FastSSFuzzy

We have a use case where we need to do lots of approximate text searching on large text collections. The default FuzzyQuery must enumerate the index and did not scale to our use (30 second fuzzy queries)

These researchers have an interesting solution to the problem: http://fastss.csg.uzh.ch/
By building the offline datastructure described in the paper, queries are now subsecond.

For our case we only indexed deletion neighborhoods of k=1 edit distance. (I think somewhere this is mentioned to cover around 80% of common typos) In our implementation, not wanting to actually modify the lucene index structure, instead we created an additional lucene index, called the "fuzzy index" which works in tandem with the standard index. We wrapped QueryParser with code to expand fuzzy queries against this fuzzy index.

Example Index Document:
*DOCID: 1
*Text: The quick brown fox ... Mispellings are common.

After indexing this into a normal lucene index, we build the second index, the fuzzy index. The method is to do a TermEnum on the just-built standard index, and then index each term with its deletion neighborhood

Example fuzzy Index document:
*DOCID: 1
*WORD: mispellings
NEIGHBORHOOD: mispellings ispellings mspellings misellings mispllings mispelings mispellngs *mispellins mispelling

At runtime, to expand a fuzzy query such as "MISPELLIGNS", the deletion neighborhood for the term is generated and OR'ed together as a query

MISPELLIGNS -> (neighborhood:mispelligns OR neighborhood:ispelligns OR neighborhood:mspelligns OR ... OR **neighborhood:mispellins**)

this returns all the terms with edit distance of 1 from the query term, in exhaustive offline fashion. the results from this query are then used to expand the fuzzy term against the original index.

While this solution of using a secondary index and expanding the query is not exactly a tight integration, someone with more motivation could certainly build this (I did not want to modify index structure to store additional data). For our case, we are happy to be able to easily upgrade to new lucene releases with this loosely coupled implementation.

If you are concerned with the quality of approximate text searches, you can take advantage of this additional index to improve the quality as well, by indexing additional fields like phonetic keys and using the combination of offline edit distance, phonetic keys, and "smarter" algorithms such as Jaro-Winkler at runtime to provide a fuzzy operator more akin to the aspell algorithm.

If you are stuck in a similar situation, need more explanation, or have other interesting solutions to this problem feel free to email me at rcmuir@gmail.com