Why systems need knowledge to find what you really want

WILLIAM A. WOODS,
SUN MICROSYSTEMS LABORATORIES

Searching vs. Finding

Finding information and organizing it so that it can be found are two key aspects of any company's knowledge management strategy. Nearly everyone is familiar with the experience of searching with a Web search engine and using a search interface to search a particular Web site once you get there. (You may have even noticed that the latter often doesn't work as well as the former.) After you have a list of hits, you typically spend a significant amount of time following links, waiting for pages to download, reading through a page to see if it has what you want, deciding that it doesn't, backing up to try another link, deciding to try another way to phrase your request, et cetera. Eventually you may find what you want, or you may ultimately give up and decide that you can't find it. Why is this so difficult?

I have been asking myself this question since I was a graduate student and took my first course in information retrieval. I was appalled to discover what information retrieval systems actually did. I expected them to understand what I was asking for and to find documents that were about that topic. What they did was count words and push the numbers through an equation to compute a ranking.

Searching vs. Finding

Since then, information retrieval researchers have explored many techniques, but modern document retrieval systems still tend to be blunt instruments, retrieving many nonrelevant documents (errors in precision) and missing many relevant documents (errors in recall). The user is left with a significant task of reading or scanning the retrieved results to determine whether they actually have the information sought and to figure out whether and how to rephrase a request to see if any relevant documents were missed.

This article describes what I have learned from years of thinking about these problems and from a research project at Sun Microsystems Laboratories that combines the respective strengths of humans and computers in a knowledge-based system to help people find information.

I have been trying to develop systems that come closer to my original assumption, focusing on helping people find specific information, not just documents. I've found some interesting approaches, using some linguistic and cognitive science insights, several kinds of knowledge, and some new algorithms. These techniques require more computation than do traditional techniques, but if a search engine can be made more helpful by spending more time being smarter, then a decrease in the speed of the search engine can be more than offset by the increased speed of finding what you want.

One of the problems that makes searching difficult is that people often ask for information using different terms from those used in what they need to find. Researchers have explored a variety of techniques for addressing this problem, some of which use various kinds of knowledge. I have been trying to understand what kinds of knowledge are necessary to make connections between what you ask for and what you want. I started studying this problem by catching people in what I call an "information seeking state" and writing down, in their own words, what they said they were looking for. I then tried to capture what they ultimately found and do a linguistic analysis of the relationships between the request and the result. Two important kinds of relationships were apparent: morphological and semantic (about which I will say more directly).

PROBLEM 1: MORPHOLOGICAL RELATIONSHIPS

Early in my career, I built a question-answering system for the NASA Manned Spacecraft Center to answer questions about the Apollo 11 moon rocks. In addition to answering English questions such as, "What is the average concentration of silicon in high-alkali rocks?" (which it did by understanding the structure of the question and then computing the answer), this system included cross-references to the articles from which its data had been extracted and a key-phrase search capability for searching those articles. The key-phrases were extracted from the texts by an automatic extraction technology developed by the Defense Documentation Center. One problem we encountered was that there was an article indexed under the phrase *acid glass* and another article indexed under *acidic glass*, and the system had no idea that these phrases and these articles had anything to do with each other.

Although the **stemming technique is elegant** and has a simple matching criterion, it has several problems.

The relationship between *acid* and *acidic* is morphological (morphology being the linguistic study of how words are formed), because the word *acidic* is derived from the word *acid* by adding the suffix *ic*. Information retrieval systems often deal with morphological relationships between terms by using a technique called stemming, which consists of removing recognized suffixes from a word (perhaps repeatedly) until you are left with a residual, or stem. In this case, removing *ic* from *acidic* results in the stem *acid*.

The idea is to index documents by the stems of the words they contain and to search using only the stems of the words in a query. Thus, all of the words in both the document and the query are reduced to a standard form (the stem), which can be matched directly. The user gets hits for all words that are morphologically related to the query terms, while the system has a simple matching criterion. In the previous example, the phrase *acidic glass* would be standardized to the same stem as *acid glass*, and both documents would be indexed under the same phrase.

Limitations of Stemming. Although the stemming technique is elegant and has a simple matching criterion, it has several problems—one of which is that stemming sometimes makes mistakes. It can reduce unrelated words to the same stem, and it can fail to reduce related words to a common stem. For example, *computing* reduces to *comput*, which is different from the word *compute*, so the stemmer treats the final *e* as if it were a suffix and removes it from *compute* to get the common stem *comput*. But sometimes, final *e*'s matter, as in *cap* versus *cape*, so the stemmer has to decide whether to remove it or not. Without knowledge of what words exist, a stemmer will inevitably get some of these cases wrong. In one stemmer that I tested, the words *copper*, *cop*, *cope*, and *copulate* all reduced to the stem *cop*. Another one considered *uncapable* and *uncapped* to have the same stem *uncap*.

Most stemmers are heuristic efforts, falling short of a full understanding of the morphology of the language. For example, in the previous examples, *copper* could not come from *cope*, since only a final letter would be doubled, and *uncapable* could not come from *uncap* since the final *p* would be obligatorily doubled before adding *able*. Moreover, stemmers generally deal only with suffixes, so the correct analysis of "un++capable" is not available. Dealing with prefixes, as well as suffixes, is much more complicated and more prone to false analyses, since it requires resolving choices about which affix is applied first (e.g., *un++capable* versus *uncap++able*).

Even when correct, stemming loses information, eliminating the user's ability to discriminate among different forms of a word. If *acidic glass* means something slightly different from *acid glass*, the user is blocked from expressing this difference. A user who wants to ask about *subjectivity* will be forced to deal with hits on *subjects*, *subjected*, and *subjection* as well.

Lexicon-based Morphology. Robert Krovetz provides an excellent discussion of stemming and argues the merits of using a dictionary as an additional source of knowledge.[1] Krovetz implemented a modified stemmer that used a machine-readable English dictionary to restore proper endings to words and to stop removing suffixes when it had produced a word that was in the dictionary. He explored several variations on this idea and showed that this method generally exceeded the performance of a standard stemmer. His modified stemmers used a commercial dictionary of 27,855 words and analyzed 106 suffixes.

I have also been working with a morphological engine[2] that analyzes prefixes and suffixes, as well as lexical compounds (such as *bitmap* and *replybuffer*). It uses a lexicon developed for natural language parsing, and it automatically constructs new lexical entries for unknown words. These entries, like those for known words, capture and represent information about a word, such as its syntactic categories (noun, verb, adjective, etc.), its morphological structure, and its relationships to other words. The lexicon is used to handle exceptions and to test whether hypothesized

TABLE 1 Fifteen reasons why you need to know particular words (false morphological analyses of ordinary words)

Word	Morphology	Analysis
caress	(car + ess)	female car
cashier	(cashy + er)	wealthier
lacerate	(lace + rate)	speed of needlework
marinate	(marine + ate)	make into a marine
phony	(phone + y)	always on the telephone
rehearse	(re + hearse)	put the coffin back in the van
daredevil	(dared + evil)	serious risk
copout	(co + pout)	sulking together
detergent	(deter + gent)	a discouraging gentleman
pigeon	(pig + eon)	the age of peccaries
infantry	(infant + ry)	childish behavior
pantry	(pant + ry)	heavy breathing
molestation	(mole + station)	depot for moles
contractor	(con + tractor)	prison vehicle
extractor	(ex + tractor)	former tractor

Searching vs. Finding

base forms satisfy syntactic (and sometimes semantic) conditions for a rule to apply. The lexicon is also used by parsing algorithms to analyze phrases and sentences, and it is used as a source of semantic and morphological relationships between words.

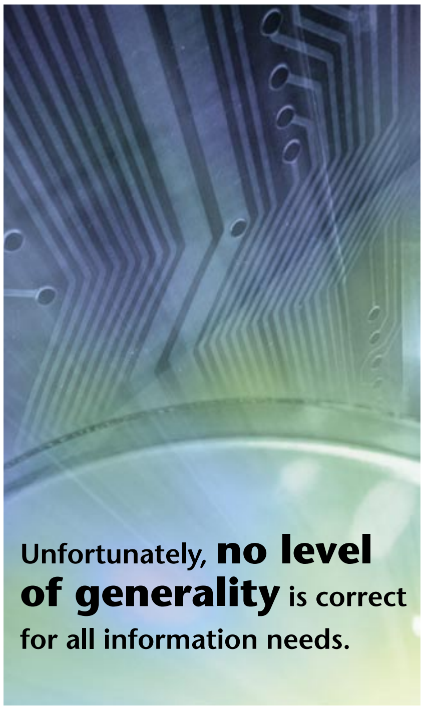
This morphology engine currently has 1,724 knowledge-based rules that analyze 690 prefixes and 276 suffixes. Its rules capture many subtleties of English morphology, including such basics as doubling final letters and inserting final *e*'s, and it has heuristic criteria for selecting preferred analyses of a word from the sometimes many variations that would be linguistically possible. An effective lexicon of more than 250,000 word forms is automatically generated from a core of 40,000 entries by applying the morphology engine to a list of known words and several lists of proper names.

One reason that a dictionary of known words is important is that without it, many ordinary words would be incorrectly analyzed as being morphologically derived from other unrelated words. For example, *delegate* does not mean to take the legs off something (de+leg+ate), and *ratify* does not mean to infest with rodents (rat+ify). Table 1 illustrates some of the more interesting cases from among thousands of ordinary words that could receive false morphological analyses if the dictionary didn't already know what they meant.

PROBLEM 2: SEMANTIC RELATIONSHIPS

Like morphological relationships, semantic relationships may be necessary to make a connection between what you ask for and what you need. Semantic relationships have to do with how the meanings of words are related. For example, a system would need to know the semantic relationship between *moon* and *lunar* to retrieve a document indexed under *lunar rocks* in response to a request for *moon rocks*. Semantic relationships are often addressed in information retrieval systems by means of a synonym thesaurus. In a synonym thesaurus, words are divided into "synonym sets," each of which contains words that have the same meaning. These synonym sets can be used to expand a query by adding all of the synonyms of the query terms, so documents that involve those synonyms will also be found. For example, *moon* and *lunar* could be placed in a synonym set so that queries using the term *moon* would be expanded with the term *lunar*, and vice versa.

Synonyms Are Not Enough. Attempts to expand queries automatically using a synonym thesaurus often fail to improve the effectiveness of retrieval systems and frequently degrade results. Part of the problem is that few true synonyms exist in English (or any other language), and members of synonym sets in such thesauri often differ significantly in meaning. For example, {*automobile, car, truck, bus, taxi, motor vehicle*} might be grouped as synonyms in such a thesaurus. If your query is for *motor vehicle*, then expansion with this set could give useful results, but if your query is for *cars*, you might not be happy getting hits on *trucks* and *buses*. The problem is that some of these terms are more general than others. Choosing to treat such terms as synonyms amounts to generalizing the query to a level of abstraction where the differences don't matter. Unfortunately, no level of generality is correct for all information needs.


Unfortunately, **no level of generality** is correct for all information needs.

Generality and Subsumption. A better system would be one that captures and exploits generality relationships, so that a user can ask questions at whatever level of generality is desired. I have been exploring the use of a kind of subsumption technology[3] to address this problem. The idea is that general terms subsume specific ones, and terms can be organized into a structured conceptual taxonomy based on these subsumption relationships. Formally, a term subsumes itself, any more specific terms, and any true synonyms that it may have. When searching, a term in a request will match any term in a target that is subsumed by the requested term. Thus, a request for *motor vehicle* would retrieve all kinds of motor vehicles, whereas a request for *automobile* would retrieve *cars* and *taxis* but not *trucks* and *buses*. You can also treat root words as subsuming their derived and inflected forms, so that *car* would subsume *cars*. In this way, one can combine semantic and morphological relationships in a uniform, intuitive framework.

Conceptual Indexing. To explore the hypothesis that subsumption technology could improve the effectiveness of online search, I built a system to extract words and phrases from text and automatically assimilate them into a structured conceptual taxonomy organized by the subsumption relationship. The resulting structure, which I call a conceptual index, turned out to be an intuitive structure for people to browse, and it reveals many interesting relationships between words and phrases that occur in indexed material and in queries. For example, when searching a business directory for *automobile cleaning*, it found a relationship to *car washing*, because it knew from its lexicon that a *car* was a kind of *automobile* and that *washing* was a kind of *cleaning*. It was able to infer that each part of the phrase *automobile cleaning* subsumed a corresponding part of the phrase *car washing*, and therefore the former concept subsumed the latter.

Viewing the Conceptual Taxonomy. I have noticed that people normally ask questions at the level of generality that they are interested in, without even thinking about it. Hence, it is usually a bad idea to generalize a user's query automatically. Occasionally, however, someone will miss a generalization, and in such cases, being able to see where a query term fits in the conceptual taxonomy can be highly useful. For example, when I asked for *brown fur* in a collection of articles about animals, I found only three subsumed phrases, but I saw that my request had been classified under *brown coat* in the taxonomy. Figure 1 shows the highly relevant concepts that I found by generalizing my request to *brown coat*. Without seeing the conceptual taxonomy, I might not have thought of this generalization and would have missed a lot of what I wanted.

Specific Passage Retrieval. Experimenting with conceptual indexing, I observed that sometimes words in a relevant passage of text are not related in a single phrase that would be subsumed by the query. For example, the *brown coat* query didn't find an article containing the sentence, "The coat is reddish brown," because this sentence doesn't contain an explicit phrase that is subsumed by *brown coat*. Sometimes the relevant information is spread over several sentences.

Fragment of a Conceptual Taxonomy

```
(BROWN COAT)
    ├k—(BRIGHT REDDISH BROWN COAT)
    ├k—(BROWN BLACK COAT)
    ├k—(BROWN COATS)
    │       ├k—(FAWN COATS)
    │       │       ├v—((FAWN) COATS)
    │       │
    │       ├k—(REDDISH BROWN UPPER COATS)
    │
    ├k—(BROWN FUR)
    │       ├k—(GRAY BROWN FUR)
    │       ├k—(RICH BROWN FUR)
    │       ├k—(WHITE-SPOTTED BROWN FUR)
    │
    ├k—(BROWN HAIR)
    │       ├k—(BROWN HAIRS)
    │       │       ├k—(REDDISH BROWN GUARD HAIRS)
    │       │
    │       ├k—(BROWN WOOL)
    │       │       ├k—(REDDISH BROWN WOOL)
    │       │
    │       ├k—(BROWNISH HAIR)
    │       ├k—(REDDISH BROWN HAIR)
    │               ├k—(REDDISH BROWN GUARD HAIRS)
    │               ├k—(REDDISH BROWN WOOL)
    │
    ├k—(BROWN-GRAY COAT)
    ├k—(BROWNISH COAT)
    │       ├k—(BROWNISH HAIR)
    │       ├k—(BROWNISH SUMMER COAT)
    │       ├k—(COARSER BROWNISH COAT)
    │       ├k—(DARK BROWNISH COAT)
    │
    ├k—(COAT OF BROWN)
    │       ├k—(WOOLLY COAT OF REDDISH BROWN)
    ├k—(TAWNY COAT)
```

FIG 1

Searching vs. Finding

To deal with such cases, I extended my system with an ability to find passages where all (or almost all) of the elements of a request occurred near each other and nearly in the same relationships. Information from the conceptual taxonomy about where concepts occurred in documents was used to determine where such passages existed, and they were ranked by how nearly they approximated the input request. This technique, which I called specific passage retrieval, turned out to be especially effective for helping people find information. Rather than merely finding documents, it was able to find and display the passages in the document that were most likely to contain the information sought.

The specific-passage-retrieval algorithm ranks passages by a penalty score that it computes from a number of factors expressing ways that a relevant passage can differ from an input request. As the words in the passage get farther apart, the likelihood that they are related in the desired way decreases, so a penalty is computed that is proportional to the number of intervening words. Similarly, if the words in the passage are in a different order than those in the query, then a penalty is computed proportional to the amount of word reordering present. If a term in the passage is morphologically different from the corresponding term in the query or is a semantically more specific term, then a small penalty is added to induce a small preference for exact matches. If one of the terms of the query has no match in a passage, then the passage receives a significant penalty that can depend on the kind of term that is missing.

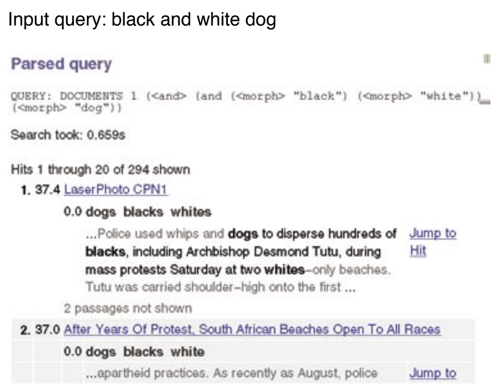
I picked coefficients for these factors that seemed to make sense and found that the resulting penalty-based scores were highly discriminating, so that the most relevant passages really did tend to get ranked first. After I applied the system to dozens of different subject domains with continued good results, I began to trust those initial guesses more than I would have trusted values that were tuned to a collection.

Figures 2 and 3 illustrate the advantages of the specific-passage-retrieval algorithm over traditional document-retrieval techniques. These figures show the output from an advanced search engine developed by my colleague, Stephen Green, a productized version of which is now incorporated into Sun's Portal and Web server products. The collection consists of several gigabytes of news articles. The engine supports multiple query operators and will apply a default operator when no operator is specified. In figure 2, the default operator is a traditional weighted Boolean AND operator. In figure 3, the default operator is an implementation of our penalty-based passage-retrieval algorithm. In both cases, an adaptation of our passage-retrieval algorithm is used to generate the summary that is shown with each hit. This enables a user quickly to determine when a hit is irrelevant and skip over it in the list.

In the Boolean AND case, you can see that the query *black and white dog* returns a document containing all of the requested terms, but they are not related in a way that has anything to do with black and white dogs. With the passage operator, on the other hand, the first hit contains an exact match of the phrase *black and white dog* and is directly on point. The second hit, which is not relevant, has a penalty because the term *dogs* is plural, separated from the phrase *black and white*, and is out of order.

This pair of examples illustrates how strongly the penalty-based ranking differentiates among near misses and how the traditional approach is insensitive to the relationship between the words. Both hits found by the

Weighted Boolean AND Query



Input query: black and white dog

Parsed query

```
QUERY: DOCUMENTS 1 (<and> (and (<morph> "black") (<morph> "white"))
(<morph> "dog"))
```

Search took: 0.659s

Hits 1 through 20 of 294 shown

1. 37.4 LaserPhoto CPN1
   0.0 **dogs blacks whites**
   ...Police used whips and **dogs to disperse hundreds of blacks, including Archbishop Desmond Tutu, during mass protests Saturday at two whites**–only beaches. Tutu was carried shoulder–high onto the first ... Jump to Hit
   2 passages not shown
2. 37.0 After Years Of Protest, South African Beaches Open To All Races
   0.0 **dogs blacks white**
   ...apartheid practices. As recently as August, police Jump to

FIG 2

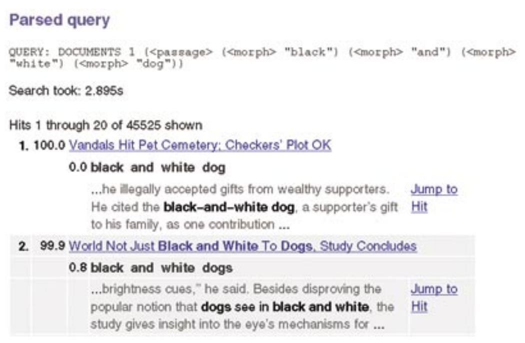
passage operator would be in the list of hits from the AND operator, but they are far down in the list and are not visible in the top ten choices. This is because of the way that the traditional approach assigns weights to terms in computing its rankings.

Knowledge and Search. Historically, many attempts to use natural language processing to improve information retrieval have either made little difference or actually made things worse. This has been observed for morphological and semantic expansion of queries and also for part-of-speech disambiguation of words. A fundamental problem is that techniques such as semantic expansion that have the potential to improve recall (the proportion of relevant documents that are retrieved) also tend to reduce precision (the proportion of retrieved documents that are relevant), and techniques aimed at improving precision tend to reduce recall.

Significantly, in one of my group's early experiments, comparing the specific-passage-retrieval algorithm with traditional document retrieval, we found that adding semantic knowledge and doing morphological analyses of unknown words with the penalty-based passage-retrieval method improved results, whereas incorporating some of the same information into the synonym thesaurus of a commercial search engine made things worse. Semantic expansion from the synonym thesaurus found a few additional relevant documents, but it found a much larger number of irrelevant documents that pushed good hits out of the top ten positions. The penalty-based method, because of its more discriminating ranking, seemed able to benefit from the additional recall without losing precision.

Penalty-based PASSAGE Query

Input query: black and white dog



Parsed query

```
QUERY: DOCUMENTS 1 (<passage> (<morph> "black") (<morph> "and") (<morph>
"white") (<morph> "dog"))
```

Search took: 2.895s

Hits 1 through 20 of 45525 shown

1. 100.0 Vandals Hit Pet Cemetery; Checkers' Plot OK

0.0 **black and white dog**

...he illegally accepted gifts from wealthy supporters. He cited the **black-and-white dog**, a supporter's gift to his family, as one contribution ... Jump to Hit

2. 99.9 World Not Just Black and White To Dogs, Study Concludes

0.8 **black and white dogs**

...brightness cues," he said. Besides disproving the popular notion that **dogs see in black and white**, the study gives insight into the eye's mechanisms for ... Jump to Hit

FIG 3

For example, with semantic expansion and the PASSAGE operator for the *black and white dog* query, the system found the phrase *black and white mongrel* as the second hit (because the lexicon knew that a *mongrel* is a kind of *dog*). Doing the same thing with the AND operator produced no relevant hits in the top ten choices. Several experiments have now shown that with this passage-retrieval method, adding knowledge improves results.[4]

A few researchers have explored passage-retrieval methods in the past, but these were usually based on segmenting the material into paragraphs or sentences and then indexing and searching those passages as if they were small documents. Unlike these earlier systems, the specific-passage-retrieval method I have been exploring identifies passages dynamically in response to input queries. The size of a passage depends on the query and the quality of the hit. Generally, for a given query, passages get longer and less relevant as you go down the list of hits, and you can stop looking when the penalty gets sufficiently high and the hit passages stop being relevant.

Strengths and Weaknesses of Different Methods. Our first experiment comparing specific passage retrieval with traditional document retrieval showed that the passage-retrieval algorithm, without any morphological or semantic knowledge, was roughly comparable to a commercial search engine in terms of the number of relevant documents found in the top ten choices. Each method found relevant documents that the other did not.

Intuitively, the penalty-based method found better hits when the relationship among the terms in the query was important or when some specific information within the document was sought. The traditional technique found more relevant documents when the mere occurrence of the query terms in the document was sufficient and the topic of the document as a whole was at issue. Both techniques have their uses, depending on the nature of the information need.

We found that the penalty-based technique was good for short queries and short targets such as document titles, chapter headings, and section headings, where traditional methods do poorly. Traditional word-counting techniques typically require targets of at least paragraph length to get traction and have been shown to work best if they can get paragraph-size queries as well.

Superficially, specific passage retrieval resembles phrase matching and proximity matching, but it subsumes both

Searching vs. Finding

of these and does more. It's as if the system automatically asked your query in a variety of different ways—including an exact phrase, reordered phrase, ordered proximity, unordered proximity, generalization by dropping words, substituting morphological variants, and substituting semantically related terms. It automatically finds the results of all of these, and ranks them so that the best come first.

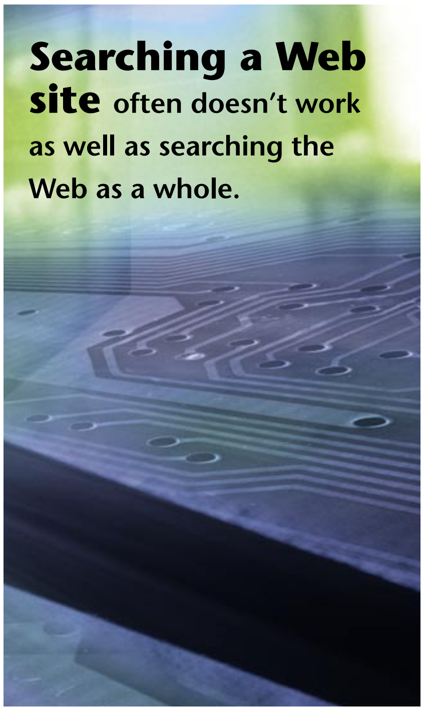
If you don't have a passage-retrieval operator, then you can use proximity and phrase operators to get some of the same benefits, but with a less useful ranking and with more thought and effort spent phrasing and rephrasing your query. Often the passage operator finds useful hits that would have been missed by using a more specific operator. For example, using a phrase search for a name like *William Woods* would avoid documents where these two terms were not adjacent, but it would miss occurrences such as *William A. Woods*.

The passage operator also helps with problems of word-sense ambiguity, since when the terms in a passage are related in the text as they are in the query, they are also more likely to be used in the same sense. For example, searching for *bill woods* with a passage operator is more likely to find *bill* as a name (rather than an invoice) when the passage has a low penalty (e.g., few or no intervening words).

Searching the Web versus Searching a Web Site. As I mentioned earlier, searching a Web site often doesn't work as well as searching the Web as a whole. There are several reasons for this. For one thing, the Web is so vast that almost everything is out there, expressed in almost every possible way, so your request is more likely to find a direct match. Almost any way of expressing a query will find thousands of hits. For a Web search engine, the important thing is to choose a small number of those hits to display and to do the query processing as quickly and cheaply as possible. Paraphrase support is not usually considered important, and it takes extra time and effort.

On the other hand, when searching a Web site or a corporate knowledge base, what you need to find may be worded in a particular way, and morphological and semantic paraphrase support may be necessary to find it. For example, in an experimental version of our search engine, indexing the Sun Labs Web site, the query *clockless design* retrieved passages involving *asynchronous design*. The conceptual index revealed that only two documents on the site mentioned *clockless*, and 142 mentioned *asynchronous*. If you were looking for this information without benefit of paraphrase support, you might go away without knowing that you had missed most of what was there.

It would be possible, in principle, to apply the same kinds of semantic and morphological expansions to the entire Web, using the specific-passage-retrieval technique, but that has not been my primary target. The Web is so vast that it is difficult to predict what would happen without trying it. There would probably be more issues with word sense ambiguity, and a global conceptual taxonomy would be awe inspiring. It would be an interesting challenge. Certainly the cost would be greater than for current Web search engines and might not fit their business models.


Searching a Web site often doesn't work as well as searching the Web as a whole.

The specific-passage-retrieval algorithm lends itself to applications of large scale, because it allows a collection to be subdivided and the search to be distributed, with the results easily collated (because the penalty scores are independent of collection statistics). In theory, this could be used for a kind of federated Web search in which owners of content could provide their own indexing and search and could update their indexes whenever the content changed. This would address a fundamental problem of Web searching: the never-ending task of repeatedly crawling the Web, trying to keep the indexes current.

It is interesting to contemplate a federation knit together by a spanning network of systems (possibly a peer-to-peer network) that distribute queries and collate the results. Some of the members of the federation could be large content providers who index their own content, whereas others could be crawler-based services like current Web search engines. Of course, this would take a heretofore untold amount of cooperation among many players that are currently fierce competitors, making this scenario perhaps nothing more than theoretical for the time being.

Further Challenges. Specific passage retrieval is a heuristic technique that correlates fairly well with whether the terms in a passage are related in the way that they are in the query, without needing a more complex system that would parse the query and the passage and understand how all of the terms are syntactically related. It provides a useful way for people to find answers to specific questions, but it relies on human judgment to recognize the answers when it finds them. Much of my research with this technique has focused on finding relevant passages and displaying information to enable the human user to make this judgment quickly.

Beyond finding passages that are likely to contain an answer lies the challenge of understanding whether a passage has an answer and what the answer is. Sometimes this can require a considerable amount of knowledge-based reasoning.

For example, if an article contains the sentence, "Senator Daniel Patrick Moynihan told his colleagues that he wanted to make the longest word in the English language by taking the term 'floccinaucinihilipilification' and adding the suffix '-ism'," then the specific-passage-retrieval algorithm can find it in response to any of these queries: *longest word in the English language*; *longest word in English*; *longest English word*—or even *What is the longest word in the English language*. It takes some significant reasoning, however, to deduce from this sentence and its surrounding context that floccinaucinihilipilification is a possible answer. (Incidentally, this word, reported as the longest word in the first edition of the *Oxford English Dictionary*, is used in the 1994 Patrick O'Brian novel, *Master and Commander*, on which the recent movie was based.)

If you can understand whether a passage has an answer and what the answer is, then a next step would be to combine items of information from multiple sources to deduce answers. For example, if another article contains the sentence, "Since 1961, the longest word in the unabridged *Webster's Third New International Dictionary* has been 'pneumonoultramicroscopicsilicovolcanoconiosis'," then you can conclude that this is a better answer.

These tasks will require systems that can determine what passages are saying and reason with the resulting knowledge, and they will require additional sources of knowledge and advancements in automated reasoning. An active research area devoted to question-answering is pursuing such goals. Q

REFERENCES

1. Krovetz, R. Viewing morphology as an inference progress. *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval* (1993), 191202. (It also appears as a UMass technical report TR-93-36.)
2. Woods, W. A. Aggressive morphology for robust lexical coverage, *Proceedings of ANLP-2000*, Seattle, WA, May 1-3, 2000. (Preliminary version: Technical Report SMLI TR-99-82, Sun Microsystems Laboratories, Mountain View, CA, December 1999; http://www.sun.com/research/techrep/1999/abstract-82.html.)
3. Woods, W. A. Understanding subsumption and taxonomy: a framework for progress. In Sowa, J. (Ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, San Mateo: CA, 1991, 45-94.
4. Woods, W. A., Bookman, L. A., Houston, A., Kuhns, R. J., Martin, P., and Green, S. Linguistic knowledge can improve information retrieval. *Proceedings of ANLP-2000*, Seattle, WA, May 1-3, 2000; http://research.sun.com/features/tenyears/volcd/papers/woods.htm (final version with author's introduction).

WILLIAM A. WOODS is a principal scientist and distinguished engineer at Sun Microsystems Laboratories in Burlington, Massachusetts. He is internationally known for his research in natural language processing, continuous speech understanding, and knowledge representation, and he is currently interested in technology for improving people's access to information. He earned his Ph.D. at Harvard University, where he then served as an assistant professor and later as Gordon McKay Professor of the Practice of Computer Science. He is a past president of the Association for Computational Linguistics, a Fellow of the American Association for Artificial Intelligence, and a Fellow of the American Association for the Advancement of Science.