LexEVS 5.1 Search Algorithm Discussion

Archived from the GForge wiki of the LexEVS project. Last edited August 14, 2009.

- LexEVS 5.1 Search Algorithms Feedback

NOTE: For all questions regarding special characters, see GForge http://gforge.nci.nih.gov/tracker/index.php?func=detail&aid=22565&group_id=491&atid=1853 Many of the characters referred to below are dropped during our indexing process. We may choose to not drop some of these, but we will need to decide which ones, as denomalizing our indexes by leaving in special characters will impact search precision.

- Lucene Query
- phrase ("phrase in string" matches "there is a phrase in string sometimes")
  - The algorithm seems to have difficulty finding terms containing '*' or '^' character.
- contains (* term* *)
  - The method may throw exception if the search string contains special characters; for example, (R*,S*)~~4,4'~~(1,2-Diethyl-1,2-ethanediyl)bis [phenol].
- Leading And Trailing Wild Card (**term**)
  - Question: If I search against NCI MetaThesaurus using **ell aggregation factor**as a search string, do I expect the algorithm to return the concept Cell Aggregation (C0007580)? And, how does this algorithm differ from the substring algorithm?
    - The idea is for the user to not have to insert any wildcard characters at all. The algorithms will do that automatically. So, Leading and Trailing Wild Card algorithm inserts a wildcard at the beginning and end of each term. So, if the user inputs 'ell aggregation factor', the algorithm will turn that into '**ell aggregation factor**'. If the user wants to be in control of where the wilcards are placed, they should use 'Lucene Query' and place the wildcards where necessary. But getting to the question, if the user inputs 'ell aggregation factor' in 'Leading And Trailing Wild Card' - it WILL return "Cell Aggregation" because the term matches are currently OR'ed. So the query looks like (**ell** OR **aggregation** OR **factor**). The more terms that match the higher the score will be. This can be an 'AND' if that is preferred – (**ell** AND **aggregation** AND **factor**), this will NOT match Cell Aggregation because it won't contain the '**factor**' term.
- This differs from substring because substring does not place a leading wildcard (this is for performance reasons – leading wildcards are very expensive searches). If a user inputs 'hear ttack' – two things will happen:
  - First, the indexes will be searched for 'hear*' and 'ttack*'
  - Next, the indexes will be searched for 'raeh* and kcatt*'. We index every property value in reverse, so this step will be compared to the reverse index. This way we can eliminate the leading wildcard.
  - Implementation aside, what we end up with is ((**hear OR hear**) OR (**ttack OR ttack**))
  - NOTICE for this search, you cannot input 'ear' and expect to match 'heart' because even searching against the reverse index, this would require a leading wildcard to match.
- exactMatch
  - The algorithm seems to have difficulty finding terms containing '*', '^', and posssibly '?' character.
    - test
- subString (**term or term** but NOT **term**) NOTE: Not functional yet in NCI Meta Browser Prototype – pending re-indexing
  - Problem iterating through a large search result:
    - A RemoteAccessException can be thrown when iterating through a large search result on a client machine with not enough memory. This occurs while calling the following statement multiple times:
      - iterator.next(maxReturn).getResolvedConceptReference();
    - It fails non-deterministically. For example, the following search fails on a machine with 2GB of RAM:
      - searchText = "protein", algorithm = "subString", maxReturn = 100
      - Note: Should find close to 70,000 references.
- Stemmed Lucene Query
- startsWith (term*)
  - The algorithm seems to have difficulty finding terms containing '?' or '^' character.
- Double Metaphone Lucene Query (spelling-error tolerant)
  - The ranking part of the algorithm may need to be improved. For example, searching for "brest kancer" would return NCI MetaThesaurus concepts in the following order:
    - (1) C0006142:Malignant neoplasm of breast
    - (2) C0376358:Malignant neoplasm of prostate
    - (3) C0600139:Prostate carcinoma
    - (4) C0678222:Breast Carcinoma
    - (5) CL324486:Prostate Cancer
    - (6) C1863600:PROSTATE CANCER/BRAIN CANCER SUSCEPTIBILITY
    - (7) CL008190:Breast Cancer Surveillance Consortium
    - (8) C0007104:Female Breast Carcinoma
    - ...
  - The search may fail if the target string contains a special character like '*'.
- Weighted Double Metaphone Lucene Query (spelling-error tolerant, but exact entered search string is also considered)
  - The ranking part of the algorithm may need to be improved. For example, searching for "brest kancer" would return NCI MetaThesaurus concepts in the following order:
    - (1) C0006142:Malignant neoplasm of breast
    - (2) C0313407:hemoglobin Brest
    - (3) C0376358:Malignant neoplasm of prostate
    - (4) C0600139:Prostate carcinoma
    - (5) C0678222:Breast Carcinoma
    - (6) CL324486:Prostate Cancer
    - (7) C1863600:PROSTATE CANCER/BRAIN CANCER SUSCEPTIBILITY
    - (8) CL008190:Breast Cancer Surveillance Consortium
    - (9) C0007104:Female Breast Carcinoma

        - (10) C0007112:Prostate Adenocarcinoma
        - ...
    - The search may fail if the target string contains a special character like '*'.
- Reg Exp
    - The algorithm fails on ".**ene.**" (Refer to GF#21849).
        - Lucene builds RegExp queries by constructing a query with multiple Boolean Query Clauses. Because of this, it does not scale well to large ontologies. For we now (as of Aug 3rd) will throw a parameter exception if the user inputs a RegExp query that is too general. Basically, we are catching the 'exceeded maxClause error' of Lucene. Leading And Trailing Wild Card will handle this much more gracefully – that algorithm will accept 'ene', turn it into '**ene**', and return results.
- literal (escapes all special characters)
    - searches on ALL characters, even ones that are Lucene special characters. Everything the user inputs is taken literally – if they input a special character is it escaped by the algorithm automatically.