An Interesting Hierarchical Lossless Data Compression Algorithm

Greg Nelson, John Kieffer, and Pamela Cosman

Elec. Engr. Dept., University of Minnesota, Minneapolis, MN 55455, USA

Abstract — **We represent a data sequence of finite length hierarchically as a list of strings obtained by making parallel substitutions. We then encode the sequence by encoding the substitutions. The resulting hierarchical lossless data compression algorithm can be shown to outperform any finite-state lossless sequential data compression algorithm asymptotically as the length of the data sequence tends to infinity.**

I. Hierarchical Data Representation

Let A be a finite set of size $\alpha \geq 2$, and let x be a generic notation for a data sequence of length $|x| < \infty$ that can be formed from the symbols in A. The terminology *string* shall refer to a sequence of finite length whose entries can come from any symbol set. If $s = (s_1, s_2, \ldots, s_m)$ and $t = (t_1, t_2, \ldots, t_n)$ are strings, we write $s \sim t$ if $m = n$ and there is a one-to-one map β such that $\beta(s_i) = \beta(t_i)$ for each i, we write $s \to t$ if $n = 2m$ and $s \sim ((t_1, t_2), \ldots, (t_{2m-1}, t_{2m}))$, and we write st to denote the string $(s_1, \ldots, s_m, t_1, \ldots, t_n)$. Throughout this paper, logarithms shall be to base two.

To compress a data sequence x, we shall employ a hierarchical representation of x consisting of strings $s_1, s_2, \ldots, s_k$ in which

(1.1) $k = \lceil \log(|x| + 1) \rceil$.
(1.2) $|s_i| = 2^{i-1} \ \ \forall i$;
(1.3) $s_1 \to s_2 \to \ldots \to s_k$;
(1.4) The entries of s_i $(i < k)$ are positive integers;
(1.5) Letting $s_k = (u_1, u_2, \ldots, u_r)$, where $r = 2^{k-1}$, the entries u_i belong to $A \cup A^2$, and $x = u_1 u_2 \ldots u_r$.

We obtain this representation by means of a mapping (called the *bisection algorithm*) which maps each data sequence x into the pair (x_1, x_2) in which

(1.6) $x = x_1 x_2$, and
(1.7) $|x_1| = |x_2|$ if $|x|$ is even, and $|x_1| = |x_2| + 1$ if $|x|$ is odd.

Example 1. Let $A = \{\sigma, \tau\}$ and let $x = \sigma\sigma\sigma\tau\tau\sigma\sigma\sigma\tau\sigma$. Applying the bisection algorithm repeatedly, we obtain six bisections

$x_1 \to (x_2, x_3)$, where $x_1 = x,\ x_2 = \sigma\sigma\sigma\tau\tau,\ x_3 = \sigma\sigma\sigma\tau\sigma$.
$x_2 \to (x_4, x_5)$, where $x_4 = \sigma\sigma\sigma,\ x_5 = \tau\tau$.
$x_3 \to (x_4, x_6)$, where $x_6 = \tau\sigma$.
$x_4 \to (\sigma\sigma, \sigma)$.
$x_5 \to (\tau, \tau)$.
$x_6 \to (\tau, \sigma)$.

Replacing each x_i by its index i, we obtain the hierarchical representation s_1, s_2, s_3, s_4 of x in which

$s_1 = 1$
$s_2 = (2, 3)$
$s_3 = (4, 5, 4, 6)$
$s_4 = (\sigma\sigma, \sigma, \tau, \tau, \sigma\sigma, \sigma, \tau, \sigma)$

Each string s_2, s_3, s_4 is obtained from the previous one by means of parallel substitutions. For example, to go from s_3 to s_4, one simultaneously substitutes the pairs $(\sigma\sigma, \sigma)$, (τ, τ), (τ, σ) for the symbols $4, 5, 6$ in s_3, respectively. These substitutions correspond to the last three bisections in our earlier list of six bisections.

II. Definitions

(2.1) Let $s = (u_1, u_2, \ldots, u_n)$ be a string. Let $T = \{v : v = u_i \text{ for some } i\}$. Let $v_1, v_2, \ldots, v_r$ be the elements of T, listed in order of their first appearance in s from left to right. Define $\mathrm{emp}(s)$, the *empirical* of s, to be the vector $[n_1, n_2, \ldots, n_r]$ in which n_i is the number of times v_i appears in s. (For example, the empirical of the string $(2, 1, 3, 3, 1, 3, 1, 1)$ is $[1, 4, 3]$.)

(2.2) Define EMP to be the set of all vectors with positive integer components adding up to a power of two. We point out that all the empiricals of all the strings appearing in our hierarchical representations belong to EMP.

(2.3) Let s be a string and let $\pi = [n_1, n_2, \ldots, n_r] \in \mathrm{EMP}$. Form the list L consisting of the lexicographical ordering of all strings t formed from $\{1, 2, \ldots, r\}$ satisfying $s \to t$ and $\mathrm{emp}(t) = \pi$. Define $L(s, \pi)$ to be the list obtained from L by removing from L all t for which there exists $t' \sim t$ preceding t in L. To illustrate, for s_3 in Example 1, and $\pi = [2, 3, 3]$,

$$L(s_3, \pi) = \{12231233, 12321233, 12331223, 12331232\}.$$

(2.4) Let s, t be strings such that $s \to t$. Let t' be the unique member of $L(s, \mathrm{emp}(t))$ such that $t' \sim t$. Define $\mathrm{ind}(t|s)$ to be the index of t' in $L(s, \mathrm{emp}(t))$. To illustrate, for s_3, s_4 in Example 1, $\mathrm{ind}(s_4|s_3) = 4$ because $12331232 \sim s_4$ and 12331232 is the fourth member of the list $L(s_3, \mathrm{emp}(s_4))$.

III. Hierarchical Compression

Let $\{0, 1\}^*$ be the set of all binary sequences of finite length. Let $\psi : EMP \to \{0, 1\}^*$ and $\phi : \{1, 2, 3, \ldots\} \to \{0, 1\}^*$ be one-to-one mappings which map their respective domains onto a subset of $\{0, 1\}^*$ satisfying the prefix condition. We place the following requirements on ψ, ϕ.

(3.1) Let x be a data sequence and let $s_1, s_2, \ldots, s_k$ be the hierarchical representation of x. We require that

$$\sum_{i=2}^{k} |\psi(\mathrm{emp}(s_i)| = o(|x|)$$

(3.2) We require that $|\phi(n)| = \log n + o(\log n)$.

There is a natural way to define ψ so that (3.1) is valid, but we shall not discuss this here. It is well-known how to select ϕ so that (3.2) is valid. (One can use, for example, the Elias code.)

We describe a lossless data compression algorithm, consisting of encoder and decoder, which employs the hierarchical data representation described in Section 1. Let x be a data sequence and let $s_1, s_2, \ldots, s_k$ be the hierarchical representation of x. One losslessly encodes x into a binary sequence

$$b_1 b_2 \ldots b_k,$$

where

$$b_i = \psi(\mathrm{emp}(s_{i+1}))\phi(\mathrm{ind}(s_{i+1}|s_i)),\ i < k,$$

and b_k is a binary sequence of length at most $(\alpha^2+\alpha)\lceil\log(\alpha^2+\alpha)\rceil$. Using the sequences b_i, $i < k$, the decoder reconstructs strings $\hat{s}_1 \sim s_1, \hat{s}_2 \sim s_2, \ldots, \hat{s}_k \sim s_k$. Using the sequence b_k, the decoder reconstructs s_k (and subsequently x) from $\hat{s}_k$. The following example illustrates the encoding and decoding process.

Example 2. Let x be the data sequence considered in Example 1, and let s_1, s_2, s_3, s_4 be its hierarchical representation. The empiricals of s_2, s_3, s_4 are $[1,1], [2,1,1], [2,3,3]$, respectively. In the encoding process, one employs the following lists.

$L(s_1,[1,1]) = \{12\}$.
$L(s_2,[2,1,1]) = \{1123, 1213, 1231\}$.
$L(s_3,[2,3,3]) = \{12231233, 12321233, 12331223, 12331232\}$.

Define $\hat{s}_1 = 1,\ \hat{s}_2 = 12,\ \hat{s}_3 = 1213,\ \hat{s}_4 = 12331232$. Note that $\hat{s}_i \sim s_i$ for each i. For $i > 1$, $\hat{s}_i$ is characterized as the unique member of $L(s_{i-1}, \mathrm{emp}(s_i))$ such that $\hat{s}_i \sim s_i$. This allows us to compute indices as follows.

$$\mathrm{ind}(s_2|s_1) = 1,\ \mathrm{ind}(s_3|s_2) = 2,\ \mathrm{ind}(s_4|s_3) = 4.$$

For example, $\mathrm{ind}(s_3|s_2) = 2$ because $\hat{s}_3$ is the second term in the list $L(s_2, \mathrm{emp}(s_3))$. The encoder encodes x into

$$\psi([1,1])\phi(1)\psi([2,1,1])\phi(2)\psi([2,3,3])\phi(4)010000001.$$

Reading from left to right, the decoder first constructs the list $L(s_1,[1,1]) = L(\hat{s}_1,[1,1])$ and then determines $\hat{s}_2$ as the first member of this list. Then, $\hat{s}_3$ is computed as the second member of the list $L(s_2,[2,1,1]) = L(\hat{s}_2,[2,1,1])$, and $\hat{s}_4$ is computed as the fourth member of the list $L(s_3,[2,3,3]) = L(\hat{s}_3,[2,3,3])$. The sequence 010000001 is parsed into the 3-bit blocks 010, 000, 001, which are then converted into the integers 2, 0, 1. Adding one to each of these integers, the decoder chooses the third, first, and second members of the list $\{\sigma, \tau, \sigma\sigma, \sigma\tau, \tau\sigma, \tau\tau\}$. This tells the decoder that the parallel substitutions $1 \to \sigma\sigma$, $2 \to \sigma$, $3 \to \tau$ are made in the string $\hat{s}_4 = 12331232$ to obtain the string $s_4 = (\sigma\sigma, \sigma, \tau, \tau, \sigma\sigma, \sigma, \tau, \sigma)$. The decoder now knows that $x = (\sigma, \sigma, \sigma, \tau, \tau, \sigma, \sigma, \sigma, \tau, \sigma)$.

IV. Results

First Result. Finite-state lossless sequential data compression algorithms form a class of codes that have been studied since the 1970's. Our first result relates the performance of the hierarchical compression algorithm of Section III to the performance of finite-state sequential algorithms. Let $L(x)$ be the length of the codeword into which the data sequence x is encoded by the hierarchical compression algorithm. If s is a positive integer, let $L_s(x)$ be the minimal length of all codewords into which x is encoded via finite-state lossless sequential data compression algorithms describable using no more than s states. Then, we can prove the following two statements.

$$\text{(4.1)} \qquad \limsup_{|x|\to\infty} \frac{L(x)}{L_s(x)} \le 1,\ \ \forall s.$$

$$\text{(4.2)} \qquad \liminf_{|x|\to\infty} \frac{L(x)}{L_s(x)} = 0,\ \ \forall s.$$

Statements (4.1)-(4.2) tell us that the hierarchical compression algorithm of Section III outperforms the class of finite-state lossless sequential compression algorithms, asymptotically as the length of the data sequence goes to infinity. The proofs of these statements shall be given elsewhere. However, we give the reader an idea of the thinking that went into the proof of statement (4.1). For each positive integer n, one partitions the set A^n into subsets of form $\{x \in A^n : \mathrm{emp}(s_i) = \pi_i,\ i = 2, \ldots, k\}$, where $\{\pi_i : i = 2, \ldots, k\}$ is a sequence of empiricals. Then, one shows that the variation of $L_s(x)$ for x in such a set is $o(n)$.

Second result. Let $X_1, X_2, \ldots$ be an infinite sequence of random variables taking their values in the set A. For each positive integer n, let $S_{1,n}, S_{2,n}, \ldots, S_{k,n}$ denote the hierarchical representation of $(X_1, X_2, \ldots, X_n)$. While each of the strings in this representation is random, we point out that the number of strings k is not a random variable; in fact, $k = \lceil\log(n+1)\rceil$. Our second result consists of the following statement.

(4.3) Suppose the random sequence $X_1, X_2, \ldots$ is stationary and ergodic, and let H be the entropy rate of the sequence. Then

$$\sum_{i=1}^{k-1} \log\{\mathrm{ind}(S_{i+1,n}|S_{i,n})\} = nH + o(n) \text{ almost surely.}$$

It is helpful to discuss the implications of statement (4.3) for hierarchical compression. In compressing the first n samples of a stationary, ergodic random sequence, source coding theory tells us that the best we can do is to achieve a codeword length of about nH bits. In hierarchical compression, the goal should be to allocate a portion of these nH codebits to the encoder at each level of the hierarchical data representation, and require that each of these encoders yield a number of codebits approximately equal to its allocated portion. For the hierarchical compression algorithm of this paper, one can interpret the number $Q_i = \lceil\log\{\mathrm{ind}(S_{i+1,n}|S_{i,n})\}\rceil$ to be the number of codebits allocated to the encoder at the i-th level. This encoder has to encode both the integer $\mathrm{ind}(S_{i+1,n}|S_{i,n})$ and the empirical $\mathrm{emp}(S_{i+1,n})$; however, as it can be shown that the total number of codebits needed to encode the empiricals at the different levels is $o(n)$, the i-th level encoder is indeed successful in generating a number of codebits approximately equal to Q_i.

Open Problems. For simplicity, we described the encoder/decoder operation of the hierarchical compression algorithm in terms of lists; however, the construction of these lists is computationally inefficient. One can implement the algorithm more efficiently. However, a computational complexity characterization of the most efficient implementations still eludes us. Another open problem is to determine whether the hierarchical compression algorithm outperforms some interesting class of compression algorithms larger than that of the finite-state sequential algorithms.

Acknowledgement. The authors thank Dr. En-hui Yang for helpful discussions concerning this research.

This work was supported by NSF Grant NCR-9304984 and by an NSF postdoctoral fellowship.