Learning Search Engine Specific Query Transformations for Question Answering

Eugene Agichtein [*]
Columbia University
New York, NY 10027
eugene@cs.columbia.edu

Steve Lawrence
NEC Research Institute
Princeton, NJ 08540
lawrence@research.nj.nec.com

Luis Gravano
Columbia University
New York, NY 10027
gravano@cs.columbia.edu

ABSTRACT

We introduce a method for learning query transformations that improves the ability to retrieve answers to questions from an information retrieval system. During the training stage 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 transforms 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 re-ranking 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 web search engines as well as a commercial search engine specializing in question answering.

Keywords

Web search, query expansion, question answering, information retrieval

1. INTRODUCTION

A significant number of 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 pre-compiled information, metasearching, and other proprietary methods, while services such as AskMe (www.askme.com) facilitate interaction with human experts.

Web search engines such as AltaVista (www.altavista.com) and Google (www.google.com) typically treat a natural language question as a list 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 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 company websites 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 most 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 containing terms and phrases expected to appear in documents containing answers to the questions (Section 3). We evaluate *Tritus* on a set of questions chosen randomly from the Excite query logs, and 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).

2. RELATED WORK

There is a large body of research on Question-Answering, most recently represented in the Text Retrieval Evaluation Conference (TREC) Question-Answering track [22], which involves retrieving a short (50 or 250 byte long) answer to a set of test questions. In our work we 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?").

The first part of the task, relevant to our work here, is to retrieve promising documents from a collection. The systems in the latest TREC evaluation submitted the original questions to various information retrieval systems for this task [22].

A number of systems aim to extract answers from documents. For example, Abney et al. [1] describe a system in which documents returned by the SMART information retrieval system are processed to extract answers. Questions are classified into one of a set of known "question types" that identify the type of entity corresponding to 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., and Aliod et al. [13, 2] present systems that re-rank and postprocess the results of regular information retrieval systems with the goal of returning the best passages. Cardie et al. [7] 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.

The systems above use the general approach of retrieving documents or passages that are similar to the original question with

[*]Work partially done while the author was an intern at NEC Research Institute during Summer 2000.

WWW10, May 1-5, 2001, Hong Kong.
ACM 1-58113-348-0/01/0005.

variations of standard TF-IDF term weighting schemes [18]. The most promising passages are chosen from the documents returned 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. This is particularly important when retrieving documents is expensive or limited to a certain number of documents, as is the case with web search engines.

Other systems attempt to modify queries in order to improve the chance of retrieving answers. Lawrence and Giles [11] introduced *Specific Expressive Forms*, 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 [11] the transforms are hand crafted (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 [10] use a set of hand-crafted query transformations in order to retrieve documents containing descriptive phrases of proper nouns. Harabagiu et al. [8] describe a system that transforms questions using a hierarchy of question types. The hierarchy is built semi-automatically using a bootstrapping technique. Schiffman and McKeown [19] 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.

Also related is a large body of research (e.g., [12]) that describes methods for automatically expanding queries based on the relevance of terms in the top-ranked documents. An interesting approach presented in [23] describes how to automatically expand a query based on the co-occurrence of terms in the query with the terms in the top-ranked documents for the original query. In general, automatic query expansion systems expand queries at run time on a query-by-query basis using an initial set of top-ranked documents returned by the information system in response to the original query.

In contrast to the previous research, 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

Often, it is not sufficient to submit a natural language question (e.g., *"How do I tie shoelaces?"*) to a search engine in its original form. Most search engines will treat such a query as a bag of terms and retrieve documents similar to the original query. Unfortunately, the documents with the best answers may contain only one or two terms from the original query. 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. In this section we define a strategy for transforming natural language questions into effective search engine queries. Section 3.1 describes how our *Tritus* system learns to reformulate questions. Then, Section 3.2 shows how *Tritus* evaluates a query at run time by applying these transformations.

```
(1)  Generate Question Phrases from Questions
       in Training Data (Section 3.1.1)
(2)  Generate Candidate Transforms from Answers
       in Training Data (Section 3.1.2)
(3)  Evaluate Candidate Transforms for each
       Search Engine (Section 3.1.3)
(4)  Output Best Transforms for each Search Engine
```

Figure 1: Outline of the process used to train the *Tritus* system.

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

Table 1: Question type phrases used for evaluation (Section 4).

3.1 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 1.

3.1.1 Selecting Question Phrases

In the first stage of the learning process (Step (1) in Figure 1) 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"*.

The input to this stage is a set of questions. These questions and their corresponding answers constitute the training data. 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 *minQphraseCount* 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 1.

This method of selecting question phrases can produce many phrases, which may include 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 the training for the results reported here to phrases that match the regular expressions shown in Figure 2. The regular expressions match common questions, and allow us to concentrate our resources on the most useful phrases. 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.

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 3.2), categorizing a question using phrase matching can be incorporated with negligible overhead in the overall processing time of queries.

```
^what (is|are|were|does|do|did|should|can)\s
^who (is|are|was|were|did|do|does)\s
^how (to|is|do|did|does|can|would|could|should)\s
^why (is|do|are|did|were|does)\s
^where (is|was|can|are|were|do|does)\s
^when (is|was|are|were|do|did|does)\s
^which\s
```

Figure 2: The regular expressions used to filter the automatically generated question phrases that are candidates for transformation.

3.1.2 Generating and Filtering Candidate Transforms

In the second stage of the learning algorithm (Step (2) in Figure 1) 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 1). *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. A sample of the original collection is given in Figure 5. 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 [4], which is widely used in the natural language processing community and is available at http://www.cs.jhu.edu/~brill/.

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 2. These phrases are heavily biased towards electronics or the computer domain. These phrases 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 may change the intended topic of a query. Recall that the transformations we are trying to learn should improve accuracy of the retrieved set of documents, yet preserve the topic of the original query. Therefore we would need to filter out phrases such as *"telephone"*, which intuitively would not be good transformations for general questions (e.g., *"What is a rainbow?"*).

We address this problem by filtering out candidate transform phrases containing nouns. We observe that in most of the queries the nouns are *content* words, or words expressing the topic of the query. For example, in the query *"what is a rainbow"*, the term *"rainbow"* is a noun and a content word. Likewise, the word *"telephone"* is a noun. Thus, we filter candidate transform phrases by checking if a generated answer phrase contains a noun, and if it does, the phrase is discarded. We use the part of speech information, which is computed once for the whole collection as described in the beginning of this subsection.

Of the resulting n-grams, we keep the *topKphrases* with the highest frequency counts. 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 [16]. 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 [18] 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 [17] uses a similar idea. In the original definition of BM25, each term t_i in the document is assigned the Robertson-Spark Jones term weight $w_i^{(1)}$ [15] with respect to a specific *query topic* and is calculated as:

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

Table 2: A sample of candidate transforms generated without disregarding phrases containing a noun.

$$w_i^{(1)} = \log \frac{(r + 0.5)/(R - r + 0.5)}{(n - r0.5)/(N - n - R + r + 0.5)} \quad (1)$$

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 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 non-relevant, and apply the formula in Equation 1.

We then compute the *term selection weights*, wtr_i, for each candidate transform tr_i, as described in [14] in the context of selecting terms for automatic query expansion as:

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

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 strategy 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 3 the phrase "named after" is a better discriminator for the question

Question Phrase	*Candidate Transform*	qtf_i	$w_i^{(1)}$	wtr_i
"what is a"	"refers to"	30	2.71	81.3
	"refers"	30	2.67	80.1
	"meets"	12	3.21	38.52
	"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

Table 3: Sample candidate transforms along with their frequency count qtf_i, BM25 term weight $w_i^{(1)}$, and the resulting term selection weight wtr_i.

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

Table 4: A sample of candidate transforms grouped into buckets according to the transform length. These transforms were generated for the question phrase *"what is a"*.

phrase *"what is a"*, it does not occur as often as those 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 3.

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 expect that longer phrases may be processed differently by the search engines, and this step was done in order to include such longer, potentially higher precision transforms in the set of candidate transforms, whereas primarily shorter transforms with higher frequency counts may be chosen otherwise. In Table 4, we show a sample of phrases with the highest selection weights from each candidate transform bucket.

3.1.3 Weighting and Re-ranking Transforms using Search Engines

In the third and final stage of training, we evaluate the performance of each candidate transform, tr_i, on web search engines. Figure 3 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. This is done by sorting all of the <*Question, Answer*> pairs in the collection in order of increasing answer length, and using up to *numExamples* of the first <*Question, Answer*> pairs that contain questions beginning with *QP*. The sorting step is done because we believe the evaluation may be more accurate for questions that have shorter answers; however, this is a topic for future research.

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 transform $\{tr_i\}$ to the *Question* one at a time (Step (2)). Consider a question *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 *Query* = $\{C \;\; \text{AND} \;\; tr_i\}$. 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 *Query* = " {(*lisp machine lispm*) AND (*"refers to"*)}."

We use the appropriate query syntax for each search engine. We also encode the transforms so that they are treated as *phrases* by each search engine.

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 [5] and may discard some words that appear in its stopword list.

There are other possibilities for rewriting *Question*, for example, requiring or not requiring parts of the query in matching pages, and combining multiple transformations into a single query.

In Step (3) of Figure 3 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 subdocuments in D most similar to *Answer* are 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*, i.e., that the key information/set of phrases will appear in close proximity of each other – within subdocuments of length *subDocLen*. The subdocuments overlap by *subDocLen*/2 words, 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 $D_1, D_2, D_3, D_4, ...$, 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 D_i in D. More formally, *docScore(Answer, D)* = *Max(BM25$_{phrase}$(Answer, D_i))* where *BM25$_{phrase}$* is an extension of the *BM25* metric [16] 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 as:

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

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 tokens, $avdl$ is the average document length in tokens, and $w_i^{(1)}$ and qtf_i are the relevance weight and query topic frequency as described previously.[1]

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

```
(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 = -1
(4a)            Subdocuments = getSubDocuments(Document, subDocLen)
                for each SubDocument in SubDocuments
(4b)               tmpScore = DocumentSimilarity(Answer, SubDocument)
                   if (tmpScore > docScore) docScore = tmpScore

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

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

Figure 3: Automatically evaluating the effectiveness of candidate transforms.

In the $BM25_{phrase}$ metric, the "terms" in the summation (Equation 3) include phrases, with weights learned over the training data as in the previous subsection. The weight for a term or phrase t is calculated as follows:

$$w = \begin{cases} w_t^{(1)} & \text{if } w_t^{(1)} \text{ is defined for } t \\ \log IDF(t) & \text{if } IDF(t) \text{ is defined for } t \\ \textit{NumTerms}(t) \cdot \sum_{t_i \in t} \log IDF(t_i) & \text{otherwise} \end{cases}$$

This multi-step assignment procedure is used because terms encountered may not be present in the training collection. We use *IDF* (Inverse Document Frequency, which is high for rare terms, and low for common terms) weights derived from a much larger sample (one million web pages, obtained from the collection of pages used in the TREC Web Track [9]). The last, fall-back case is necessary in order to handle phrases not present in the training data. Intuitively, it assigns the weight of phrase t inversely proportional 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 may be improved on with further work. While document ranking is important during run time operation of the system described in Section 3.2, re-ranking 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 non-relevant 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:

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

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 5.

3.2 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 4.

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

Table 5: Some of the top ranked transforms for the question phrase *"what is a"* automatically optimized for AltaVista and Google.

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 the" would be preferred over "what is". In Step (1b), the corresponding set of transforms is 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 (3) the transformed queries are submitted to the search engine and the first page of results is retrieved.

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 process of scoring the document is the same as described in Section *3.1.3*. The important difference is in Step (4c). If a document is retrieved through the application of multiple transforms, then the final score for the document is the maximum of each of the individual document scores.

In Step (5) the returned documents are ranked with respect to their final document scores, and in Step (6) the top ranked *topKdocs* documents are returned as the final result produced by the *Tritus* system.

4. EXPERIMENTAL SETTING

In this section we present the experimental setting that we use to evaluate the performance of *Tritus*. Section 4.1 details the training of *Tritus* for the evaluation. 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

```
(1a)    QP = matchQuestionPhrase(Question)
(1b)    {tr} = retrieveTransforms(QP, numTransforms)

        for each tr_i in {tr}
(2)        Query = ApplyTransform(Question, tr_i)
(3)        Results = SubmitQuery(Query, SE)
           for each Document in Results
              maxScore = Maximum Score for this Document so far
(4a)          Subdocuments = getSubDocuments(Document, subDocLen)
              for each SubDocument in SubDocuments
(4b)             Score = DocumentSimilarity(Query, SubDocument)
(4c)             if (Score > maxScore) Update Maximum Score of Document to Score

(5)     RankedDocuments = Sort Documents in decreasing order of document Score
(6)     Return topKdocuments with highest Score
```

Figure 4: Evaluating questions at run time.

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 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. ...

Figure 5: Sample question-answer pairs from the training collection.

Type	*Phrase(s)*	*Question-Answer Pairs in Collection*
Where	*"where can i"*	1408
	"where is"	198
What	*"what is"*	3486
	"what are"	1604
	"what is a"	486
How	*"how do i"*	2654
Who	*"who was"*	37

Table 6: The number of training questions for each question type.

Question (FAQ) files on various subjects. Figure 5 shows a sample of the question-answer pairs. We obtained the FAQ files from the FAQFinder project [6]. 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 6.

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 7. We did not test parameters exhaustively and further fine-tuning may improve the performance of the system.

4.2 Retrieval Systems Compared

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* when Tritus uses the corresponding search engine-specific transformations. We also evaluated AskJeeves for comparison. The five systems evaluated are:

- Google **(GO)**: The Google search engine as is.

Parameter	*Value*	*Description*
minQcount	30	Min. frequency for generating question phrases
minAcount	3	Min. frequency for generating candidate transforms
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

Table 7: *Tritus* training parameters.

- 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 3.2.
- 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 3.2.
- AskJeeves (**AJ**): The results of the AskJeeves [3] 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* [18]. Each document in the collection at hand is judged to be either *relevant* or *non-relevant* 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 non-relevant 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 [21].

Recently, TREC has incorporated a *Web Track* [9] 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, which operate over the web at large. Consequently, to evaluate *Tritus* rigorously we needed to define an alternative experimental setting.

To evaluate *Tritus* we inspect the top 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 following metrics.

DEFINITION 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 the percentage of questions where a given system provides the best performance of all systems tested:

DEFINITION 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.*

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.*

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 8). 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 [20].

Type	*Phrase(s)*	*Total*	*Presented to judges*	*Evaluated*
Total queries		2,477,283		
Where	*"where"*	184,634		
	"where can i"	162,929	46	10
	"where is"	7,130	47	11
What	*"what"*	69,166		
	"what is"	35,290	45	11
	"what are"	10,129	43	12
	"what is a"	5,170	48	15
How	*"how"*	20,777		
	"how do i"	13,790	44	15
Who	*"who"*	16,302		
	"who was"	1,862	40	15
Total questions		290,000 (approx.)	313	89

Table 8: The number of questions of each question type in the Excite query log.

Parameter	*Value*	*Description*
numTransforms	15	Max. number of transforms to apply
subDocLen	50	Size of the sub-document used for re-ranking
maxPhraseLen	4	Max. length of query phrases for re-ranking
Timeout	30	Individual page timeout

Table 9: Parameters used for evaluation.

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 of each type (Table 8) are submitted to all of the five 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 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 9.
- **TR-GO**: The top 10 documents returned by *Tritus* using transforms optimized for Google and the parameters in Table 9.
- **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 pre-selected value of a term (or terms) in the question and an associated "submit" button, which, if selected, will take the user to an answer.
3. Pages that other users found useful for the given question (or a similar question).

We retrieved all of the results in (1) and (2) and the top results from (3) (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 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.

Figure 6 shows the average *precision* at K for varying K of AskJeeves **(AJ)**, AltaVista **(AV)**, Google **(GO)**, Tritus-Google **(TR-GO)**, and Tritus-AltaVista **(TR-AV)** over the 89 test questions. As we can see, *Tritus* optimized for Google has the highest precision at all values of document cutoff K. Also note that both **TR-GO** and **TR-AV** perform better than the underlying search engine used. *Tritus-AV* shows a large improvement over AltaVista.

Figure 7 shows the percentage of questions where each system performed the best in terms of the number of relevant documents returned at document cutoff 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.

We 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 response to the original query as more documents are examined in Figure 8. In Figure 8(a) we report the percentage of all *Tritus* documents (without re-ranking) that are contained in the top N (10-150) search-engine responses for the original query. In Figure 8(b) we plot the percentage of top 10 *Tritus* documents (after reranking) contained in the original search engine results, and in Figure 8(c) we show percentage of relevant *Tritus* documents contained in the original search engine results. Only the top 10 *Tritus* documents had relevance judgements 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 re-ranking), but slowly increases and levels off at around 45% for the top 10 and relevant *Tritus* documents as more of the original Google 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.

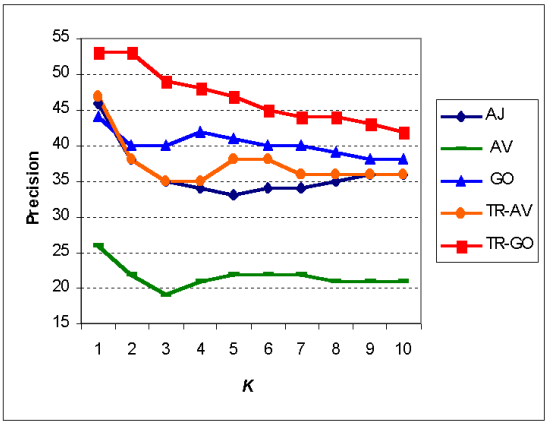

Figure 6: Average *precision at* K of AskJeeves (AJ), AltaVista (AV), Google (GO) and Tritus (TR-AV, TR-GO) over 89 test queries, for varying number of top documents examined K. Tritus consistently outperforms the underlying search engine that it is based on, and Tritus-Google is the best performing system.

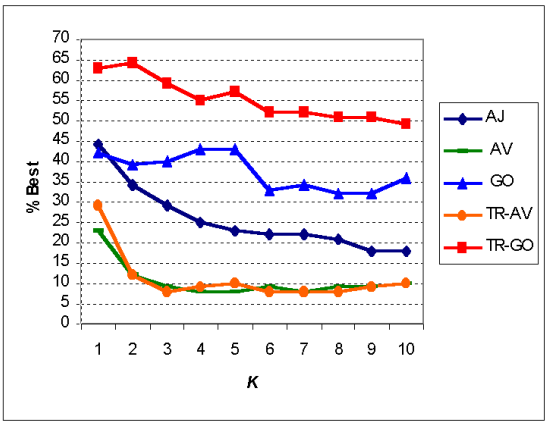

Figure 7: The percentage of questions for which each system returns the most relevant documents for varying number of top documents examined K. The search engines are AskJeeves (AJ), AltaVista (AV), Google (GO) and Tritus (TR-AV, TR-GO) over 89 test queries. When the highest number of relevant documents for a given question is returned by multiple search engines, all of these engines are considered to have returned the most relevant documents. 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.

In Table 10 we report the average precision (%) at K of the document sets retrieved by **AJ**, **AV**, **GO**, **TR-GO**, and **TR-AV**. The results are separated by question phrase. As we examine documents with lower ranks, precision in general decreases. In the case of *AskJeeves*, the reason may be that the top-ranked documents could be selected by human editors as answers to the questions. Note that at least one of the *Tritus* systems has higher precision than AskJeeves even for the top ranked document alone in 6 out of 7 question phrases.

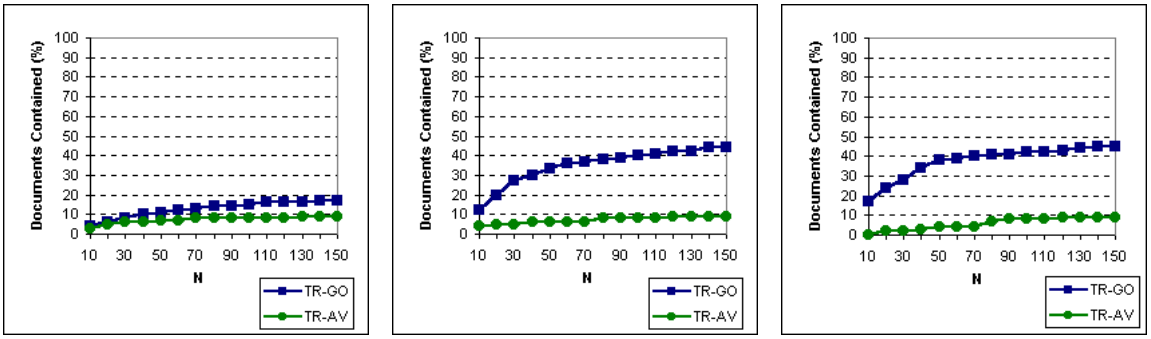

Figure 8: Average percentage of all (a), top 10 (b) and relevant in top 10 (c) Tritus (TR-GO and TR-AV) documents contained in top N documents returned for each original query by the underlying search engine.

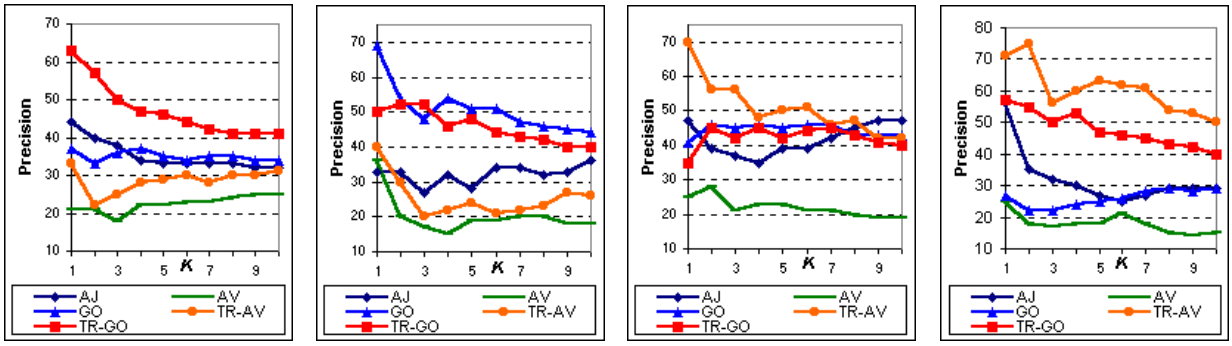

Figure 9: Average *precision at* K of AskJeeves (AJ), AltaVista (AV), Google (GO) and Tritus (TR-AV, TR-GO) by question type: *What* (a), *How* (b), *Where* (c), and *Who* (d) .

In Figure 9 we report the average precision (%) at K of the document sets retrieved by AskJeeves **(AJ)**, AltaVista **(AV)**, Google **(GO)**, Tritus-Google **(TR-GO)**, and Tritus-AltaVista **(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 9(a), (c) and (d)) for $K \leq 5$, while Google has the highest precision for *How* [2].

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, which 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.

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* questions 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.

[2]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 webpage?"* can take many different forms, from a direction *"Click here to create your free webpage"*, 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.

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.

6. FUTURE WORK AND SUMMARY

There are many avenues 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. Multiple search engines could be used simultaneously. We plan to investigate creating phrase transforms that contain content words from the questions. We also plan to make the transformation process dynamic. For example, transformations where we expect high precision may be submitted first. Based on the responses received, the system may 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 answers to questions from an information retrieval system. The method involves classifying questions into different question types, generating candidate query transformations from a training set of question/answer

Question Type	Question Phrase	K	System Precision (%)				
			AJ	*AV*	*GO*	*TR-AV*	*TR-GO*
What	*"what is a"*	1	64	9	40	25	57
		2	56	14	38	17	54
		3	53	15	41	21	48
		5	48	18	44	23	46
		10	41	25	37	27	41
	"what are"	1	22	50	40	25	67
		2	32	30	25	25	48
		3	25	21	27	33	45
		5	20	26	23	45	45
		10	24	26	33	49	40
	''what is"	1	45	11	44	43	64
		2	29	26	50	29	68
		3	30	25	53	25	55
		5	27	26	52	30	49
		10	26	24	40	27	43
How	*"how do i"*	1	33	36	69	40	50
		2	33	20	54	30	52
		3	27	17	48	20	52
		5	28	19	51	24	48
		10	36	18	44	26	40
Where	*"where can i"*	1	57	25	63	100	60
		2	53	31	65	67	65
		3	41	31	65	71	63
		5	49	36	65	67	66
		10	59	34	64	53	65
	"where is"	1	44	25	22	57	18
		2	30	21	32	50	32
		3	34	13	31	50	27
		5	32	14	33	45	25
		10	39	7	28	39	23
Who	*''who was"*	1	56	25	27	71	57
		2	35	18	22	75	55
		3	32	17	22	56	50
		5	27	18	25	63	47
		10	29	15	29	50	40

Table 10: Average *precision* at K for AskJeeves (AJ), AltaVista (AV), Google (GO) and Tritus (TR-AV, TR-GO) separated by question type and question phrase, for K = 1, 2, 3, 5 and 10 documents.

pairs, and evaluating the candidate transforms on the target information retrieval systems. This technique for processing natural language questions could be applicable to a wide range of information retrieval systems. We have implemented and evaluated the method as applied to web search engines. Blind evaluation on a set of real queries from a web search engine shows that the method significantly outperforms the underlying web search engines for common question types.

Acknowledgements

The first author acknowledges NEC Research Institute where the bulk of the research was accomplished, and the support from National Science Foundation under Grant No. IIS-9733880. We thank Robin Burke for graciously giving us the training collection of FAQ files in semi-structured format and Nick Craswell for giving us the document frequency counts collected over 1 million web pages. We also thank Eric Glover for his help throughout the project, and Regina Barzilay, Min-Yen Kan, Kathy McKeown, Kazi Zaman and the anonymous referees for helpful comments.

Finally we would like to thank all the great people who took time off from their busy schedules to help evaluate our system: Regina Barzilay, Nicolas Bruno, Adiel Cohen, Junyan Ding, Noemie Elhadad, Rakan El-Khalil, Eleazar Eskin, David Evans, John Gassner, Eric Glover, Tobias Hoellerer, Panagiotis Ipeirotis, Min-Yen Kan, Sharon Kroo, Amelie Marian, Eugene Mesgar, Yoko Nitta, Gedalia Pasternak, Dmitry Rubanovich, Carl Sable, Roman Shimanovich, Alexey Smirnov, Viktoria Sokolova, Michael Veynberg, Brian Whitman, Kazi Zaman, and an anonymous benefactor.

7. REFERENCES

[1] S. Abney, M. Collins, and A. Singhal. Answer extraction. In *ANLP-2000*, 2000.

[2] D. Aliod, J. Berri, and M. Hess. 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)*, 1998.

[3] AskJeeves. http://www.askjeeves.com .

[4] E. Brill. A simple rule-based part of speech tagger. In *Third Conference on Applied Natural Language Processing (ANLP-92)*, 1992.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *7th International World Wide Web Conference (WWW-7)*, 1998.

[6] R. Burke, K. Hammond, and J. Kozlovsky. Knowledge-based information retrieval for semi-structured text. In *AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval*, pages 19–24, 1995.

[7] C. Cardie, V. Ng, D. Pierce, and C. Buckley. Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering system. In *ANLP-2000*, 2000.

[8] S. M. Harabagiu, M. A. Pasca, and S. J. Maiorano. Experiments with open-domain textual question answering. In *COLING-2000*, 2000.

[9] D. Hawking, N. Craswell, P. Thistlewaite, and D. Harman. Results and challenges in web search evaluation. In *8th International World Wide Web Conference (WWW-8)*, 1999.

[10] H. Joho and M. Sanderson. Retrieving descriptive phrases from large amounts of free text. In *Proceedings of CIKM 2000*, 2000.

[11] S. Lawrence and C. L. Giles. Context and page analysis for improved web search. *IEEE Internet Computing*, 2(4):38–46, 1998.

[12] M. Mitra, A. Singhal, and C. Buckley. Improving automatic query expansion. In *ACM SIGIR*, 1998.

[13] D. Moldovan, S. Harabagiu, M. Pasca, R. Mihalcea, R. Goodrum, R. Girju, and V. Rus. Lasso: A tool for surfing the answer net. In *Proceedings of TREC-8*, 1999.

[14] S. Robertson. On term selection for query expansion. In *Journal of Documentation*, volume 46, pages 359–364, 1990.

[15] S. Robertson and K. S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:129–146F, 1976.

[16] S. Robertson and S. Walker. On relevance weights with little relevance information. In *SIGIR 97*, 1997.

[17] S. Robertson, S. Walker, and M. Beaulieu. Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive track. In *TREC-7 Proceedings*, 1998.

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

[19] B. Schiffman and K. R. McKeown. Experiments in automated lexicon building for text searching. In *COLING-2000*, 2000.

[20] A. Spink, S. Milchak, M. Sollenberger, and A. Hurson. Elicitation queries to the Excite web search engine. In *CIKM 2000*, 2000.

[21] E. Voorhees. Overview of the Eighth Text REtrieval Conference (TREC-8). In *Proceedings of TREC-8*, 1999.

[22] E. Voorhees. The TREC-8 question answering track report. In *Proceedings of TREC-8*, 1999.

[23] J. Xu and W. B. Croft. Improving the effectiveness of information retrieval with local context analysis. *ACM Transactions on Information Systems (TOIS)*, 18(1):79–112, 2000.