpfulness*, and *Mean Reciprocal Rank* (*MRR*) metrics that we describe next.

Definition 4.1.   The *precision* at K of a retrieval system S for a query q is the percentage of documents relevant to q among the top K documents returned by S for q.

Example 1.   Consider a query q and the top 10 documents returned by Google for this query. If we judge that 8 of these 10 documents are relevant to q, then the precision at 10 of Google for q is $\frac{8}{10} \cdot 100\% = 80\%$.

We also compute *helpfulness*, or the percentage of questions where a given system provides the best performance of all systems tested:

Definition 4.2.   Consider systems $S_1, \ldots, S_n$ and query q. A system S_i, at document cutoff K for a query q, is considered (one of) the best performing systems if S_i has the highest number of relevant documents among the top K documents that it produced, compared to all other systems. The *helpfulness* at K of S is then the percentage of questions on which S was (one of) the best performing systems.

Note that multiple systems may have identical performance on a given question, in which case they may all be considered the best.

Example 2.   Consider a query q, the top 10 documents returned by Google for q, and the top 10 documents returned by AltaVista for the same query. If there are 7 documents relevant to q among the top 10 Google documents and only 5 among the top 10 AltaVista documents, then Google is considered to have the best performance at document cutoff 10 for q. Intuitively, Google gets a "vote" for q because it was the best of the two retrieval systems for that query.

Finally, we compute the *Mean Reciprocal Rank* (*MRR*) [Voorhees and Tice 1999] used in the TREC QA evaluation, which synthesizes our set of precision at K values into one real number between 0 and 1. While the performance of *Tritus* for the Web document retrieval is not directly comparable with the TREC QA system performance figures, we nevertheless report results on the MRR metric to provide an intuitive measure of performance of the compared document retrieval systems.

Definition 4.3. The *Reciprocal Rank* of a system S, for a query q, is defined as $\frac{1}{r_{S,q}}$, where $r_{S,q}$ is the highest rank of a document retrieved by S that is judged relevant for q. The *Mean Reciprocal Rank* (*MRR*) of S is the average of the Reciprocal Rank values of S over all evaluated queries.

Note that, following the TREC QA definition of MRR, systems are not given any credit for retrieving multiple relevant documents for each question, and only one highest-ranked relevant document is considered from only the 5 top-ranked documents. In contrast, *Tritus* and other evaluated systems return up to 10 documents, where often more than one of these can be relevant for a query. However, since MRR has become a standard measure for evaluating question answering systems, we decided to implement it exactly as specified in Voorhees and Tice [1999].

Example 3. Consider a query q and the top 5 documents returned by Google for this query. If we judge that there are two *relevant* documents for q that were ranked 3 and 5, then the Reciprocal Rank for Google for q is $\frac{1}{3} = 0.33$.

4.4 Evaluation Queries and Their Relevance Judgments

Once we have trained *Tritus* as discussed above, we evaluate its performance against the retrieval systems in Section 4.2, using the metrics described in Section 4.3. We used real user questions from a log of queries received by the Excite search engine on the 20th of December, 1999. The portion of the log that we have access to consists of 2.5 million queries, out of which we estimate that around 290,000 are reasonably well-formed English questions. We focus our evaluation on four basic question types: *Where, What, How,* and *Who* (Table VIII). These are the most common types of questions found in the Excite log, and have been estimated to account for more than 90% of natural language questions submitted to the search engine [Spink et al. 2000].

The set of test questions was generated by scanning for the question phrases in the query log. A random sample of 50 questions was chosen for each question type. The sample was manually filtered to remove queries that might be offensive or are likely to return offensive documents. We also checked that the questions were not present verbatim in our training collection. None of the test questions were used for tuning the parameters of *Tritus*.

The test questions, 50 of each question type from the queries in the Excite query log (Table IX), are submitted to all of the six systems (Section 4.2) without any modifications to the original question. From each system, we retrieve up to 10 top-ranked URLs. The rank for each of the result URLs is stored. The

Table VIII. The Number of Questions of Each
Question Type in the Test Collection (Derived
from the Excite query log from 1999)

Type	Phrase(s)	Queries
Total queries		2,477,283
Where	"where"	184,634
	"where can i"	162,929
	"where is"	7,130
What	"what"	69,166
	"what is"	35,290
	"what are"	10,129
	"what is a"	5,170
How	"how"	20,777
	"how do i"	13,790
	"how can i"	13,790
Who	"who"	16,302
	"who was"	1,862
	"who is"	5,518
Total questions		290,000 (approx.)

Table IX. Parameters Used for Evaluation

Parameter	Value	Description
numTransforms	15	Max. number of transforms to apply
subDocLen	50	Size of the sub-document used for reranking
Timeout	30	Individual page timeout

top 10 documents (or all documents if fewer than 10) are retrieved as follows:

—**AV**: The top 10 documents returned by AltaVista.

—**GO**: The top 10 documents returned by Google.

—**TR-AV**: The top 10 documents returned by *Tritus* using transforms optimized for AltaVista and the parameters in Table IX.

—**TR-GO**: The top 10 documents returned by *Tritus* using transforms optimized for Google and the parameters in Table IX.

—**TR-ALL**: The top 10 documents returned by *Tritus* by combining documents retrieved by **TR-GO** and **TR-AV**.

—**AJ**: The top 10 documents returned by AskJeeves. We had to take special steps to process the AskJeeves results. In the AskJeeves interface, the first page of results returned may contain one or more of the following:

(1) One or more links to best page(s) for the given question (or a similar question), selected by a professional editor.

(2) One or more drop-down lists with a preselected value of a term (or terms) in the question and an associated "submit" button, which, if selected, will take the user to an answer.

We retrieved all of the results in (1) and (2) (in that order) to bring the total number of retrieved pages to at most 10.

To assign relevance judgments fairly, the resulting URLs from all of the systems are mixed together and presented in random order to a number of

Table X. The Number of Questions of Each Question
Type Evaluated by the Judges During the 2000
Evaluation

Type	Phrase(s)	Presented to judges	Evaluated
Where	*"where"* *"where can i"* *"where is"*	46 47	10 11
What	*"what"* *"what is"* *"what are"* *"what is a"*	45 43 48	11 12 15
How	*"how"* *"how do i"*	44	15
Who	*"who"* *"who was"*	40	15
Total questions		313	89

volunteers. The volunteers are blind to the system that returned the URLs. We set up an evaluation script to present volunteers with the result pages for a query one page at a time, and allow the volunteers to judge each page as "good", "bad" or "ignore" for the query.

The volunteers were told to use the following criteria to judge documents. A good page is one that contains an answer to the test question on the page. If a page is not relevant, or only contains links to the useful information, even if links are to other documents on the same site, the page is judged bad. If a page could not be viewed for any reason (e.g., server error, broken link), the result is ignore.

5. EVALUATION RESULTS

In this section, we report the results of the experimental evaluation using the methodology described in the previous section. First, in Section 5.1 we report the results of our original evaluation performed in the Fall of 2000, which totalled 89 questions evaluated by volunteers, mostly acquaintances and colleagues of the authors who were requested to help with the evaluation. In Section 5.2 we present the results of the new, more extensive evaluation performed in the Spring of 2002, which additionally involved anonymous and unknown judges that participated in evaluating all of the updated systems as described above.

5.1 Results from the Original (2000) Evaluation

During this evaluation, 89 questions were evaluated by volunteer judges. Table X lists the number of questions of each type that were presented to the judges and the number of questions that were actually evaluated by the judges.

Figure 7(a) shows the average precision at K for varying K of **AJ**, **AV**, **GO**, **TR-GO**, and **TR-AV** over the 89 test questions. (**TR-ALL** is new and was not part of the 2000 evaluation.) As we can see, *Tritus*, optimized for Google, has the highest precision at all values of document cutoff K. Also note that both

(a)   (b)   (c)
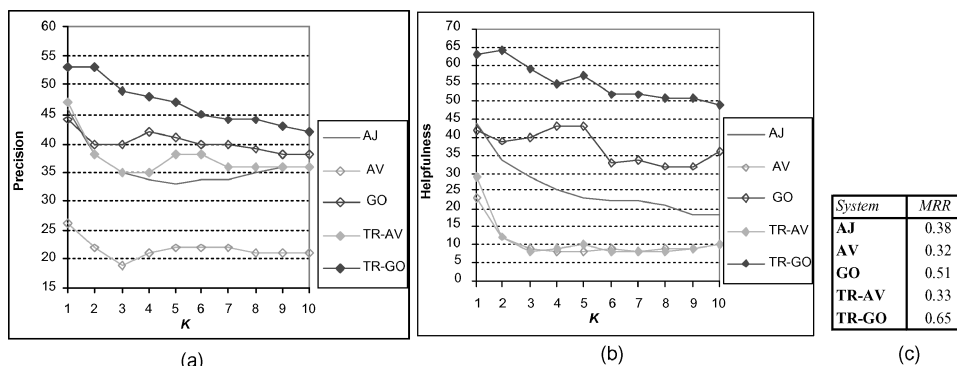
Fig. 7.   Average precision (a), helpfulness (b), and MRR (c) of **AJ**, **AV**, **GO**, **TR-AV**, and **TR-GO** over 89 test queries for varying numbers of top documents examined during the 2000 evaluation.

TR-GO and **TR-AV** perform better than the underlying search engine used. **TR-AV** shows a large improvement over **AV**.

Figure 7(b) shows the average helpfulness at K for varying K over all 89 test questions. As we can see, **TR-GO** performs the best on more questions than any other system. Even though **TR-AV** performs relatively poorly on this metric, its performance is comparable to the original AltaVista search engine. Because the lower performing systems perform best for only a small number of questions, comparison of the lower performing systems on this metric is not very meaningful.

Figure 7(c) reports the MRR values for all systems. **TR-GO** performs substantially better than the other systems, noticeably improving the performance of **GO**, the underlying search engine. Interestingly, while **TR-AV** has higher precision than **AV**, their MRR values are very similar: while the documents retrieved by **TR-AV** have high precision, **TR-AV** was not able to return any documents for some queries, thereby not improving **AV**'s performance on the helpfulness and MRR metrics.

In Figure 8, we report the average precision at K of the document sets retrieved by **AJ**, **AV**, **GO**, **TR-GO**, and **TR-AV**, separated by question type. Note that at least one of the *Tritus* systems has the highest precision at K for 3 out of 4 question types (*What, Where, and Who* in Figures 14(a), (c) and (d)) for $K \leq 8$, while **GO** has the highest precision for *How*.[6]

Google's impressive performance on the *How* questions may be due to a common practice on the Web of linking to the pages that people find useful for solving specific problems that, by their nature, contain good answers to *How* types of questions. Since Google exploits *anchor text* for document retrieval, it achieves high results for this type of question.

---

[6]The volunteer judges complained that the *How* questions were the hardest to evaluate. For example, an answer to a test question *"How do I create a Web page?"* can take many different forms, from a direction *"Click here to create your free Web page,"* to HTML references, Web hosting promotions, and Web design software manuals. Often it was not clear whether to judge a page relevant or not for this type of question.
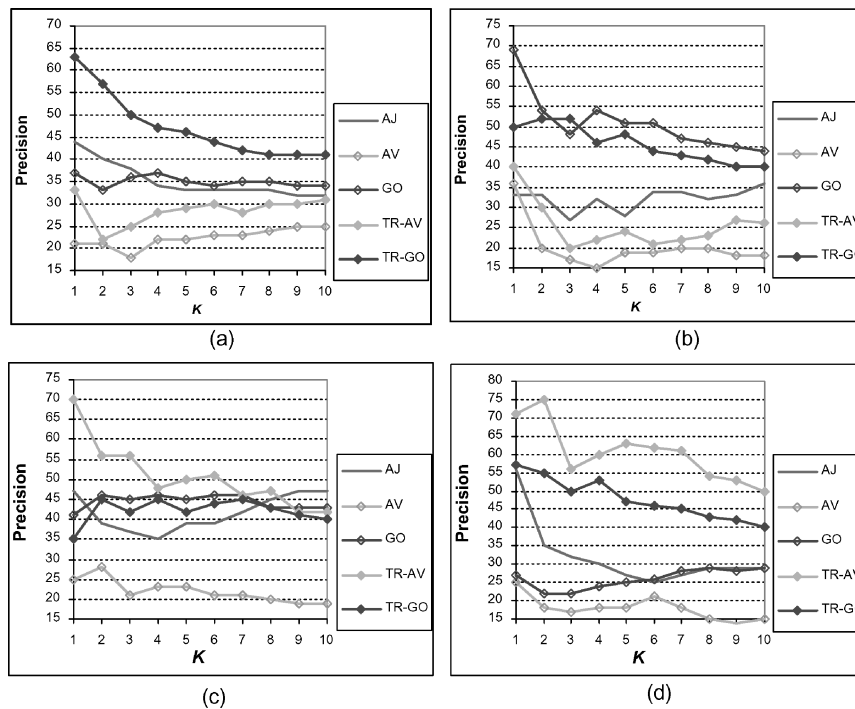
Fig. 8. Average precision of **AJ**, **AV**, **GO**, **TR-AV**, and **TR-GO** by question type: *What* (a), *How* (b), *Where* (c), and *Who* (d), during the 2000 evaluation.

It is interesting to note that the systems do not perform uniformly well across different question types, or even across different subtypes of the basic question type. We can explain this phenomenon by observing that questions such as "What are" and "What is a," even though both questions fall into the *What* question type, are typically used to express different purposes. "What is a" usually indicates a request for a definition or explanation of a term or concept, while "What are" often indicates a request for a list of characteristics or features of a concept.

Also note that **TR-GO** performs better than **TR-AV** on *What* and *How* types of questions, while **TR-AV** is clearly better at *Where* and *Who* questions. This suggests an approach of routing the questions to particular search engine(s) that perform well for a given question type and transforming the question in a manner optimal for that search engine. Note that the relatively small number of questions in each category limits the accuracy of the results for individual question types.

5.2 Results from the New (2002) Evaluation

We now report results from the new evaluation, performed more than a year after the original evaluation. As expected, the performance of all of the systems compared has improved over that time period. We discovered in our preliminary experiments, which are not reported here, that the queries generated by *Tritus*

Fig. 9.   Evaluation directions presented to all judges during the 2002 evaluation.

were still good, and returned many documents containing answers to questions. However, the original reranking method described in Section 3.2.3 was not performing very well. Therefore, we adopted a more robust reranking algorithm (Section 3.3), as well as a variation of *Tritus* that combined results of different search engines in order to improve the performance of *Tritus* (**TR-ALL**).

The current evaluation was performed by two groups of judges: the first group of volunteers, which we refer to as **colleagues**, was a group of acquaintances and colleagues, whom we directly asked to help in the evaluation. The second group, which we will refer to as **CiteSeer**, was recruited via a link placed on CiteSeer (http://citeseer.org/), a large online repository of computer science related publications [Lawrence et al. 1999]. Because of the large time commitment required to perform the evaluation, we thought that recruiting participants with CiteSeer might help increase the number of questions evaluated, although we were concerned that unknown participants obtained in this manner might not perform very careful evaluations. Both groups were shown the same directions (Figure 9) and each judge then proceeded to rate pages retrieved by each system in response to a particular question as described above. The ratings of both groups together form the **combined** results. The number of test questions of each type presented to judges during the 2002 evaluation is shown in Table XI. The same test questions were presented to each of the groups of judges.

The results of the evaluation by the **colleagues** group are similar to the encouraging results of the 2000 evaluation, and are shown in Figure 10. As before, both **TR-GO** and **TR-AV** perform substantially better than the underlying search engines, **GO** and **AV**. Interestingly, on this evaluation **TR-AV** also noticeably improves the MRR score of the underlying search engine, **AV**. As we

Table XI. The Number of Questions of Each
Question Type Evaluated by the Judges During the
2002 Evaluation

Type	Phrase(s)	Presented to judges	Evaluated
Where	*"where"* *"where can i"* *"where is"*	40 49	5 10
What	*"what"* *"what is"* *"what are"* *"what is a"*	47 48 50	15 10 19
How	*"how"* *"how do i"* *"how can i"*	47 47	8 12
Who	*"who"* *"who was"* *"who is"*	49 47	22 12
Total questions		424	113

System	MRR
AJ	0.27
AV	0.39
TR-AV	0.61
GO	0.58
TR-GO	0.76
TR-ALL	0.77

Fig. 10. Average precision (a), helpfulness (b), and MRR (c) of **AJ**, **AV**, **GO**, **TR-AV**, **TR-GO**, and **TR-ALL** evaluated by the **colleagues** group of judges during the 2002 evaluation.

expected, **TR-ALL** provides an additional modest improvement over **TR-GO**, and narrowly outperforms **TR-GO** as the best performing system on precision, helpfulness, and MRR metrics for all values of K.

The results of the evaluation by the **CiteSeer** group are more noisy. Many of the volunteer judges were not motivated enough to complete the 20–30 minute evaluation and rated only some of the pages in the set. As we thought, the time commitment required to carefully read and follow the instructions was too much to ask from unknown participants. For example, by examining a sample of the ratings, we observed that participants did not always follow the directions and assigned a good rating to pages that did not actually contain an answer, but rather had a link to the answer in a different document. Despite this, we did not manually filter any of the ratings. The results are shown in Figure 11. In this evaluation, **TR-GO** did not improve **GO's** performance on the precision metric, but did improve performance on the helpfulness metric, suggesting that **TR-GO** more consistently performs well across questions. The decrease of performance
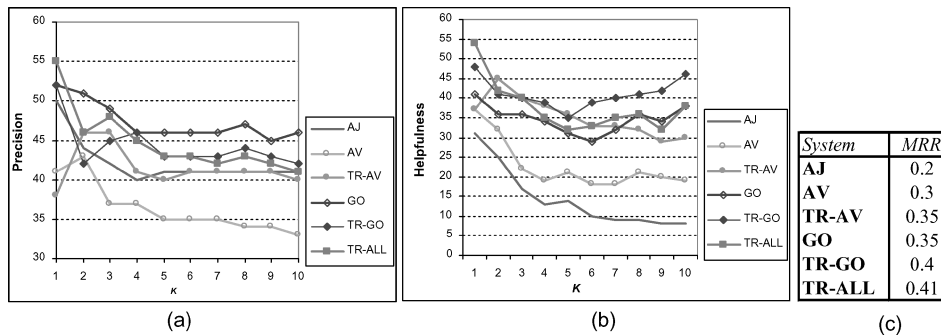
Fig. 11. Average precision (a), helpfulness (b), and MRR (c) of **AJ**, **AV**, **GO**, **TR-AV**, **TR-GO**, and **TR-ALL** evaluated by the **CiteSeer** group of judges during the 2002 evaluation.
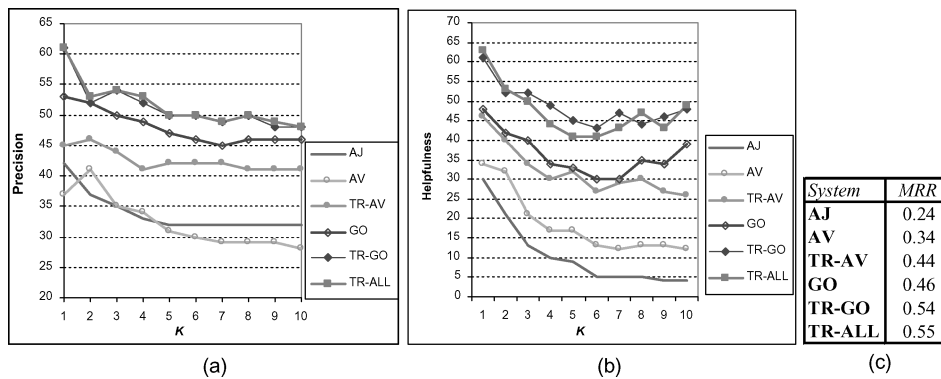

Fig. 12. Average precision (a), helpfulness (b), and MRR (c) of **AJ**, **AV**, **GO**, **TR-AV**, **TR-GO**, and **TR-ALL** as calculated using the **combined** results from the **colleagues** and **CiteSeer** ratings during the 2002 evaluation.

for *Tritus* in this evaluation may be related to the tendency of the unknown participants to mark pages good if they contained a link to an answer, especially with regard to **GO**, which tends to return authoritative sites that link to a lot of information. The increase in performance for **AV** and **AJ** according to the **CiteSeer** evaluators provides additional support for our belief that the **CiteSeer** group did not perform careful and accurate evaluations (compared to the **colleagues** group, the results tend to move towards the mean, which would happen with random evaluations). **TR-AV** still substantially outperforms **AV** in this evaluation. Furthermore, both **TR-GO** and **TR-AV** noticeably improve the MRR values of the underlying search engines, **GO** and **AV**.

Finally, results from the **combined** group are shown in Figure 12. These results are the most comprehensive (containing evaluation results for 113 questions), but still contain some noise inherited from the incomplete and less careful evaluations from the **CiteSeer** group. However, we can still see the large improvement of **TR-AV** over **AV**, and a modest improvement of **TR-GO** and **TR-ALL** over **GO**. Also note that the actual precision of the respective *Tritus*
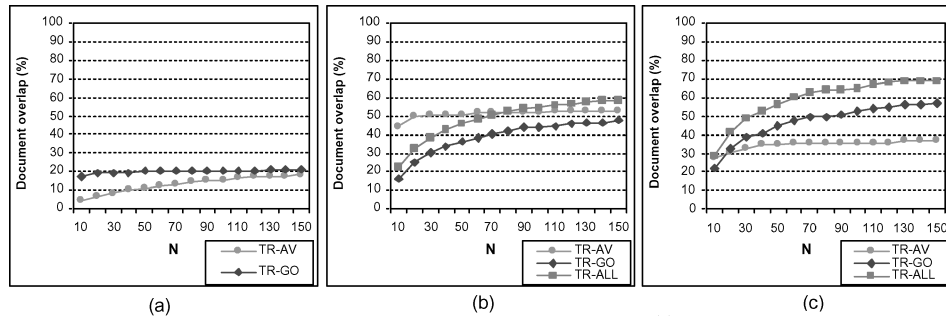
Fig. 13. Average percentage of all (a), top 10 (b), and relevant in top 10 (c) **TR-GO**, **TR-AV**, and **TR-ALL** documents contained in top N documents returned for each original query by the underlying search engine during the 2002 evaluation.

implementations is somewhat higher than the corresponding precision from the 2000 evaluation.

In the rest of this section, we analyze the performance of *Tritus* in more detail. We first consider the overlap of the documents retrieved by *Tritus* compared to the information retrieval system using the original queries, but retrieving more documents. We then examine the effect that a successful reranking strategy has on the quality of the final top 10 documents returned to the user.

We have considered the possibility that the documents returned by *Tritus* could also be retrieved simply by examining more documents returned by the search engines for the original query. We report the percentage of *Tritus* documents contained in the set of responses to the original query as more documents are examined in Figure 13. Figure 13(a) reports the percentage of all *Tritus* documents (without reranking) that are contained in the top N (10–150) search engine responses for the original query. In Figure 13(b), we plot the percentage of top 10 *Tritus* documents (after reranking) contained in the original search engine results, and in Figure 13(c), we show the percentage of relevant *Tritus* documents contained in the original search engine results. For **TR-ALL**, the overlap is measured against the union of the top 150 documents from both **AV** and **GO**. Only the top 10 *Tritus* documents had relevance judgments assigned. The figures show that, for **TR-AV**, there is very little overlap between result sets retrieved in response to the transformed queries and the documents retrieved for the original query. For **TR-GO**, the overlap is low for all of **TR-GO** documents (without reranking), but increases and levels off at around 50% for the top 10, and between 50% and 60% for the relevant *Tritus* documents as more of the underlying search engine documents are examined. These experiments indicate that a significant fraction of the relevant documents retrieved by *Tritus* would not be found using an underlying search engine. Interestingly, the overlap is quite high (as high as 69%) for **TR-ALL**, since the reranking algorithm that we use tends to rank the more "popular" pages higher (i.e., those retrieved by multiple search engines and by multiple transformed queries).

In Figure 14, we report the average precision at K of the document sets retrieved by **AJ**, **AV**, **GO**, **TR-GO**, **TR-AV**, and **TR-ALL**, separated by question type. At least one of the *Tritus* systems has the highest precision at all values
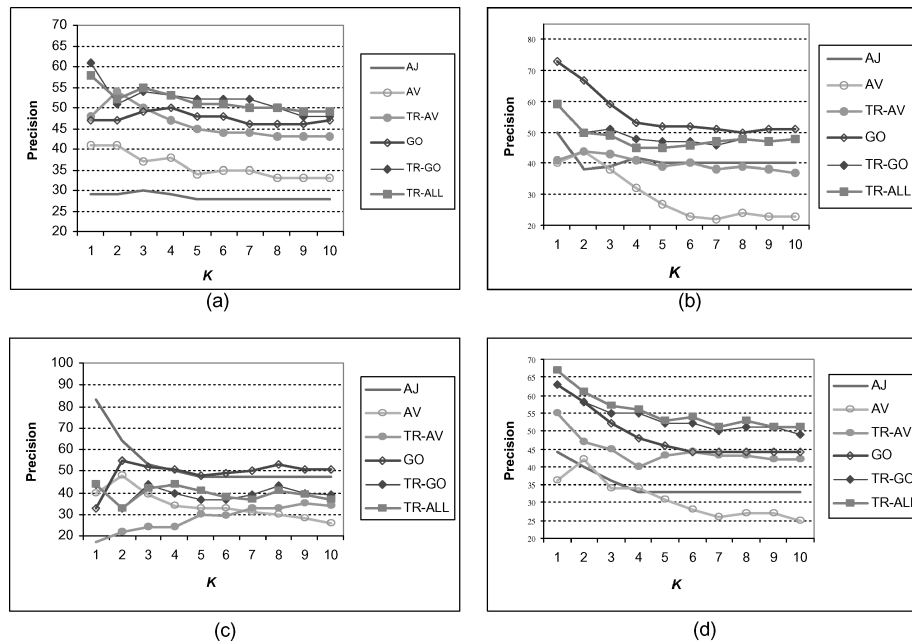
Fig. 14.   Average precision of **AJ**, **AV**, **GO**, **TR-AV**, **TR-GO**, and **TR-ALL** by question type: *What* (a), *How* (b), *Where* (c), and *Who* (d) using the **combined** set of results on the 2002 evaluation.

of *K* for 2 out of 4 question types (*What* and *Who* in Figures 14(a) and (d)), while Google has the highest precision for *How* (as in the original evaluation), and Google and AskJeeves share the best performance for *Where*. However, note that *Where* and *How* had the smallest number of questions evaluated, especially from the **colleagues** group, so the results for these questions types may not be significant.

One of the challenges in evaluating a system like *Tritus* is that it is difficult to separate the querying formulation performance and the reranking performance of the system in a real evaluation. Unfortunately, since a human judge needs to read the retrieved documents, evaluating all documents retrieved before the reranking is not feasible. Thus, we are forced to solve the additional hard problem of reranking and merging output from information retrieval systems. Alternative reranking and answer extraction from the retrieved documents could substantially improve performance. For example, a system such as the one described in Radev et al. [2002] could be used to select the best candidate answer phrases among all of the documents retrieved by *Tritus*.

Since we focus on the problem of learning effective queries for retrieving a good candidate set of documents, a revealing performance characteristic is the overall quality of the *complete* retrieved document set, rated by the human judges, as opposed to just the top 10 documents after reranking. Since we found that there is significant overlap among the documents retrieved by the systems, we can exploit this extra information to estimate the relevance of some of the documents that were retrieved by the systems, but not ranked in the top 10 documents for that system. By using the combined pool of relevant
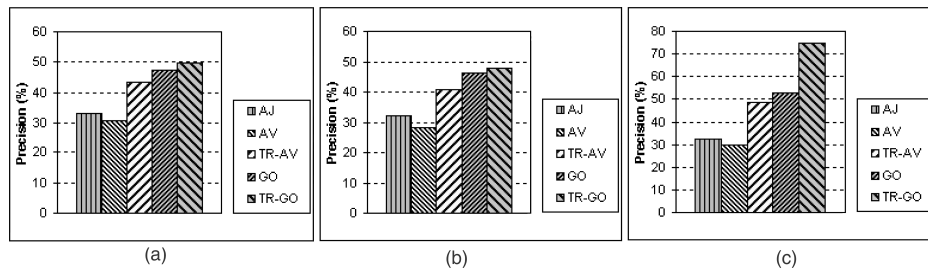
Fig. 15.   Average precision of **AJ**, **AV**, **GO** and *Tritus* (**TR-AV**, **TR-GO**) in the 2002 evaluation for all 150 documents retrieved (a), the top 10 documents using the current reranking method (b), and the top 10 documents if a perfect reranking method was used (c).

and nonrelevant documents as rated for all of the systems in question, we can estimate the average precision of the complete document set retrieved by each system.

The overall precision for all basic systems is shown in Figure 15. Note that the current reranking methods used by each evaluated system do not appear to substantially improve the quality of the documents: the average precision across the top 10 documents for each system is estimated to be similar to the average precision across the *complete* set of the top 150 documents. For example, the average precision of **TR-GO** over all 150 documents (but considering only documents for which we have relevance judgments) is 50%, while the precision of the top 10 documents selected by the current reranking method is 48%. In contrast, if a perfect reranking method was used to select the top 10 documents, the quality of the top 10 documents increases substantially. For example, the average precision of the top 10 documents reranked from the set retrieved by **TR-GO** would increase from 48% to 75%, while the precision of **GO** would increase from 46% to 53%. The precision of this "perfect" set of top 10 documents is calculated by assuming that a perfect reranking mechanism would float all of the known relevant documents for each question to the top 10, and the remaining slots in the top 10, if any, would be filled with the documents that have been rated as nonrelevant for that question. Most encouragingly, the precision of **TR-GO** and **TR-AV** increases dramatically if the perfect reranking is used to rerank documents returned by all of the evaluated systems.

It is clearly not ideal that the evaluation of the system depends, even if implicitly, on the quality of reranking. One way to avoid this problem in the future might be to adopt a more automatic method of evaluation, which would allow us to examine *all* of the documents retrieved by the transformed queries from *Tritus*, not just the top 10, and compare them to the documents retrieved by the original queries. Automatic evaluation could be performed with a subset of questions where automatically identifying the known correct answer is easy; however, the results might not accurately represent performance on a more general class of questions. In contrast, we have evaluated the systems over a set of questions drawn from query logs of a real Web search engine.

6. FUTURE WORK AND SUMMARY

Many avenues exist for future research and improvement of our system. For example, existing methods for extracting the best passages from documents could be implemented. Domain knowledge, heuristics, and natural language parsing techniques could be used to improve the identification of question types. Multiple transformations could be combined into a single query. Questions could be routed to search engines that perform best for the given question type. Additionally, an interesting direction to explore is creating phrase transforms that contain content words from the questions. Yet another direction of research would be to make the transformation process dynamic. For example, transformations where we expect high precision could be submitted first. Based on the responses received, the system could try lower precision transforms or fall back to the original query.

In summary, we have introduced a method for learning query transformations that improves the ability to retrieve documents with answers to questions using an information retrieval system. The method involves classifying questions into different question types, generating candidate query transformations from a training set of question/answer pairs, and evaluating the candidate transforms on the target information retrieval systems. We have implemented and thoroughly evaluated the method as applied to Web search engines. In two separate and extensive blind evaluations more than a year apart, we have shown that the method substantially outperforms the underlying Web search engines.

REFERENCES

ABNEY, S., COLLINS, M., AND SINGHAL, A. 2000. Answer extraction. In *Proceedings of the Applied Natural Language Processing Conference (ANLP-2000)*. 296–301.

AGICHTEIN, E., LAWRENCE, S., AND GRAVANO, L. 2001. Learning search engine specific query transformations for question answering. In *Proceedings of the World Wide Web Conference (WWW-10)*. 169–178.

ALIOD, D., BERRI, J., AND HESS, M. 1998. A real world implementation of answer extraction. In *Proceedings of the 9th International Workshop on Database and Expert Systems, Workshop: Natural Language and Information Systems (NLIS-98)*. 143–148.

BERGER, A., CARUANA, R., COHN, D., FREITAG, D., AND MITTAL, V. O. 2000. Bridging the lexical chasm: statistical approaches to answer-finding. In *Proceedings of the ACM SIGIR Conference*. 192–199.

BRILL, E. 1992. A simple rule-based part of speech tagger. In *Proceedings of the Applied Natural Language Processing Conference (ANLP-92)*. 152–155.

BRILL, E., LIN, J., BANKO, M., DUMAIS, S., AND NG, A. 2001. Data-intensive question answering. In *Proceedings of the TREC-10 Question Answering Track*. 393–400.

BRIN, S. AND PAGE, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Comput. Netw. ISDN Syst. 30*, 1–7, 107–117.

BURKE, R., HAMMOND, K., AND KOZLOVSKY, J. 1995. Knowledge-based information retrieval for semi-structured text. In *AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval*. 19–24.

CARDIE, C., NG, V., PIERCE, D., AND BUCKLEY, C. 2000. Examining the role of statistical and linguistic knowledge sources in a general-knowledge question answering system. In *Proceedings of the Applied Natural Language Processing Conference (ANLP-2000)*. 180–187.

CROFT, W. B. 2000. Combining approaches to information retrieval. *Advan. Info. Retrieval*. 1–36.

GLOVER, E., FLAKE, G., LAWRENCE, S., BIRMINGHAM, W. P., KRUGER, A., GILES, C. L., AND PENNOCK, D. 2001. Improving category specific Web search by learning query modifications. In *Symposium on Applications and the Internet (SAINT-2001)*. 23–31.

HARABAGIU, S. M., PASCA, M. A., AND MAIORANO, S. J. 2000. Experiments with open-domain textual question answering. In *Proceedings of the International Conference on Computational Linguistics (COLING-2000)*. 292–298.

HAWKING, D., CRASWELL, N., THISTLEWAITE, P., AND HARMAN, D. 1999. Results and challenges in Web search evaluation. *Computer Networks (Amsterdam, Netherlands) 31*, 11–16, 1321–1330.

HOVY, E., GERBER, L., HERMJAKOB, U., JUNK, M., AND LIN, C.-Y. 2000. Question answering in Webclopedia. In *Proceedings of the TREC-9 Question Answering Track*. 655–672.

ITTYCHERIAH, A., FRANZ, M., ZHU, W.-J., AND RATNAPARKHI, A. 2000. IBM's statistical question answering system. In *Proceedings of the TREC-9 Question Answering Track*. 231–234.

JOHO, H. AND SANDERSON, M. 2000. Retrieving descriptive phrases from large amounts of free text. In *Proceedings of the International Conference on Knowledge Management (CIKM-2000)*. 180–186.

KLAVANS, J. L. AND KAN, M.-Y. 1998. Role of verbs in document analysis. In *Proceedings of the International Conference on Computational Linguistics (COLING/ACL-98)*. 680–686.

KWOK, C. C. T., ETZIONI, O., AND WELD, D. S. 2001. Scaling question answering to the Web. In *Proceedings of the World Wide Web Conference (WWW-10)*. 150–161.

LAWRENCE, S., BOLLACKER, K., AND GILES, C. L. 1999. Indexing and retrieval of scientific literature. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM-99)*. 139–146.

LAWRENCE, S. AND GILES, C. L. 1998. Context and page analysis for improved web search. *IEEE Internet Comput. 2*, 4, 38–46.

MANN, G. 2002. Learning how to answer questions using trivia games. In *Proceedings of the International Conference on Computational Linguistics (COLING-2002)*.

MILLER, G. A. 1995. Wordnet: A lexical database for English. *Comm. ACM*. 39–41.

MITRA, M., SINGHAL, A., AND BUCKLEY, C. 1998. Improving automatic query expansion. In *Proceedings of the ACM SIGIR Conference*. 206–214.

MOLDOVAN, D., HARABAGIU, S., PASCA, M., MIHALCEA, R., GOODRUM, R., GIRJU, R., AND RUS, V. 1999. Lasso: A tool for surfing the answer net. In *Proceedings of the TREC-8 Question Answering Track*. 175–184.

PRAGER, J., CHU-CAROLL, J., AND CZUBA, K. 2002. Statistical answer-type identification in open-domain question answering. In *Proceedings of the Human Language Technology Conference (HLT-2002)*. 137–143.

RADEV, D., FAN, W., QI, H., WU, H., AND GREWAL, A. 2002. Probabilistic question answering on the Web. In *Proceedings of the World Wide Web Conference (WWW-2002)*. 408–419.

RADEV, D. R., QI, H., ZHENG, Z., BLAIR-GOLDENSOHN, S., FAN, Z. Z. W., AND PRAGER, J. M. 2001. Mining the web for answers to natural language questions. In *Proceedings of the International Conference on Knowledge Management (CIKM-2001)*. 143–150.

ROBERTSON, S. 1990. On term selection for query expansion. In *J. Document. 46*, 359–364.

ROBERTSON, S. AND SPARCK-JONES, K. 1976. Relevance weighting of search terms. *J. Amer. Soc. Info. Sci. 27*, 129–146.

ROBERTSON, S. AND WALKER, S. 1997. On relevance weights with little relevance information. In *Proceedings of the ACM SIGIR Conference*. 16–24.

ROBERTSON, S., WALKER, S., AND BEAULIEU, M. 1998. Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive track. In *TREC-7 Proceedings*. 253–264.

Rocchio, J. 1971. Relevance feedback in information retrieval, G. Salton, Ed. *The SMART Retrieval System–Experiments in Automatic Document Processing*. 313–323.

Salton, G. 1989. *Automatic Text Processing: The transformation, analysis, and retrieval of information by computer*. Addison-Wesley.

Schiffman, B. and McKeown, K. R. 2000. Experiments in automated lexicon building for text searching. In *Proceedings of the International Conference on Computational Linguistics (COLING-2000)*. 719–725.

Spink, A., Milchak, S., Sollenberger, M., and Hurson, A. 2000. Elicitation queries to the Excite Web search engine. In *Proceedings of the International Conference on Knowledge Management (CIKM-2000)*. 134–140.

Voorhees, E. 1999a. Overview of the Eighth Text REtrieval Conference (TREC-8). In *Proceedings of TREC-8*. 1–24.

Voorhees, E. 1999b. The TREC-8 question answering track report. In *Proceedings of TREC-8*. 77–82.

Voorhees, E. 2000. Overview of the TREC-9 question answering track. In *Proceedings of TREC-9*. 71–80.

Voorhees, E. 2001. Overview of the TREC-2001 question answering track. In *Proceedings of TREC-10*. 42–51.

Voorhees, E. and Tice, D. M. 1999. The TREC-8 question answering track evaluation. In *Proceedings of TREC-8*. 84–106.

Xu, J. and Croft, W. B. 2000. Improving the effectiveness of information retrieval with local context analysis. *ACM Trans. Info. Syst. (TOIS) 18*, 1, 79–112.