A Method for Compressing Lexicons

Strahil Ristov, *Rudjer Boskovic Institute, Zagreb, Croatia, ristov@rudjer.irb.hr*
Eric Laporte, *Université de Marne-la-Vallée, France, laporte@bastille.univ-mlv.fr*

Lexicon lookup is an essential part of almost every natural language processing system. Natural language lexicon is a set of strings where each string consists of a word and the associated linguistic data. Its computer representation is a structure that returns appropriate linguistic data on a given input word. It should be small and fast.

We are proposing a method for lexicon compression based on a very efficient trie compression method [2] and the inverted file paradigm [3]. The method was applied on a 664 000 string, 18 Mbyte, French phonetic and grammatical electronic dictionary for spelling-to-phonetics conversion. Entries in the lexicon are strings consisting of a word, it's phonetic transcription, and some additional codes. An example entry is (*phonétique,fonetik,.A31:fs)*.

Although a trie can be built from original lexicon and compressed, the compression performance is much better for sets of shorter strings. The phonetic lexicon can be separated in two sets of shorter strings: *words* and *phonetics*, where only French words are in the first, and the phonetic transcriptions and the additional data in the second. When compressed tries are produced from two sets, the sum of their sizes is much less than the size of the compressed original lexicon. The challenge lies in how to connect appropriate word/phonetics pairs in two structures in such a way that overall reduction in size justifies the added complication. To compress the trie efficiently, input sets must be sorted in advance [2] therefore in-set ordering can't be used. Furthermore, there may exist multiple phonetics for a given word, so the problem is a complex one. It can be solved in a way reminiscent to inverted file index compression [3].

Compressed trie allows for a procedure that transforms it to order preserving minimal perfect hash function [1]. The number of final states that can be reached from the node in trie is stored in the node. Then, when traversing the trie following the input word we can extract that word's ordinal number. Inversely, a word can be obtained through its number. Since members of pairs word/phonetics don't have the same numbers, some sort of index must be built. The index is a list of phonetics numbers for each word number. The search procedure then goes as follows. First, input word is looked up in *words* trie, if it exists a number w is returned. Then, the list of phonetics numbers p_w in index position w is retrieved. Finally, *phonetics* trie is searched returning phonetics entry for every number in the list. The largest part of the system is index so it is compressed as follows. Every N-th list in index starts with the explicit phonetics number and the rest of the numbers are represented as the difference from the preceding value enabling very good compression with canonical Huffman codes. The drawback is necessity for slower sequential search but the speed is still acceptable: tens of thousands words per second for N around 100. Overall size of our compressed searchable lexicon is 7% of its textual representation.

[1] Lucchesi and Kowaltowski. Applications of finite automata representing large vocabularies, *Software-Practice and Experience*, 23(1) pp. 15-30, 1993.
[2] Ristov and Laporte. Ziv Lempel compression of huge natural language data tries using suffix arrays, *Proceedings of CPM'99,* LNCS 1645, pp. 196 – 211, 1999.
[3] Witten, Moffat and Bell, *Managing Gigabytes*, Morgan Kaufmann, 1999.