On the Use of Suffix Arrays for Memory-Efficient Lempel-Ziv Data Compression

Artur Ferreira[1,4], Arlindo Oliveira[2,3], Mário Figueiredo[2,4]
[1]*Instituto Superior de Engenharia de Lisboa,* [2]*Instituto Superior Técnico,*
[3]*INESC-ID,* [4]*Instituto de Telecomunicações,*
Lisboa, PORTUGAL Email: arturj@cc.isel.ipl.pt

The Lempel-Ziv 77 (LZ77) and LZ-Storer-Szymanski (LZSS) text compression algorithms use a sliding window over the sequence of symbols, with two sub-windows: the *dictionary* (symbols already encoded) and the *look-ahead-buffer* (LAB) (symbols not yet encoded). Binary search trees and *suffix trees* (ST) have been used to speedup the search of the LAB over the dictionary, at the expense of high memory usage [1]. A *suffix array* (SA) is a simpler, more compact data structure which uses (much) less memory [2,3] to hold the same information. The SA for a length m string is an array of integers $(a[1], ...a[k], ...a[m])$ that stores the lexicographic order of suffix k of the string; sub-string searching, as used in LZ77/LZSS, is done by searching the SA.

We propose two new SA-based LZ77/LZSS encoding algorithms. Algorithm A1 builds a single SA for the initial dictionary and every time the end of the LAB is reached, the LAB is slid into the dictionary and the SA is efficiently updated: delete suffixes $\{1, ..., |LAB|\}$; compute the SA for the encoded LAB, P_{LAB}, and set it to $P_{LAB} = P_{LAB} + |dictionary| - |LAB|$; do a sorted insertion of P_{LAB} into P. Besides the dictionary and LAB, the encoder uses only the SA. Algorithm A2 computes the SA and *longest common prefix* (LCP) values for the concatenation of dictionary and LAB, in order to get the position and the length of the longest sub-string. The encoder uses two integer arrays (SA and LCP) plus two strings (dictionary and LAB).

Our algorithms, which are also adequate for text retrieval, although slower, use 3 to 5 times less memory than the tree-based encoders (see Table I). More importantly, the required memory is independent of the text, thus only the strictly needed memory has to be allocated. This is important, *e.g.*, in embedded systems where memory is at a premium and speed is not critical.

Table I: Encoding time (in seconds) and compression ratio (in bpb), for $(|\text{dictionary}|, |\text{LAB}|) = (1024, 128)$. Memory usage: 3200 bytes for A1; 5248 bytes for A2; 12300 bytes for Binary-Tree (BT).

		A1		**A2**		**BT**	
File	**Size**	**Time**	**bpb**	**Time**	**bpb**	**Time**	**bpb**
paper5	11954	0.11	5.73	0.06	5.68	0.02	5.18
progl	71646	0.65	4.38	0.42	4.26	0.10	3.95
alice29.txt	152089	1.57	5.54	0.86	5.49	0.22	5.39
lcet10.txt	426754	4.56	5.66	2.46	5.55	0.37	5.38

References

[1] N. Larsson, *Structures of String Matching and Data Compression*, PhD thesis, Department of Computer Science, Lund University, Sweden, 1999.
[2] U. Manber and G. Myers, *Suffix Arrays: a new method for on-line string searches*, SIAM Journal on Computing, 22(5):935–948, 1993.
[3] S. Zhang and G. Nong, *Fast and space efficient linear suffix array construction*, IEEE Data Compression Conference DCC2008, p. 553, 2008.

1068-0314/09 $25.00 © 2009 IEEE
DOI 10.1109/DCC.2009.50