

Chapter 10

How a Search Engine Works

Elizabeth Liddy

Many Internet nomads are confounded when they enter a search query and get back a set of over 10,000 “relevant” hits, viewable in batches of 10. There are occasions when the intrepid searcher will plow through the list hoping to find the perfect link, but as Sue Feldman has shown us in Chapter 9, many times there are other factors at work that cause inappropriate results to rise to the top of the list. One of the factors that can lead to this type of misinformation may be erroneous assumptions by searchers as to what’s really going on “behind the curtain.” As a public service to all those who are curious to know, we offer this explanation for what happens after you enter a search query into the little box.

“Search engine” is the popular term for an Information Retrieval (IR) system. While researchers and developers take a broader view of IR systems, consumers think of them more in terms of what they want them to do, namely search the Web, an intranet, or a database. Consumers would actually prefer a finding engine, rather than a search engine.

Search engines match queries against an index that they create. The index consists of the words in each document, plus pointers to their locations within the documents. This is called an inverted file. A search engine or IR system comprises four essential modules:

- A document processor
- A query processor
- A search and matching function
- A ranking capability

While users focus on “search,” the search and matching function is only one of the four modules. Each of these four modules may cause the expected or unexpected results that consumers get when they use a Search Engine.

Document Processor

The Document Processor prepares, processes, and inputs the documents, pages, or sites that users search against. The Document Processor should perform some or all of the following steps:

1. *Normalize the document stream to a predefined format*
2. *Break the document stream into desired retrievable units*
3. *Isolate and meta-tags sub-document pieces*
4. *Identify potential indexable elements in documents*
5. *Delete stop words*
6. *Stem terms*
7. *Extract index entries*
8. *Compute weights*
9. *Create and update the main inverted file against which the search engine searches in order to match queries to documents*

Steps 1-3: Preprocessing

While essential and potentially important in affecting the outcome of a search, these first three steps simply standardize the multiple formats encountered when deriving documents from various providers or handling various Web sites. They serve to merge all the data into a single

consistent data structure that all the downstream processes can handle. The need for a well-formed, consistent format is of relative importance in direct proportion to the sophistication of later steps of document processing. Step 2 is important because the pointers stored in the inverted file will enable a system to retrieve various sized units, either site, page, document, section, paragraph, or sentence.

Step 4: Identify potential indexable elements in documents

Identifying potential indexable elements in documents dramatically affects the nature and quality of the document representation that the engine will search against. In designing the system, we must define the following: What is a term? Is it the alphanumeric characters between blank spaces or punctuation? If so, what about noncompositional phrases (phrases where the separate words do not convey the meaning of the phrase, like skunk works or hot dog), multiword proper names, or interword symbols such as hyphens or apostrophes that can denote the difference between “small business men” vs. small-business men?” Each search engine depends on a set of rules that its document processor must execute to determine what action is to be taken by the “tokenizer,” i.e., the software used to define a ‘term’ suitable for indexing.

Step 5: Delete stop words

This step helps save system resources by eliminating from further processing, as well as potential matching, those terms that have little value in finding useful documents in response to a customer’s query. This step used to matter much more than it does now when memory has become so much cheaper and systems so much faster, but since stop words may comprise up to 40 percent of text words in a document, it still has some significance. A stop word list typically consists of those word classes known to convey little substantive meaning, such as articles (*a, the*), conjunctions (*and, but*), interjections (*oh, but*), prepositions (*in, over*), pronouns (*he, it*), and forms of the “to be” verb (*is, are*). To delete stop words, an algorithm compares index term candidates in the documents against a stop word list and eliminates certain terms from inclusion in the index for searching.

Step 6: Stem terms

Stemming removes word suffixes, perhaps recursively in layer after layer of processing. The process has two goals. In terms of efficiency, stemming reduces the number of unique words in the index, which in turn reduces the storage space required for the index and speeds up the search process. In terms of effectiveness, stemming improves recall by reducing all forms of word to a base or stemmed form. For example, if a user asks for *analyze*, he or she may also want documents which contain *analysis, analyzing, analyzer, analyzes, and analyzed*. Therefore, the document processor stems document terms to *analy-* so that documents which include various forms of *analy-* will have equal likelihood of being retrieved, which would not occur if the engine only indexed variant forms separately and required the user to enter all. Of course, stemming does have a downside. It may negatively affect precision in that all forms of a stem will match, when, in fact, a successful query for the user would have come from matching only the word form actually used in the query.

Systems may implement either a strong stemming algorithm or a weak stemming algorithm. A strong stemming algorithm will strip off inflectional suffixes (*-s, -es, -ed*) and derivational suffixes (*-able, -aciousness, -ability*), while a weak stemming algorithm will strip off only the inflectional suffixes (*-s, -es, -ed*).

Step 7: Extract index entries

Having completed steps 1 through 6, the document processor extracts the remaining entries from the original document. For example, the following paragraph shows the full text as sent to a search engine for processing:

“Milosevic's comments, carried by the official news agency Tanjug, cast doubt over the governments at the talks, which the international community has called to try to prevent an all-out war in the Serbian province. President Milosevic said it was well known that Serbia and Yugoslavia were firmly committed to resolving problems in Kosovo, which is an integral part of Serbia, peacefully in Serbia with the participation of the representatives of all ethnic communities, Tanjug said. Milosevic was speaking during a meeting with British Foreign Secretary Robin Cook, who delivered an ultimatum to attend negotiations in a week's time on an autonomy proposal for Kosovo with ethnic Albanian leaders from the province. Cook earlier told a conference that Milosevic had agreed to study the proposal.”

Steps 1 through 6 reduce this text for searching to the following text:

“Milosevic comm carri offic new agen Tanjug cast doubt govern talk interna commun call try prevent all-out war Serb province President Milosevic said well known Serbia Yugoslavia firm commit resolv problem Kosovo integr part Serbia peace Serbia particip representa ethnic commun Tanjug said Milosevic speak meeti British Foreign Secretary Robin Cook deliver ultimatum attend negoti week time autonomy propos Kosovo ethnic Alban lead province Cook earl told conference Milosevic agree study propos”

The output of step 7 is then inserted and stored in an inverted file that lists the index entries and an indication of their position and frequency of occurrence. The specific nature of the index entries, however, will vary based on the decision in Step 4 concerning what constitutes an “indexable term.” More sophisticated Document Processors will have phrase recognizers, as well as Named Entity recognizers and Categorizers, to insure index entries such as *Milosevic* are tagged as a person and entries such as *Yugoslavia* and *Serbia* as countries.

Step 8: Compute weights.

Weights are assigned to terms in the index file. The simplest of search engines simply assign a binary weight: 1 for presence and 0 for absence. The more sophisticated the search engine, the more complex the weighting scheme. Measuring the frequency of occurrence of a term in the document creates more sophisticated weighting, with length-normalization of frequencies still more sophisticated. Extensive experience in Information Retrieval research over many years has clearly demonstrated that the optimal weighting comes from use of term frequency/inverse document frequency (tf/idf). This algorithm measures the frequency of occurrence of each term within a document. Then it compares that frequency against the frequency of occurrence in the entire database.

Not all terms are good discriminators; that is, they don't all single out one document from another very well. A simple example would be the word “THE.” This word appears in too many documents to help distinguish one from another. A less obvious example would be the word “antibiotic.” In a sports database, when we compare each document to the database as a whole, the term “antibiotic” would probably be a good discriminator among documents, and therefore would be assigned a high weight. Conversely, in a database devoted to health or medicine, “antibiotic” would probably be a poor discriminator, since it occurs very often. The tf/idf

weighting scheme assigns higher weights to those terms that really distinguish one document from the others.

Step 9: Create index

The index or inverted file is the internal data structure that stores the index information and that will be searched for each query. Inverted files range from a simple listing of every alphanumeric sequence in a set of documents/pages being indexed along with the overall identifying numbers of the documents in which that sequence occurs, to a more linguistically complex list of entries, their tf/idf weights, and pointers to where inside each document the term occurs. The more complete the information in the index, the better the search results.

Query Processor

Query processing has seven possible steps, though a system can cut these steps short and proceed to match the query to the inverted file at any of a number of places during the processing. Document processing shares many steps with query processing. More steps and more documents make the process more expensive for processing in terms of computational resources and responsiveness. However, the longer the wait for results, the higher the quality of results. Thus, search system designers must choose what is most important to their users—time or quality. Publicly available search engines usually choose time over very high quality because they have too many documents to search against.

The steps in query processing are as follows (with the option to stop processing and start matching indicated as “Matcher”):

1. Tokenize query terms
2. Recognize query terms vs. special operators
3. -----> Matcher
4. Delete stop words
5. Stem words
6. Create query representation
7. -----> Matcher
8. Expand query terms
9. Compute weights
10. -----> Matcher

Step 1: Tokenize query terms

As soon as a user inputs a query, the search engine, whether a keyword-based system or a full Natural Language Processing (NLP) system, must tokenize the query stream, i.e., break it down into understandable segments. Usually a token is defined as an alphanumeric string that occurs between white space and or punctuation.

Step 2: Recognize query terms vs. special operators

Since users may employ special operators in their query, including Boolean, adjacency, or proximity operators, the system needs to parse the query first into query terms and operators. These operators may occur in the form of reserved punctuation (e.g., quotation marks) or reserved terms in specialized format (e.g., AND, OR). In the case of an NLP system, the query processor will recognize the operators implicitly in the language used no matter how they might be expressed (e.g., prepositions, conjunctions, ordering).

At this point, a search engine may take the list of query terms and search them against the inverted file. In fact, this is the point at which the majority of publicly available search engines perform their search.

Steps 3 and 4: Delete stop words and stem words

Some search engines will go further and stop-list and stem the query, similar to the processes described in the Document Processor section. The stop list might also contain words from commonly occurring querying phrases, such as “I’d like information about...” However, since most publicly available search engines encourage very short queries, as evidenced in the size of query window they provide, they may drop these two steps.

Step 5: Creating the query representation

How each particular search engine creates a query representation depends on how the system does its matching. If a statistically based matcher is used, then the query must match the statistical representations of the documents in the system. Good statistical queries should contain many synonyms and other terms in order to create a full representation. If a Boolean matcher is utilized, then the system must create logical sets of the terms connected by AND, OR, or NOT.

An NLP system will recognize single terms, phrases, and Named Entities. If it uses any Boolean logic, it will also recognize the logical operators from Step 2 and create a representation containing logical sets of the terms to be AND’d, OR’d, or NOT’d.

At this point, a search engine may take the query representation and perform the search against the inverted file. More advanced search engines may take two further steps.

Step 6 Expand query terms

Since users of search engines usually include only a single statement of their information needs in a query, it becomes highly probable that the information they need may be expressed using synonyms, rather than the exact query terms, in the documents that the search engine searches against. Therefore, more sophisticated systems may expand the query into all possible synonymous terms and perhaps even broader and narrower terms.

This process approaches what search intermediaries did for end-users in the earlier days of commercial search systems. Then intermediaries might have used the same controlled vocabulary or thesaurus used by the indexers who assigned subject descriptors to documents. Today, resources such as WordNet are generally available, or specialized expansion facilities may take the initial query and enlarge it by adding associated vocabulary.

Step 7: Computer query term weight (assuming more than one query term)

The final step in query processing involves computing weights for the terms in the query. Sometimes the user controls this step by indicating either how much to weight each term or simply which term or concept in the query matters most and *must* appear in each retrieved document to ensure relevance.

Leaving the weighting up to the user is uncommon because research has shown that users are not particularly good at determining the relative importance of terms in their queries. They can’t make this determination for several reasons. First, they don’t know what else exists in the database and document terms are weighted by being compared to the database as a whole.

Second, most users seek information about an unfamiliar subject, so they may not know the correct terminology.

Few search engines implement system-based query weighting, but some do an implicit weighting by treating the first term(s) in a query as having higher significance. They use this information to provide a list of documents/pages to the user.

After this final step, the expanded, weighted query is searched against the inverted file of documents.

Search and Matching Functions

How systems carry out their search and matching functions differs according to which theoretical model of IR underlies the system's design philosophy. Since making the distinctions between these models goes far beyond the goals of this explanation, we will only make some broad generalizations in the following description of the search and matching function. Those interested in further detail should turn to R. Baeza-Yates and B. Ribeiro-Neto's excellent textbook on IR (*Modern Information Retrieval*, Addison-Wesley, 1999).

Searching the inverted file for documents which meet the query requirements, referred to simply as "matching," is typically a standard binary search no matter whether the search ends after the first two, five, or all seven steps of query processing. While the computational processing required for simple, unweighted, non-Boolean query matching is far simpler than when the model is an NLP-based query within a weighted, Boolean model, it also follows that the simpler the document representation, the query representation, and the matching algorithm, the less relevant the results, except for very simple queries, such as one-word, non-ambiguous queries seeking the most generally known information.

Having determined which subset of documents or pages match the query requirements to some degree, a similarity score is computed between the query and each document/page based on the scoring algorithm used by the system. Scoring algorithms base their rankings on the presence/absence of query term(s), term frequency, tf/idf, Boolean logic fulfillment, or query term weights. Some search engines use scoring algorithms not based on document contents, but rather, on relations among documents or past retrieval history of documents/pages.

After computing the similarity of each document in the subset of documents, the system presents an ordered list to the user. The sophistication of the ordering of the documents again depends on the model the system uses, as well as the richness of the document and query weighting mechanisms. For example, search engines that only require the presence of any alphanumeric string from the query occurring anywhere, in any order, in a document would produce a very different ranking from one by a search engine that performed linguistically correct phrasing for document and query representation and that utilized the proven tf/idf weighting scheme.

However, the search engine determines rank, and the ranked results list goes to the user, who can then simply click and follow the system's internal pointers to the selected document/page.

More sophisticated systems will go even further at this stage and allow the user to provide some relevance feedback or to modify their query based on the results they have seen. If either of these are available, the system will then adjust its query representation to reflect this value-added feedback and rerun the search with the improved query to produce either a new set of documents or a simple reranking of documents from the initial search.

What Document Features Make a Good Match to a Query

We have discussed how search engines work, but what features of a query make for good matches? Let's look at the key features and consider some pros and cons of their utility in helping to retrieve a good representation of documents /pages.

1. *Term Frequency*

How frequently a query term appears in a document is one of the most obvious ways of determining a document's relevance to a query. While most often true, several situations can undermine this premise. First, many words have multiple meanings or are polysemous. Think of words like "pool" or "fire." Many of the nonrelevant documents presented to users result from matching the right word but with the wrong meaning.

Also, in a collection of documents in a particular domain, such as education, common query terms such as "education" or "teaching" are so common and occur so frequently that their ability to distinguish the relevant from the nonrelevant in a collection declines sharply. Search engines that don't use a tf/idf weighting algorithm do not appropriately down-weight the overly frequent terms, nor do they assign a higher weight to appropriate distinguishing (and less frequently occurring) terms, e.g., "early-childhood."

2. *Location of Terms*

Many search engines give preference to words found in the title or lead paragraph or in the metadata of a document. Some studies show that the location, where a term occurs in a document or on a page, indicates its significance to the document. Terms occurring in the title of a document or page which match a query term are, therefore, frequently weighted more heavily than if they occur in the body of the document. Similarly, query terms occurring in section headings or the first paragraph of a document may be more likely to be relevant.

3. *Link Analysis*

Web-based search engines have introduced one dramatically different feature for weighting and ranking pages. Link analysis works somewhat like bibliographic citation practices, such as those used in the Science Citation Index. It is based on how well connected each page is, as defined by Hubs and Authorities. Hub documents link to large numbers of other pages (out-links) and Authority documents are those that are referred to by many other pages, or have a high number of "in-links" (J. Kleinberg, "Authoritative sources in a hyperlinked environment," *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*. 1998, pp. 668-77).

4. *Popularity*

Google and several other search engines add popularity to Link Analysis to help determine the relevance or value of pages. It utilizes data on the frequency with which a page is chosen by all users as a means of predicting relevance. While popularity is a good indicator at times, it assumes that the underlying information need remains the same.

5. *Date of Publication*

Some search engines assume that the more recent the information is, the more likely that it will be useful or relevant to the user. They, therefore, present results beginning with the most recent to the less current.

6. *Length*

While length per se does not necessarily predict relevance, it is a factor when used to compute the relative merit of similar pages. So, in a choice between two documents containing the same query terms, the document that contains a proportionately higher occurrence of the term relative to the length of the document is assumed more likely to be relevant.

7. *Proximity of query terms*

When the terms in a query occur near each other within a document, it is more likely that the document is relevant to the query than if the terms occur at greater distance. Though some search engines do not recognize phrases per se in queries, some search engines clearly rank documents in results higher if the query terms occur adjacent to one another or in closer proximity, as compared to documents in which the terms occur at a distance.

8. *Proper nouns*

Sometimes proper nouns have higher weights, since so many searches are performed on people, places, or things. Useful as this may be, if the search engine assumes that you are searching for a name instead of the same word as a normal everyday term, then the search results may be peculiarly skewed. Imagine getting information on “Madonna,” the rock star when you were looking for pictures of madonnas for an art history class.

Summary

The above explanation lays out the range of processing which might occur in a search engine, along with the many options that a search engine provider decides on. The range of options will help clarify users’ frequent surprise at the results their queries return. Until now, search engine providers have mainly opted for less, versus more, complex processing of documents and queries. The typical search results therefore leave a lot of work to be done by the searchers, who must wend their way through the results, clicking on and exploring a number of documents, before they find exactly what they seek. The typical evolution of products and services suggests that this status quo will not continue. Search engines that go further in the complexity and quality of the processing they perform will be rewarded with greater allegiance by searchers, as well as financially rewarding opportunities to serve as the search engine on more organizations’ intranets.

Searchers should keep watching for the best and pursuing it.