The Best Kept Secrets to Using Keyword Search Technologies

By Philip Sykes
and
Richard Finkelman

Part 1 – Understanding the Search Engines – A Comparison of dtSearch and Lucene

Introduction

Keyword searching is, and will continue to be, an important fundamental component of the electronic discovery workspace.

Effectively and correctly used, keyword searching is an important tool for the initial culling of large datasets prior to loading into a document review system. Constructing high-quality searches is a useful skill over the entire span of the litigation process.

General Information about dtSearch and Lucene

The first step in building powerful searches is to understand how the indexing and search engines work. The different eDiscovery tools incorporate various indexing and searching solutions. Some offer more than one.

For example, Relativity uses Microsoft's SQL Server Full Text Search for the document metadata and text. SQL Server Full Text Search has limited functionality, so Relativity also incorporates dtSearch for regular Boolean/proximity searching and Content Analyst for concept searching. Viewpoint also uses dtSearch. Clearwell, Intella, and SHIFT use Lucene for indexing and searching. This article compares these two indexing/search engines, explaining the similarities and differences.

dtSearch is a widely used indexing/searching tool that provides both Boolean and proximity searching options. A single-user dtSearch Desktop license costs $199.00 and is so useful that you should have it in your suite of software for testing keywords regardless of the eDiscovery applications that you use. dtSearch Desktop provides you with the capability to perform preliminary testing of key custodians' data. This is particularly important when you will be using a solution that uses other indexing/searching technology, since it gives you a control set to use for comparison with the results from the solution's searching tools.

Another reason why dtSearch is useful is that the syntax of the searches is similar to the syntax used by Concordance for full-text searching. If you have built your searches with dtSearch, it's easy to transform the searches to run in Concordance. In addition, many people who work on drafting proposed keyword terms build their lists with a dtSearch/Concordance–compatible structure.

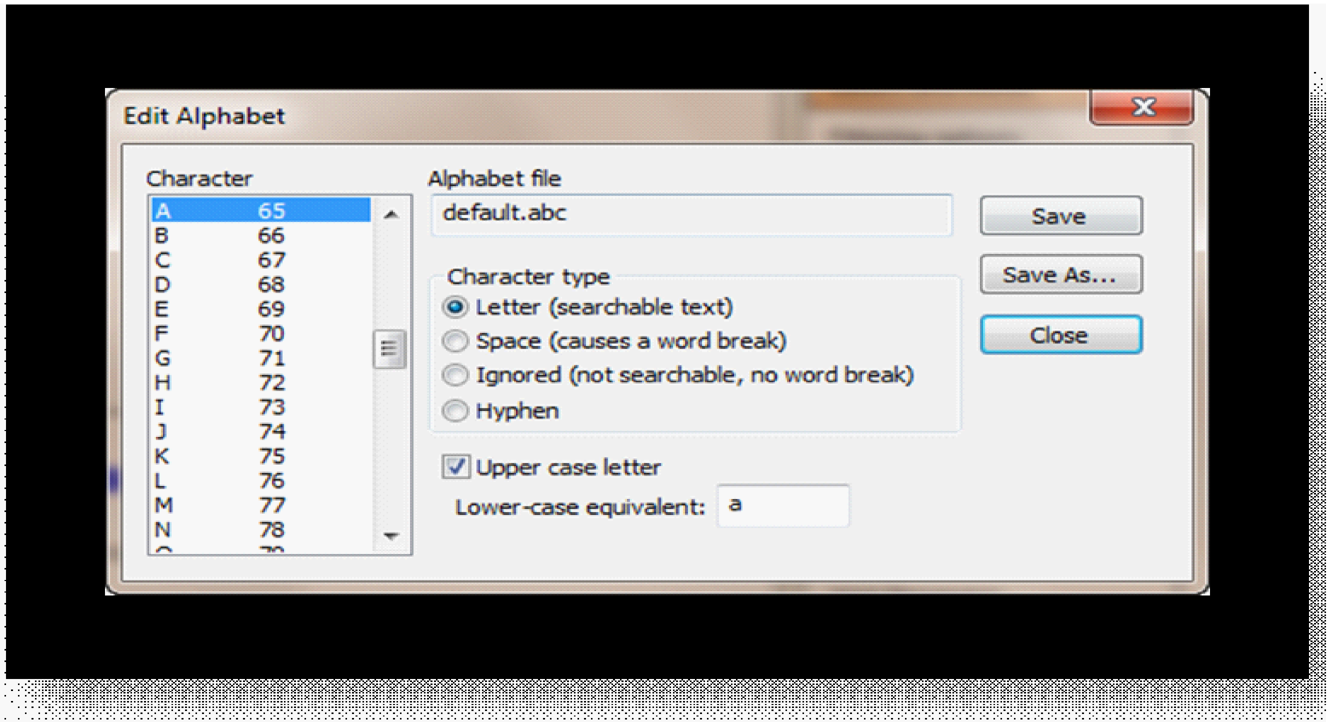
Lucene is an open-source (free) indexing and searching tool that has been used extensively to implement internet search engines. A number of eDiscovery solutions, including Clearwell, Intella, and SHIFT, use this technology, although there are differences in the way the developers have customized Lucene in the various tools. The same search may perform differently when used with these solutions, even with the same set of documents.

Differences between dtSearch and Lucene

Indexing

dtSearch and Lucene index some characters differently. Lucene treats all punctuation and symbols as word breaks. dtSearch is somewhat different, as it assigns all characters to one of four Character Types (Figure 1):

"If you have built your searches with dtSearch, it's easy to transform the searches to run in Concordance."



Character

A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78

Alphabet file: default.abc

Character type
- ● Letter (searchable text)
- ○ Space (causes a word break)
- ○ Ignored (not searchable, no word break)
- ○ Hyphen

✓ Upper case letter

Lower-case equivalent: a

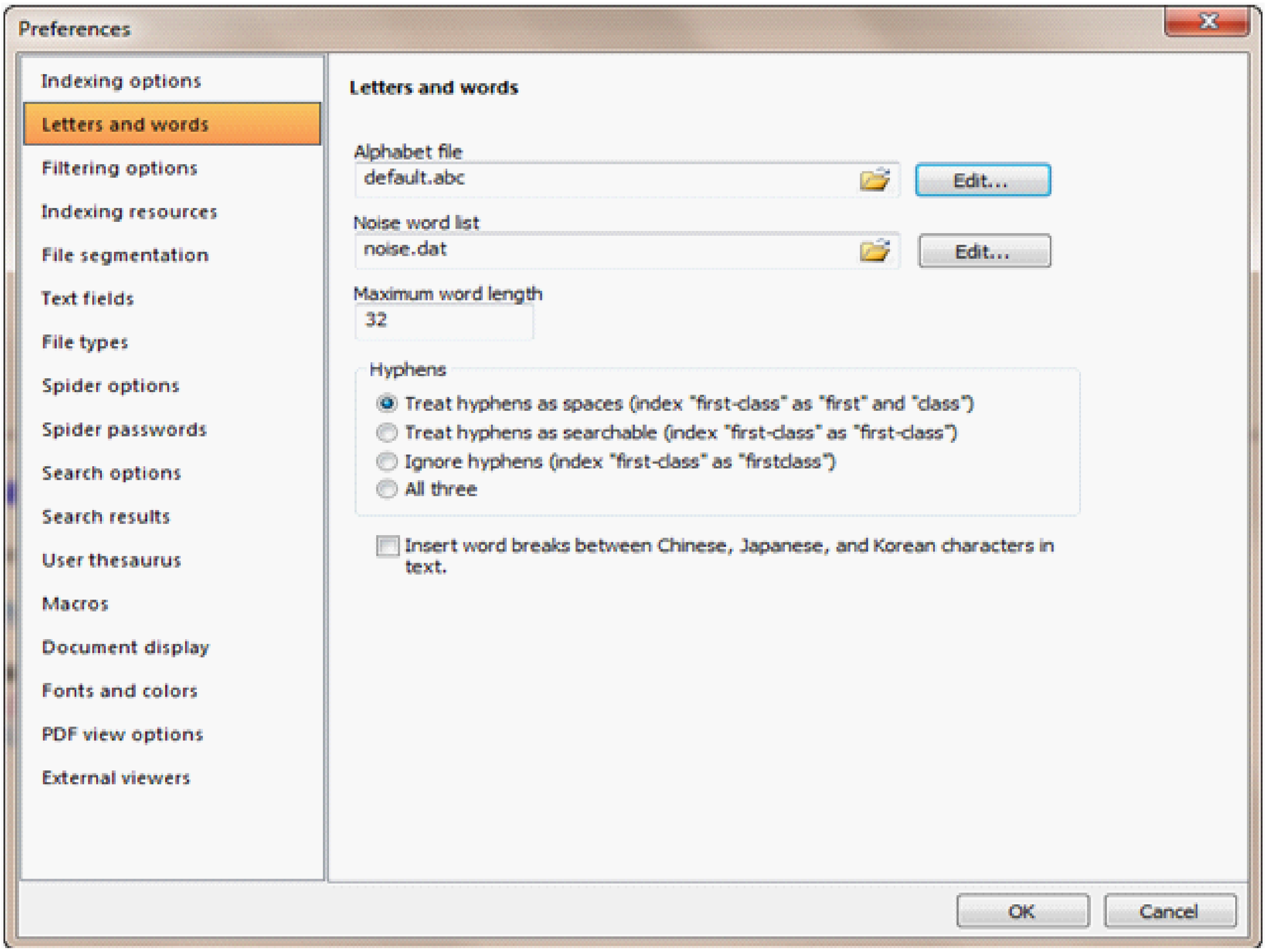
Figure 1: Character Types

dtSearch's treatment of characters is the same as Lucene except for two symbols—at least, if the default alphabet file hasn't been customized. The first difference is that dtSearch treats "_" as a letter, meaning it is indexed and there is no word break. So Smith_Tom would be indexed as a single word by dtSearch, whereas Lucene would treat it as two words: Smith and Tom.

The other difference is "%", which Lucene treats as a word break, while dtSearch sets it to Ignored and does not treat it as a word break. So Tom%Smith would be indexed as two separate words by Lucene (Tom and Smith), while dtSearch ignores the "%" and adds the word TomSmith to the index. (Note: the "%" is a reserved character in dtSearch that is used for fuzzy searching, so you may see it in search strings.)

The Hyphen Character Type, which by default only includes the hyphen character, is unique because when you build or update an index you have choices on how the Hyphen Character Type is treated (Figure 2).

"dtSearch is a widely used indexing/searching tool that provides both Boolean and proximity searching options. A single user license is $199."



Letters and words

Alphabet file: default.abc

Noise word list: noise.dat

Maximum word length: 32

Hyphens

- [✓ Treat hyphens as spaces (index "first-class" as "first" and "class")]
- Treat hyphens as searchable (index "first-class" as "first-class")
- Ignore hyphens (index "first-class" as "firstclass")
- All three

☐ Insert word breaks between Chinese, Japanese, and Korean characters in text.

Figure 2

If you select "All three" for Hyphens, the indexing of company-wide would cause the word to be indexed all three ways:

- Spaces, so there are two separate searchable words: company and wide
- Searchable, so company-wide could be searched separately from company wide
- Ignored, so companywide would be searchable.

dtSearch uses a list of Noise (Stop) Words that contains words that are considered too common to provide any value. In addition to common words like all, for, it, this, and was, dtSearch includes the letters a and i as noise words. All other single letters are indexed. dtSearch Desktop allows customization of the noise words list, but embedded versions of dtSearch often do not. One thing to note with dtSearch is how the hit-highlighting appears when there are noise words in your search. If you search for the phrase the records, when the hits are displayed in context every occurrence of records will be highlighted, as well the word immediately preceding it, since the is a noise word. So all records and some records would be considered hits, and both pairs of words would be highlighted.

For Lucene, the use of noise word lists depends upon the tool being used. Some, like Intella, do not use noise words; others do. You will need to refer to the tool's documentation or execute test searches to determine whether or not all words were indexed.

Wildcards

Both dtSearch and Lucene support wildcard characters using "?" as a single-letter wildcard and "*" as a multi-character wildcard. However, Lucene does not support wildcards inside quotation marks. So the search ["sales agreement*"] is an acceptable search using dtSearch, but Lucene ignores the wildcard character and displays results for the search ["sales agreement"]. (Note: in all search syntax examples, the actual search syntax will be inside square brackets "[ ]".)

Boolean Searches

Both dtSearch and Lucene provide Boolean and proximity searching. When using dtSearch, always have the "Search for" option set to Boolean.

dtSearch

When using dtSearch with the "Search for" option set to Boolean, a series of words is treated as a phrase, and quotation marks are not needed. (Note: if "Search for" is set to "Any words," then the list will be treated as if the words were joined by "OR"; if it is set to "All words," the list will be treated as if the words were joined by "AND.")

Lucene

Lucene's behavior depends upon the customization of the searching by the tool's software development team. Some tools treat a list of terms as if each word is separated by OR, and others treat them as if they were separated by AND.

If you run the search [sales agreement] in dtSearch, it returns documents containing the exact phrase sales agreement. Lucene would return all documents containing either sales OR agreement, or containing sales AND agreement anywhere in the document, depending on the tool's configuration. The best search in Lucene would be ["sales agreement"], since the results would not depend on how lists of individual words are treated.

While Lucene supports a list of terms separated by spaces to search for words, it's recommended that the terms be separated by AND or OR to aid in clarity, since others may not be familiar with the Lucene syntax, and so the query will perform the same in any tool based upon Lucene.

Proximity searching

Proximity searching with dtSearch and Lucene is very different, both in the syntax and how it works.

dtSearch Proximity Searching Syntax

Proximity searches in dtSearch can use either of two operators: W/ and PRE/. So the search [(records w/5 filed)] would return all documents where records has no more than four words between it and filed, regardless of the order in which records and filed appear. The search [(records PRE/5 filed)] would return all documents where records has no more than four words between it and filed and where records appears before filed. dtSearch accepts phrases and Boolean operators within proximity searches; for example, [(sales agreement w/6 (jones OR smith))].

As an example, we will use two documents. The first, which is returned by this search, contains the text:

“Sales agreement that was executed by William Jones and Thomas Smith.”

The second, which is not returned by this search, contains the sentence:

“Agreement for sales to William Jones by Thomas Smith.”

Note that while the phrase sales agreement must exist in the document for it to be returned by the search, only the last word in the phrase (in this example, agreement) must be within six words of either jones or smith for the document to be returned by the search. In this case, the terms agreement and jones are highlighted. If we increase the word count from six to seven, then both sales and agreement are within seven words of jones, so sales, agreement, and jones are highlighted.

Lucene Proximity Searching Syntax

Lucene’s proximity search syntax is [“Term1 Term2”~N], where “N” is the “edit distance” between the terms. However, Lucene’s approach to proximity searching is different than that used by dtSearch. Where dtSearch counts words, Lucene counts the edit distance value needed to match the words in the text to the words in the search, in the same order they appear in the search.

Therefore, if the query is [“price stock”~4], it is possible that different results will be returned by the search, only the last word in the phrase (in this example, agreement) must be within six words of either jones or smith for the document to be returned by the search. In this case, the terms agreement and jones are highlighted. If we increase the word count from six to seven, then both sales and agreement are within seven words of jones, so sales, agreement, and jones are high- lighted.

Continuing with the example [“price stock”~4], if we have a document that contains the text “stock has a current price,” it takes four shifts to the left to get price in the first position, so the minimum edit distance value is −4; it takes one shift to the right to get stock in the second position, so the maximum edit distance is +1. Subtracting the minimum from the maximum (1 – (−4)) = 5, so this document would not be returned by this search. It would be returned by [“stock price”~4], because stock is already in the first position, and only three shifts to the left are needed to get price into the second position.

Again, subtracting the minimum from the maximum (0 – (−3)) = 3, which is less than the specified edit distance of 4.

dtSearch and Lucene Proximity Search Comparison

For examples of how dtSearch and Lucene work differently, we will use four different text strings:

1. “price of the company’s stock”
2. “stock doesn’t have an increased price”
3. “price of the common stock”
4. “stock increase when the price”

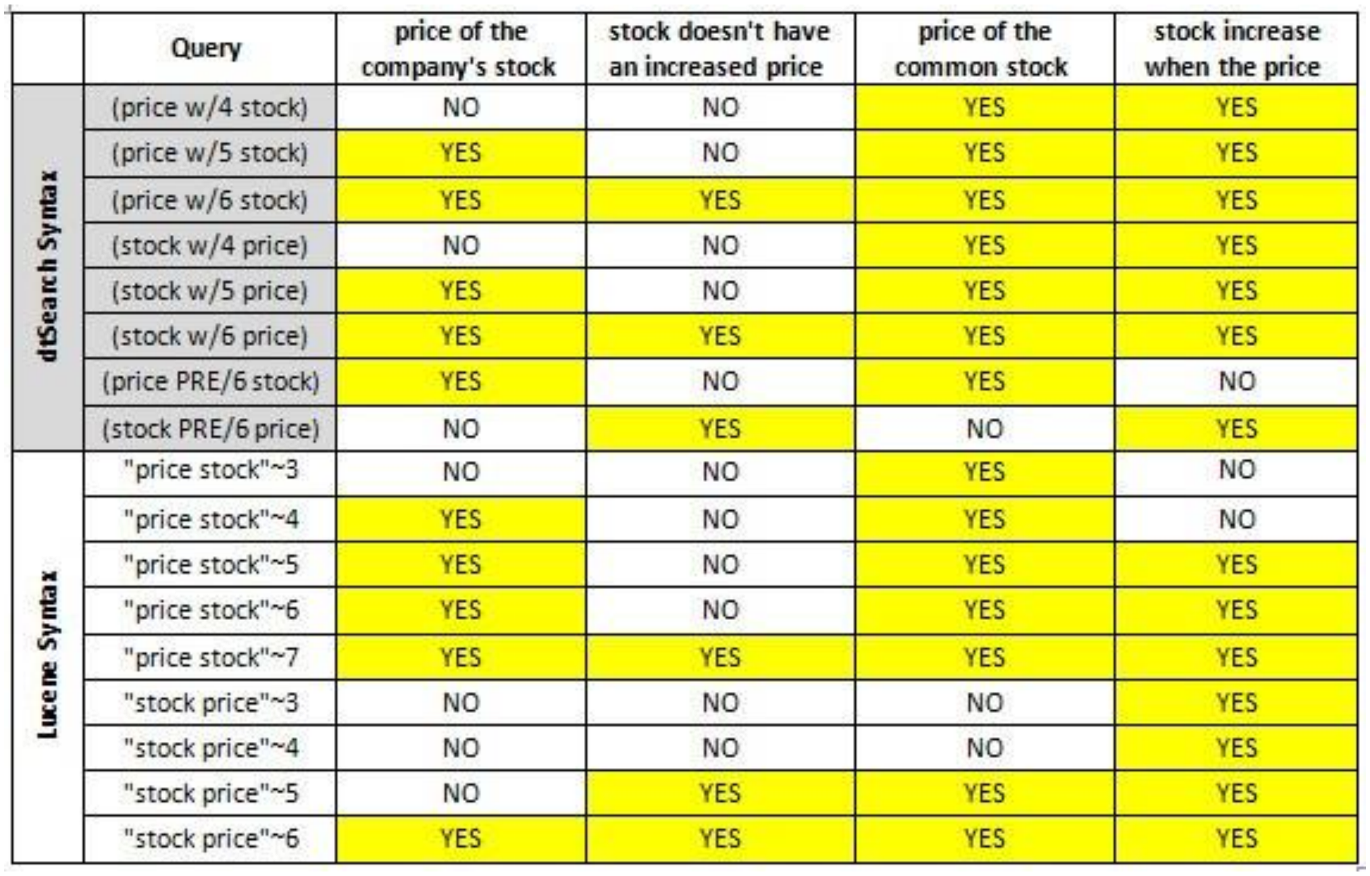
Using dtSearch, the minimum word count to obtain results is four. The query [(price w/4 stock)] returns numbers 3 and 4. The query [(price w/5 stock)] returns numbers 1, 3, and 4. (Note: the reason why number 1 was picked up by "w/5" but not "w/4" is that the apostrophe in "company's" is Character Type "space" and thus creates a word break.) The query [(price w/6 stock)] returns all four of the strings.

Reversing the term order in these queries has no impact on the results. If we switch to the query [(price PRE/6 stock)], only numbers 1 and 3 are returned. Conversely, if we search using the query [(stock PRE/6 price)], numbers 2 and 4 are returned.

Switching to Lucene, the minimum distance with results is three. The query ["price stock"~3] returns only number 1. Both numbers 1 and 3 are returned by ["price stock"~4]. The query ["price stock"~5] returns numbers 1, 3, and 4 (which contains the words in reverse order to the query). The query ["price stock"~7] returns all four strings. Reversing the order of the terms in the query changes the results.

Using ["stock price"~3] returns number 4.

Table 1 summarizes the results:



	Query	price of the company's stock	stock doesn't have an increased price	price of the common stock	stock increase when the price
dtSearch Syntax	(price w/4 stock)	NO	NO	YES	YES
	(price w/5 stock)	YES	NO	YES	YES
	(price w/6 stock)	YES	YES	YES	YES
	(stock w/4 price)	NO	NO	YES	YES
	(stock w/5 price)	YES	NO	YES	YES
	(stock w/6 price)	YES	YES	YES	YES
	(price PRE/6 stock)	YES	NO	YES	NO
	(stock PRE/6 price)	NO	YES	NO	YES
Lucene Syntax	"price stock"~3	NO	NO	YES	NO
	"price stock"~4	YES	NO	YES	NO
	"price stock"~5	YES	NO	YES	YES
	"price stock"~6	YES	NO	YES	YES
	"price stock"~7	YES	YES	YES	YES
	"stock price"~3	NO	NO	NO	YES
	"stock price"~4	NO	NO	NO	YES
	"stock price"~5	NO	YES	YES	YES
	"stock price"~6	YES	YES	YES	YES

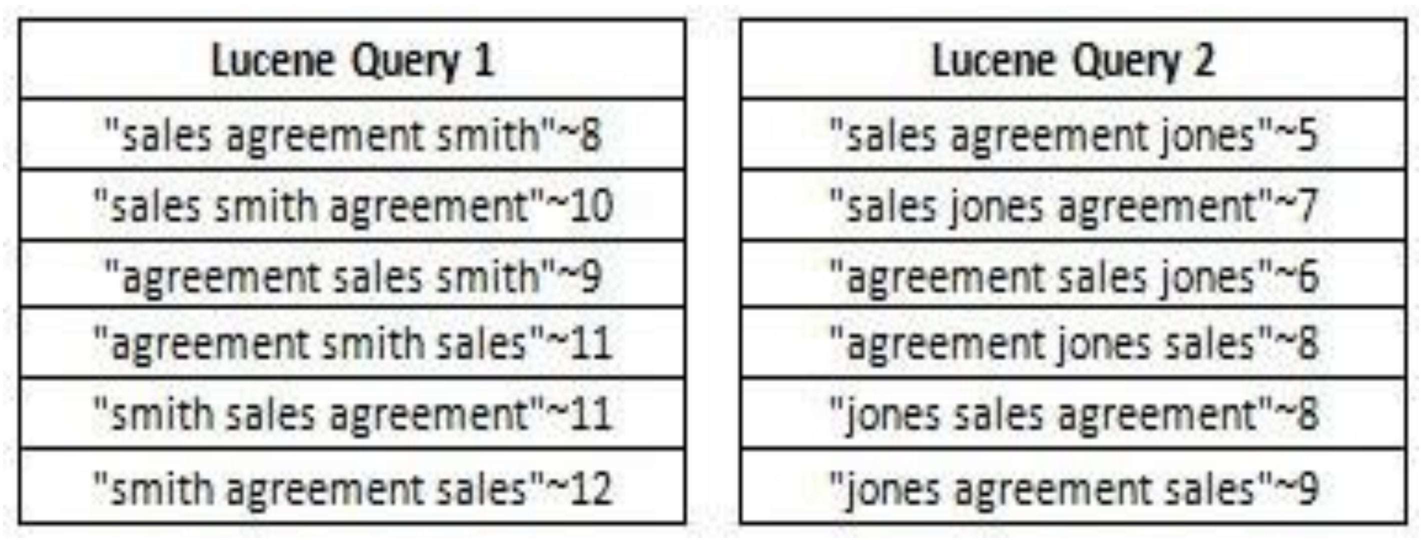
"Some tools treat a list of terms as if each word is separated byOR, and others treat themas if they were separated by AND."

Lucene also supports more than two terms within the quotes but, since the terms are inside quotes, wildcards are not permitted, and Lucene doesn't support the use of Boolean operators inside quotes.

If we need to adapt the dtSearch Boolean operator example [(sales agreement w/6 (jones OR smith))] to Lucene, it would require two proximity searches in order to deal with the OR. The most precise search would be [("sales agreement" AND "agreement jones"~N) OR ("sales agreement" AND "agreement smith"~N)]. The value of N will need to be determined through testing, but a good place to start would be N=5, since it's the dtSearch distance minus one.

A less-precise search would be ["sales agreement jones"~N OR "sales agreement smith"~N]; it is less precise in that it doesn't require the exact string sales agreement, so the terms could be in any order and not adjacent as long as sales, agreement, and jones or smith were within an edit distance of N.

Table 2 shows the value of N required to return this document (they all actually return both documents—a smaller value of N will return the second document, which has sales and agreement, but not sales agreement) using the same two documents as above, with the six possible orders of the three terms:



Lucene Query 1
"sales agreement smith"~8
"sales smith agreement"~10
"agreement sales smith"~9
"agreement smith sales"~11
"smith sales agreement"~11
"smith agreement sales"~12

Lucene Query 2
"sales agreement jones"~5
"sales jones agreement"~7
"agreement sales jones"~6
"agreement jones sales"~8
"jones sales agreement"~8
"jones agreement sales"~9

The above example was designed to show the impact of the term order differences on the value of N. In the real world when using Lucene proximity searches, the best approaches are:

If there are two words in the search, decide on the optimum number of words between the terms without regard to the term order, and run that search. For example, run the search "stock price"~4. After obtaining the results, run the search again with the terms reversed: "price stock"~4.

The results will normally be lower or higher than the first search. In order to check whether or not the value of N should be adjusted, perform "gap" queries for the search that returned the larger number of documents. For example, run the search [(("stock price"~5) AND NOT (("stock price"~4) OR ("price stock"~4)))], which will show the documents found by the new value of N that were not found by the previous value of N in either term order. Once you are satisfied with the value of N, which we'll assume is 4, then just run the search as [("stock price"~4 OR ("price stock"~4)].

If there are more than two words in the search, it becomes more difficult to test individual searches with the terms in different order (Note also that Lucene doesn’t include the edit distances for each of the terms in calculating the total edit distance for the document; it only uses the one that is the maximum value and subtracts the minimum value from it.).

Therefore, the most reasonable approach would be to start with an acceptable value for N and note the results. Then increase the value of N by the number of terms and note those results. Continue to run the search incrementing the value of N and run "gap" queries to return the documents from the search with the largest N value that were not in the previous N value search until the "gap" queries are not bringing back documents that belong in the review set.

For example, if the search ["term1 term2 term3"~8] returns 500 documents and the search ["term1 term2 term3"~10] returns 600 documents, then run the "gap" query [("term1 term2 term3"~10) AND NOT ("term1 term2 term3"~8)], and review the documents returned. If there are potential relevant documents, increase the values of N to [(“term1 term2 term3”~11) AND NOT (“term1 term2 term3”~10)], and review the documents returned.

Up Next

Part 2 will cover using structured search techniques to build your keyword searches.

About the Authors

Philip Sykes is a Senior Managing Consultant with more than 20 years of experience working in the fields of litigation support and electronic discovery. His experience includes work on high-profile cases from HSR Second Requests from the FTC and DOJ to IP matters to complex securities cases. His expertise includes keyword analytics, data processing and analysis, data management, on-line review tools, and document productions. He regularly assists counsel and experts in understanding what information exists in databases and how it is relevant to their matters.

Richard Finkelman is a Director and Practice Group Leader of Berkeley Research Group’s Electronic Discovery Practice. Mr. Finkelman brings more than 25 years of experience helping clients manage information in litigation, regulatory and business matters. His experience includes assisting clients with all aspects of litigation support in complex matters ranging from Securities Class Actions to Intellectual Property Disputes to high profile Regulatory matters.