Appendix 1: Principles of Indexing in IR

The steps of word indexing are described below. Some of the operations involved are also employed in concept-identification algorithms.

- The pre-processing step scans every document in a collection word-by-word, skipping words belonging to a stop-word list. *Stop-words* are very common words in the language or domain that are not useful for searching. A disk-based structure (the index) is created, which records the words that occur in each document and their frequency of occurrence, both in individual documents and in the collection as a whole. Some indexes also record *proximity data*—the position of individual words within the document as word, sentence or paragraph offsets.
- The user searches the collection by specifying one or more keywords, optionally combined in complex Boolean fashion. The retrieval process then returns matching documents in descending order of relevance, giving greater weight to documents containing the keywords many times, and to keywords that are less common in the collection. Several relevance-ranking metrics have been described, e.g., (4, 5).
- The index stores words either as they occur in the text, or after transformation. One type of transformation is *normalization*, which involves lower-case conversion and removal of inflections, e.g., variations in person or tense. Thus, children is normalized to child . Another transformation is *stemming* , where the word undergoes progressive suffix removal or letter substitutions to yield a root form. Stemming may yield the same end-result as normalization but is more drastic, often yielding a root that is not a word in the language. Thus, hypothesis , hypothesize and hypothetical yield the same root, hypothes .

Implementations of normalization and stemming algorithms are readily available. The National Library of Medicine (NLM) lexical tools contain a normalization routine called *norm* (6), as well as an implementation of the widely used Porter stemming algorithm, originally described in (7). Both normalization and stemming increase retrieval sensitivity at the cost of specificity. Further, the original Porter algorithm is known to have problems in stemming words with Greco-Latin suffixes (-um, -ae, -ii, etc.), which often occur in medicine. The C language implementation of the Porter algorithm as described in (8) is driven by rules that are stored in tables, however, and such suffixes can be managed by adding more rules.

For a large document collection, it is not space-efficient to store the words themselves in the index. Instead, a table is created to store every unique word in the collection, and every word is assigned an ID. This ID, which is typically a 4-byte integer, is used in the index. Indexing by concept is similar in principle: the ID of the concept in the thesaurus is used instead of a word ID.