On the Temporal HZY Compression Scheme

Z. Cohen [*] Y. Matias [†] S. Muthukrishnan [‡] S. C. Şahinalp [§] J. Ziv [¶]

The HZY compression scheme. We consider (α, β)-HZY compression scheme with temporal ordering. Given an input string $T = T[1 : n]$ from a constant sized alphabet, the (α, β)-HZY scheme compresses T as follows. T is partitioned into disjoint *blocks* (substrings) of size β, $T[1 : \beta], T[\beta + 1 : 2\beta], \ldots$. For $1 \leq k \leq n/\beta$, the kth block $T[(k-1)\beta + 1 : k\beta]$ is then replaced by its *codeword* C_k, which is the integer pair (x, y); here (i) $x \leq \alpha$, is the size of the *context* $T[(k-1)\beta - x : (k-1)\beta]$ where $T[(k-1)\beta - x : k\beta]$ is the longest suffix of $T[1 : k\beta]$ occurring in $T[1 : (k-1)\beta]$, and (ii) y is the *temporal rank* of the block $T[(k-1)\beta+1 : k\beta]$ defined to be the number of *distinct* occurrences of the context $T[(k-1)\beta - x : (k-1)\beta]$ between the leftmost occurrence of $T[(k-1)\beta - x : k\beta]$ and its current occurrence. (When counting the number distinct occurrences of a context we only consider its occurrences with distinct blocks following it.)

Comments on the HZY compression scheme. The HZY scheme given above is a context-based compression method recently proposed in [HZ98] and independently in [Yok95]. It is a *one-pass* algorithm as the codeword of a given block $T[(k-1)\beta + 1 : k\beta]$ depends only on the "past", i.e., $T[1 : (k-1)\beta]$. The critical values of the parameters identified in these works are: $\beta = O(1)$ in [Yok95]; $\beta = O(\log \log n)$ and $\alpha = O(\log n)$ in [HZ98]. Also, $2^\beta \geq \alpha \geq \beta$ must hold in order to have compression. A detailed analysis of this scheme is provided in [HZ98], which shows that on any input generated by a stationary ergodic source, the output of the temporal HZY scheme converges to the *conditional entropy* of that source. No other one-pass compression scheme is known to be optimal under this refined notion of information content which makes this HZY scheme attractive. (See [HZ98] for further details.)

Our Results. (i) We provide a tighter and more general bound on the compression attained by the HZY scheme with temporal ordering, (ii) we give the first known efficient algorithm for implementing the scheme, and (iii) we provide some preliminary experimental results that are promising.

Analysis of the compression ratio attained by the HZY scheme. Our analysis here is for binary strings. Consider the string $T[-n : \beta] = T[-n], \ldots, T[0], \ldots, T[\beta]$, a sliding-window of length $n + \beta$ over the infinite string $T = \ldots, T[0], \ldots, T[\beta]$. For any $\beta = 1, 2, 3, \ldots, O(\log n)$, let $EL(T[1 : \beta] \mid T[-n : 0])$ be the expected number of bits to represent the codeword for block $T[1 : \beta]$ (in $T[-n : 0]$).

Theorem. $EL(C(T[1 : \beta]) \mid T[-n : 0]) \leq H(T[1 : \beta] \mid T[-t : 0]) + 2 \log t_{max} + O(\log \log n)$. *Here $t = t(T[-n : \beta])$ denotes the context of block $T[1 : \beta]$, i.e., the largest integer $j = 1, 2, \ldots$ such that $T[-j - i : \beta - i] = T[-j : \beta]$ for some $i = 1, 2, \ldots, n - j$, and t_{max} denotes maximum context size α.*

Proof. (sketch) $H(T[1 : \beta] \mid T[-t : 0]) = H(T[1 : \beta], i = t \mid T[-t : 0]) - H(i = t \mid T[-i : 0])$. For every integer $0 \leq i \leq t_{max}$, let $N_T(T[1 : \beta], i = t \mid T[-i : 0])$ be the first occurrence of $T[1 : \beta], i = t$ among all instances in T with the suffix $T[-i : 0]$. Then by the conditional Kac's Lemma proven in [HZ98]: $E[N_T(T[1 : \beta], i = t \mid T[-i : 0]) \mid T[-i : 1], i = t] = 1/P(T[1 : \beta], i = t \mid T[-i : 0])$. Thus, by the convexity of the logarithmic function $E \log(N_T(T[1 : \beta], i = t \mid T[-i : 0]) \mid T[-i : 0]) \leq H(T[1 : \beta], i = t \mid T[-i : 0])$. Therefore, $E \log(N_{T[-n:\beta]}(T[1 : \beta] \mid T[-t : 0])) \leq H(T[1 : \beta], i = t \mid T[-t : 0])$. But it is possible to generate a universally decodable code for $N_{T[-n:\beta]}(T[1 : \beta] \mid T[-t : 0])$ whose length (in terms of number of bits) is no more than $\log t_{max} + \log N_{T[-n:\beta]}(T[1 : \beta] \mid T[-t : 0]) + O(\log \log n)$ (or $\log t_{max} + \log min_{i=1,\ldots,t} N_{T[-n:\beta]}(T[1 : \beta] \mid T[-i : 0]) + O(\log \log n)$). ∎

This theorem is valid for *any* $\beta = O(\log n)$ hence is more general than the analysis provided in [HZ98] (which has the requirement that $\beta = o(\log n)$ and $\log \log n = o(\beta)$). Also, our bounds are tighter and simpler to prove.

Corollary. *It was demonstrated in [HZ98] that there exists ergodic sources for which, for* any *universal noiseless encoder: $EL(T[1 : \beta] \mid T[-n : 0]) \geq H(T[1 : \ell] \mid T[-t : 0]) - O(\log \log n)$. Thus, the HZY algorithm is optimal in a min-max sense.*

Efficient Algorithm for the HZY scheme. We present the first efficient algorithm for implementing the HZY scheme. At the high level, our algorithm works as follows. The string T is read left to right, block by block. Say the current block is $B_i = T[i : i+\beta-1]$. The data structure we maintain is the

[*] Dept of EE, Technion, Israel; zeev@ee.technion.ac.il.

[†] Dept of CS, Tel-Aviv University, Israel; supported by Alon Fellowship, the Israel Science Foundation, and the Israeli Ministry of Science; matias@math.tau.ac.il.

[‡] AT&T Labs, NJ; muthu@research.att.com.

[§] Dept of EECS, Case Western Reserve University; cenk@eecs.cwru.edu.

[¶] Department of EE, Technion, Israel; ziv@ee.technion.ac.il.

trie of all substrings of the input string $T[1:i-1]$, of length $\alpha+\beta$. The string labeling the path from the root to a node u is denoted σ_u. Each node in this trie has a suffix link which we also maintain. In addition, at each node u, we maintain the largest j such that $T[j:i-1]$ has the prefix σ_u; this we denote $pos(\sigma_u)$.

Step 1. Find the node u, if any, such that σ_u is the context of B_i; in what follows, we assume u exists (otherwise, the compression is trivial).

This is done by starting at the node v such that σ_v is the context of the previous block concatenated with the previous block, and tracing down the path with string $T[i:i+\beta-1]$; if the path ends at a node, we follow the suffix link of that node and continue tracing the remainder of the string.

Step 2. Consider each string t of length β for which a node v exists such that σ_v is precisely $\sigma_u||t$. Recall that associated with node v is the largest position in T at which the string σ_v appears is given by $pos(\sigma_v)$. We determine the rank of $pos(\sigma_u||B_i)$ amongst $pos(\sigma_u||t)$ for all such strings t; since σ_u is the context of B_i, block B_i is one of the strings t for which v exists and therefore $pos(\sigma_u||B_i)$ is well-defined. The compression algorithm outputs the pair $(|\sigma_u|, rank(pos(B_i)))$.

This step is done by starting at node u (recall that σ_u is the context of B_i), and performing a depth-first-search of the tree rooted at u. However, we only need to go down upto paths of length at most β. Whenever we reach a node w that traces a path of length β from u, we obtain $pos(\sigma_w)$. One such value obtained is for $\sigma_w = \sigma_u||B_i$ and we find its rank in the set of all values obtained.

Step 3. We insert the symbols of the current block in to our data structure; the symbols are inserted one after another left to right. For each symbol $T[j]$, we trace the path of the string $T[j-\alpha-\beta-1:j-1]$ in the trie; say the node thus reached is v. We consider the *envelope* of v consisting of nodes reached by any sequence of suffix links starting at v; note that the envelope consists of at most $\alpha+\beta$ nodes. We consider the nodes w_i on the envelope in the increasing order of $|\sigma_w|$. If a node w_i on the envelope has a child with character $T[j]$, we simply update $pos(\sigma_{w_i})$ appropriately; otherwise, we create such a child of w_i and assign the appropriate pos value to it. If a child of w_i is created, its suffix link points to the child of w_{i+1} with symbol $T[j]$, and this is initialized too.

The complexity of our algorithm is summarized below; the analysis is omitted here.

Theorem. *The compression algorithm takes time $O(n(\frac{2^\beta}{\beta}+\alpha))$. When $\beta = O(1)$, it follows that $\alpha = O(1)$ as well, and the running time is $O(n)$. When $\beta = \log_2 \log n$ and $\alpha = O(\log n)$, the running time is $O(n \log n)$.*

Preliminary Experimental Evaluation. Our algorithm gives an efficient implementation which we use to compress files from the Caterbury corpus benchmark suite. [1] Our initial results seem promising. For example, we obtain nearly 6% improvement in compression of a book of size 780K over WinZip utility. The compression was done with $\alpha = 5$ and $\beta = 1$ on ascii alphabet. In the table below the compression (in percent) is the amount of file that was compressed, so larger numbers indicate more compression. We demonstrate results of a two step compression scheme in which the first step is the HZY scheme and the second step is arithmetic coding; we also note our (percent) improvement over WinZip.

File	Size	HZY	HZY+AC	WinZip
bib	117541	71.4	71.2	+2.4
Book1	785393	64.9	64.7	+5.9
Book2	626490	70.3	70.1	+4.3
Pic	513216	85.4	84.7	-3.6
Trans	94487	78.0	77.8	-1.0
Progc	41098	67.5	66.7	+0.5
Progl	73890	75.9	75.5	-2.1
geo	102400	31.7	28.3	-1.3

Concluding Remarks. An interesting variation of the HZY scheme is lexicographic (rather than temporal) ranking of the contexts, for which efficient algorithms were provided in [SMZ98]. This variant is simpler than temporal ranking, but it is an open question whether its entropic properties will match up to that of the temporal ranking.

Context based compression methods (such as PPM and its variants) are known to achieve better compression ratios in practice than the Lempel-Ziv methods, which are used in most practical compression software. However, there has been no thorough analysis of such schemes to quantify the improved compression, and they are considerably slow. The HZY scheme is a context based method which has already been analyzed, and as we show, efficient implementations are possible. It remains to perform a thorough experimental study.

References

[HZ98] Y. Hershkovits and J. Ziv. On sliding window universal data compression with limited memory. *IEEE Transactions on Information Theory*, 1998.

[SMZ98] Y. Matias S. C. Sahinalp S. Muthukrishnan and J. Ziv. Augmenting suffix trees with applications. In *European Symposium on Algorithms*, 1998.

[Yok95] H. Yokoo. An adaptive data compression method based on context sorting. In *IEEE Data Compression Conference*, 1995.

[1]Here *bib* is a bibliography file, *book1* and *book2* are fiction and non-fiction books respectively, *geo* is geophysical data, *pic* is a black and white fax picture, *progc, progp, progl* are source codes in *C, Pascal, Lisp* respectively, and *trans* is transcript of a terminal session.