Compressing Sets and Multisets of Sequences

Christian Steinruecken
University of Cambridge

Abstract

This article describes lossless compression algorithms for multisets of sequences, taking advantage of the multiset's unordered structure. Multisets are a generalisation of sets where members are allowed to occur multiple times. A multiset can be encoded naïvely by simply storing its elements in some sequential order, but then information is wasted on the ordering. We propose a technique that transforms the multiset into an order-invariant tree representation, and derive an arithmetic code that optimally compresses the tree. Our method achieves compression even if the sequences in the multiset are individually incompressible (such as cryptographic hash sums). The algorithm is demonstrated practically by compressing collections of SHA-1 hash sums, and multisets of arbitrary, individually encodable objects.

1 Introduction

This article describes a compression algorithm for multisets of binary sequences that exploits the disordered nature of the multisets.

Consider a collection $\mathcal{W}$ of N words $\{w_1 \ldots w_N\}$, each composed of a finite sequence of symbols. The members of $\mathcal{W}$ have no particular ordering (the labels w_n are used here just to describe the method); such collections occur in e.g. *bag-of-words* models. The goal is to compress this collection in such a way that no information is wasted on the ordering of the words.

Making an order-invariant representation of $\mathcal{W}$ could be as easy as arranging the words in some sorted order: if both the sender and receiver use the same ordering, zero probability could be given to all words whose appearance violates the agreed order, reallocating the excluded probability mass to words that remain compatible with the ordering. However, the correct probability for the next element in a sorted sequence is expensive to compute, making this approach unappealing.

It may seem surprising at first that a collection of strings can be compressed in a way that does *not* involve encoding or decoding the strings in any particular order. The solution presented in this article is to store them "all at once" by transforming the collection to an order-invariant tree representation, deriving an adaptive probabilistic model for this representation, and then compressing the tree using the model.

An example of this technique is presented for collections of sequences that are independently and identically distributed. The resulting compressing method is demonstrated practically for two applications: (1) compressing collections of SHA-1 sums; and (2) compressing collections of arbitrary, individually encodable objects.

This is not the first time order-invariant source coding methods have been considered. The *bits-back coding* approach puts wasted bandwidth to good use by filling it up with *additional data* [1, 2, 3]. However, it does not solve the problem of compactly encoding only the desired object. Much more generally, Varshney and Goyal [4, 5, 6] motivate a source

coding theory for compressing sets and multisets. Reznik [7] gives a concrete algorithm for compressing *sets* of sequences, also with a tree as latent representation, using an enumerative code [8, 9] for compressing the tree shape. Noting that Reznik's construction isn't fully order-invariant, Gripon et al. [10] propose a slightly more general tree-based coding scheme for multisets.

Our paper offers a different approach: we derive the exact distribution over multisets from the distribution over source sequences, and factorise it into conditional univariate distributions that can be encoded with an arithmetic coder. We also give an adaptive, universal code for the case where the exact distribution over sequences is unknown.

2 Collections of fixed-length binary sequences

Suppose we want to store a multiset of fixed length binary strings, for example a collection of hash sums. The SHA-1 algorithm [11] is a file hashing method which, given any input file, produces a rapidly computable, cryptographic hash sum whose length is exactly 160 bits. Cryptographic hashing algorithms are designed to make it computationally infeasible to change an input file without also changing its hash sum. Individual hash sums can be used, for example, to detect if a previously hashed file has been modified (with negligible probability of error), and collections of hash sums can be used to detect if a given file is one of a preselected collection of input files.[1]

Each bit digit in a random SHA-1 hash sum is uniformly distributed, which renders single SHA-1 sums incompressible. It might therefore seem intuitive at first that storing N hash sums would cost exactly N times as much as storing one hash sum. However, an *unordered* collection of SHA-1 sums can in fact be stored more compactly. The potential saving for a collection of N random hash sums is roughly $\log_2 N!$ bits. For example, the practical savings for a collection of 5000 SHA-1 sums amount to 10 bits per SHA-1 sum, i.e. each SHA-1 sum in the collection takes only 150 bits of space (rather than 160 bits).

A concrete method for compressing multisets of fixed-length bit strings (such as collections of SHA-1 sums) is described below. The algorithm makes use of arithmetic coding to encode values from binomial and Beta-binomial distributions; details are described in appendices A and B.

2.1 Tree representation for multisets of fixed-length strings

A multiset of binary sequences can be represented with a binary tree whose nodes store positive integers. Each node in the binary tree partitions the multiset of sequences into two submultisets: those sequences whose next symbol is a `0`, and those whose next symbol is a `1`. The integer count n stored in the root node represents the total size of the multiset, and the counts n_0, n_1 stored in the child nodes indicate the sizes of their submultisets. An example of such a tree and its corresponding multiset is shown in Figure 1.

To save space, nodes with zero counts may be omitted from the tree. For a multiset of fixed-length sequences, sequence termination is indicated by a leaf node, or a node that only

[1] If an application cares mainly about testing membership in a collection, even more compact methods exist, for example Bloom filters [12]. Bloom filters are appropriate when a not-so-negligible chance of false positives is acceptable.

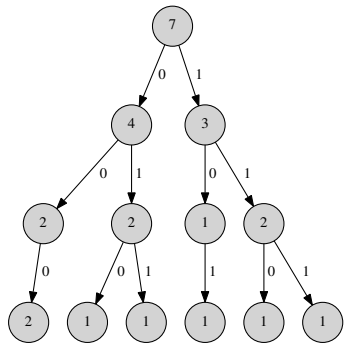


Figure 1: The binary tree representing the multiset:

{ 000, 000, 010, 011, 101, 110, 111 }.

The count at each node indicates the number of strings starting with the node's prefix: e.g. there are 7 strings starting with the empty string, 4 strings starting with 0, 3 strings starting with 1, and 2 strings starting with 01.

has children with a count of zero. The sequence of branching decisions taken to reach any given node from the root is called the node's *prefix*. To recover the original multiset from the tree, it suffices to collect the prefix of each leaf node, including each prefix as many times as indicated by the leaf node's count.

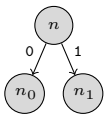
A binary tree as described above is unique for any given collection of binary strings. The tree can be constructed incrementally, and supports addition, deletion and membership testing of sequences in $O(L)$ time, where L is the sequence length. Merging two trees can be done more efficiently than adding one tree's sequences to the other individually: the counts of nodes whose prefixes are equal can simply be added, and branches missing from one tree can be copied (or moved) from the other tree. Extracting N sequences from the tree, either lexicographically or uniformly at random, takes $O(L \cdot N)$ time.

2.2 Fixed-depth multiset tree compression algorithm

The previous section showed how a multiset of N binary sequences of fixed length L can be converted to a tree representation. This section derives exact conditional probability distributions for the node counts in the resulting tree, and shows how the tree can be compactly encoded with an arithmetic coder.

Suppose that N and L are known in advance. With the exception of the leaf nodes, the count n at any given node in the tree equals the sum of the counts of its children, i.e. $n = n_0 + n_1$. If the bits of each string are independent and identically distributed, the counts of the child nodes (conditional on their parent's count) jointly follow a binomial distribution:

$$\begin{aligned} n_1 &\sim \text{Binomial}(n,\ \theta) \\ n_0 &= n - n_1 \end{aligned} \qquad (1)$$



where θ is the probability of symbol 1. If the symbols 0 and 1 are uniformly distributed (as is the case for SHA-1 sums), θ should be set to $\frac{1}{2}$. Given the parent count n, only one of the child counts needs to be communicated, as the other can be determined by subtraction from n. Since all strings in the multiset have length L, all the leaf nodes in the tree are located at depth L, making it unnecessary to communicate which of the nodes are leaves.

If N and L are known, the tree can be communicated as follows: Traverse the tree, except for the leaf nodes, starting from the root (whose count N is already known). Encode one of child counts (e.g. n_1) using a binomial code and recurse on all child nodes whose count is greater than zero. The parameters of the binomial code are the count of the parent, and the

Coding algorithm for multisets of fixed-length sequences

ENCODING

Inputs: L, binary tree T

A. Encode N, the count of T's root node, using a code over positive integers.

B. Call `encode_node`(T).

subroutine `encode_node`(t):

If node t is a leaf:

1. Return.

Otherwise:

1. Let t_0 and t_1 denote the children of t, and n_0 and n_1 the children's counts.
2. Encode n_1 using a binomial code, as $n_1 \sim \text{Binomial}(n_0 + n_1,\ \theta)$.
3. If $n_0 > 0$, call `encode_node`(t_0).
4. If $n_1 > 0$, call `encode_node`(t_1).

DECODING

Input: L **Output:** binary tree T

A. Decode N, using the same code over positive integers.

B. Return $T \leftarrow$ `decode_node`(N, L).

subroutine `decode_node`(n, l):

If $l > 0$ then:

1. Decode n_1 using a binomial code, as $n_1 \sim \text{Binomial}(n,\ \theta)$.
2. Recover $n_0 \leftarrow (n - n_1)$.
3. If $n_0 > 0$, then:
   $t_0 \leftarrow$ `decode_node`$(n_0, l - 1)$.
4. If $n_1 > 0$, then:
   $t_1 \leftarrow$ `decode_node`$(n_1, l - 1)$.
5. Return a new tree node with count n and children t_0 and t_1.

Otherwise, return null.

Algorithm 1: Coding algorithm for binary trees representing multisets of binary sequences of length L. The form and construction of the binary tree are described in section 2.1. Each tree node t contains an integer count n and two child pointers t_0 and t_1. The counts of the children are written n_0 and n_1. If n_0 and n_1 are zero, t is deemed to be a leaf, and vice versa. T denotes the tree's root node.

symbol bias θ, as shown in equation (1). The tree can be traversed in any order that visits parents before their children.

This encoding process is invertible, allowing perfect recovery of the tree. The same traversal order must be followed, and both N and L must be known (to recover the root node's count, and to determine which nodes are leaf nodes). Depending on the application, N or L can be transmitted first using an appropriate code over integers. A concrete coding procedure using pre-order traversal is shown in Algorithm 1.

Application to SHA-1 sums. For a collection of N SHA-1 sums, the depth of the binary tree is $L = 160$, and the root node contains the integer N. If the SHA-1 sums in the collection are random, the distribution over the individual bits in each sequence is uniform, making a binomial code with bias $\theta = \frac{1}{2}$ an optimal choice. However, if the collection is expected to contain duplicate entries at a rate greater than chance, the distribution over the counts is no longer binomial with a fixed bias; in fact, the bias might then be different for each node in the tree. In such a case, a Beta-binomial code may be more appropriate, as it can *learn* the underlying symbol probability θ independently for each node, rather than

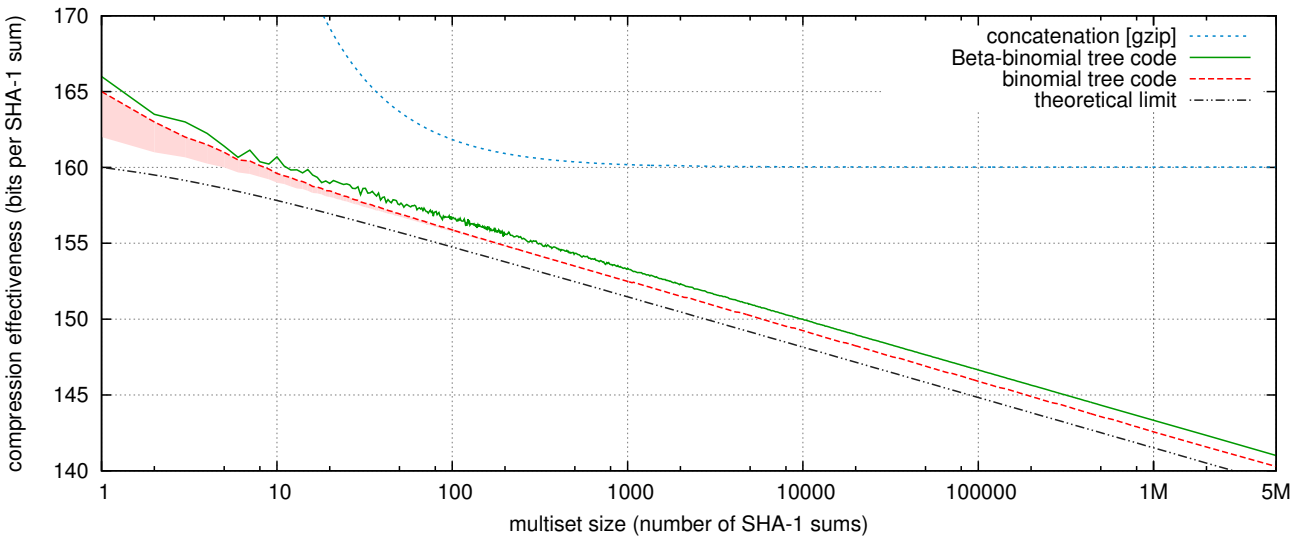


Figure 2: Practical lossless compression performance of the fixed-depth multiset tree compressor on multisets of SHA-1 sums. For each position on the x-axis, N uniformly distributed 64-bit random numbers were generated and hashed with SHA-1; the resulting multiset of N SHA-1 sums was then compressed with each algorithm. The winning compression method is Algorithm 1 using a binomial code, where N itself is encoded with a Fibonacci code. The shaded region indicates the proportion of information used by the Fibonacci code. The theoretical limit is $160 - \frac{1}{N}\log_2 N!$ bits, assuming N is known to the receiver. For comparison, `gzip` was used to compress the concatenation of the N SHA-1 sums; reaching, as expected, 160 bits per SHA-1 sum.

assuming it to have a particular fixed value:

$$\begin{aligned} n_1 &\sim \text{BetaBin}(n, \alpha, \beta) \\ n_0 &= n - n_1 \end{aligned} \tag{2}$$

The tree coding method of Algorithm 1 can be modified to use a Beta-binomial code by replacing the encoding and decoding calls in the subroutine accordingly. In our experiments, the Beta-binomial parameters were set to $\alpha = \frac{1}{2}$ and $\beta = \frac{1}{2}$.

The practical performance of the algorithm on multisets of SHA-1 sums is shown in Figure 2. The multisets used in this experiment contain no duplicate hashes, so the compression achieved by the algorithm really results from exploiting the permutation invariance of the multiset rather than any redundancy among the hashes.

3 Collections of binary sequences of arbitrary length

This section generalises the tree coding method to admit binary sequences of arbitrary length. Two approaches are considered for encoding the termination of sequences in the tree: the first approach covers collections of self-delimiting sequences, which allow the tree to be compressed without encoding additional information about termination. The second approach, for arbitrary sequences, assumes a *distribution* over sequence lengths and encodes sequence termination directly in the tree nodes. For either approach, the same binary tree structure is used as before, except that sequences stored in the tree can now have any length.

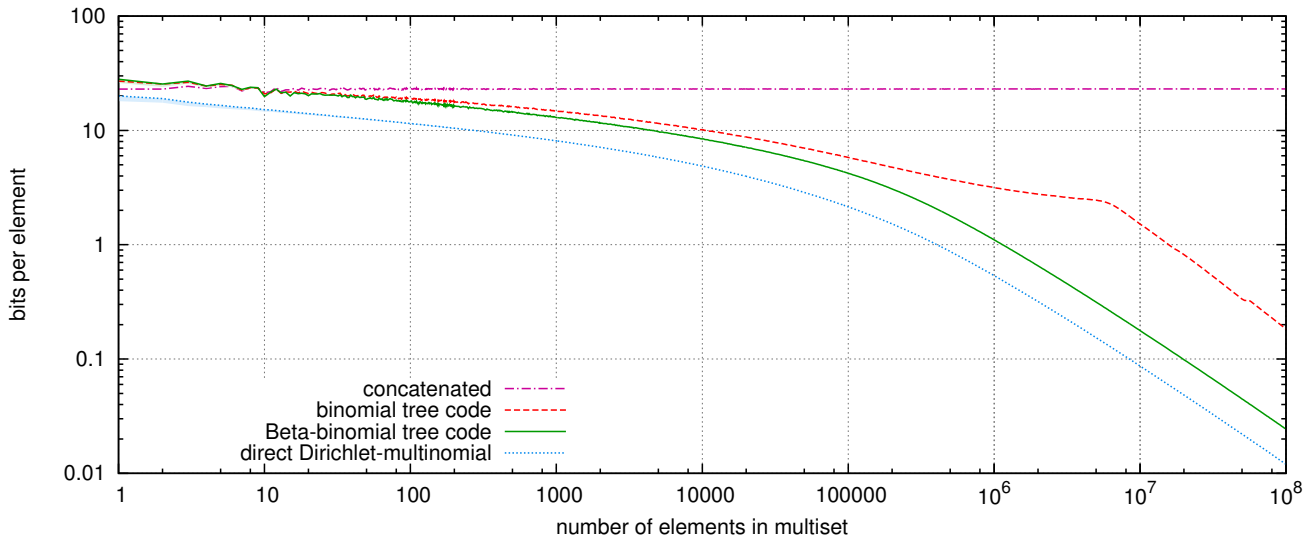


Figure 3: Experimental compression performance of various algorithms on multisets of self-delimiting sequences. For each position on the x-axis, a multiset of N self-delimiting sequences was generated by taking N uniformly distributed integers between 1 and 100 000 and encoding each number with a Fibonacci code. The multiset of the resulting code words was then compressed with each algorithm. The y-axis shows the compressed size in bits divided by N. The flat concatenation of the sequences in the multiset is included for reference (achieving zero compression). For comparison, the source multisets of integers (rather than the multisets of Fibonacci-encoded integers) were compressed directly with a Dirichlet-multinomial code. The (barely visible) shaded regions indicate the amount of information taken up by the Fibonacci code to encode N itself.

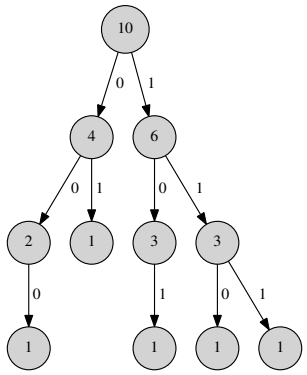
3.1 Compressing multisets of self-delimiting sequences

Self-delimiting sequences encode their own length, i.e. it can be determined from the sequence itself whether further symbols follow or the sequence has ended. Many existing compression algorithms produce self-delimiting sequences, e.g. the Huffman algorithm, codes for integers, or suitably defined arithmetic coding schemes. A multiset of such self-delimiting sequences has the property that, for any two distinct sequences in the multiset, neither can be a prefix of the other.

Consider the tree corresponding to such a multiset of binary strings. Because of the prefix property, all sequences in the tree will terminate at leaf nodes, and the counters stored in child nodes always add up to the counter of the parent node. Consequently, the same compression technique can be used as for fixed-length sequences. Algorithm 1 applies exactly as before, with the exception that the end-of-string detector in the decoder must be modified to detect the end of each self-delimiting sequence.

Compressing arbitrary multisets. Consider a random multiset $\mathcal{M}$ over an arbitrary space $\mathcal{X}$, whose elements can be independently compressed to self-delimiting binary strings (and reconstructed from them). Any such multiset $\mathcal{M}$ can be losslessly and reversibly converted to a multiset $\mathcal{W}$ of self-delimiting sequences, and $\mathcal{W}$ can be compressed and decompressed with the tree coding method as described above.

Alternative. A random multiset $\mathcal{M}$ is most effectively compressed with a compression algorithm that exactly matches $\mathcal{M}$'s probability distribution; we'll call such an algorithm a



The binary tree representing the multiset:

{ 0, 00, 000, 01, 10, 10, 101, 11, 110, 111 }.

The count at each node indicates the number of strings starting with that node's prefix. For example, there are 10 strings starting with the empty string, 4 starting with 0, and 6 starting with 1, etc.
Out of the four strings starting with 0, two continue with 0, one continues with 1, and one reached its termination.
If the root node's count were 12 rather than 10, the multiset would include two empty strings as well.

Figure 4: Binary tree representing a multiset of ten binary sequences. This tree follows the same basic structure as the tree in Figure 1, but admits sequences of variable length. The tree representation is unique for each multiset.

direct code for $\mathcal{M}$. When a direct code is not available or convenient, the indirect method of first mapping $\mathcal{M}$ to $\mathcal{W}$ might be a suitable alternative.

Experiment. Experimental results of this approach on random multisets of self-delimiting sequences are shown in Figure 3. Each multiset was generated by drawing N uniform random integers and converting these integers to self-delimiting sequences with a Fibonacci code [13, 14].[2] The Beta-binomial variant of the tree coder wins over the binomial variant, and closely follows the trajectory of a Dirichlet-multinomial code for the underlying multisets of integers.

3.2 Encoding string termination via end-of-sequence markers

Consider now a multiset containing binary sequences of arbitrary length, whose sequences lack the property that their termination can be determined from a prefix. This is the most general case. In this scenario, it is possible for the multiset to contain strings where one is a prefix of the other, for example 01 and 011. To encode such a multiset, string termination must be communicated explicitly for each string. Luckily, the existing tree structure can be used as before to store such multisets; the only difference is that the count of a node need not equal the sum of the counts of its children, as terminations may now occur at any node, not just at leaf nodes. Both child counts therefore need to be communicated. An example of such a tree is shown in Figure 4.

The counter n stored in each node still indicates the number of sequences in the collection that start with that node's prefix. The number of terminations $n_{\mathtt{T}}$ at any given node equals the difference of the node's total count n and the sum of its child counts $n_{\mathtt{0}}$ and $n_{\mathtt{1}}$.

Suppose that the number $N = |\mathcal{W}|$ of sequences in the multiset $\mathcal{W}$ is distributed according to some distribution D over positive integers, and that the length of each sequence $w_n \in \mathcal{W}$ is distributed according to some distribution L. Given D and L, a near-optimal compression algorithm for the multiset $\mathcal{W}$ can be constructed as follows.

[2]The Fibonacci code was chosen for elegance. However, any code over integers could be used, e.g. an exponential Golomb code [15] or the ω-code by Elias [16].

First, form the tree representation of $\mathcal{W}$, following the construction described in the previous section. The count of the root node can be communicated using a code for D. Each node in the tree has a count n, child counts $n_{\mathtt{0}}$ and $n_{\mathtt{1}}$, and an implicit termination count $n_{\intercal}$ fulfilling $n = n_{\mathtt{0}} + n_{\mathtt{1}} + n_{\intercal}$. Assuming that the bits at the same position of each sequence are independent and identically distributed, the values of $n_{\mathtt{0}}$, $n_{\mathtt{1}}$ and $n_{\intercal}$ are multinomially distributed (given n).

The parameters of this multinomial distribution can be derived from L as follows: The n sequences described by the current node have a minimum length of d, where d is the node's depth in the tree (the root node is located at depth 0). Out of these n sequences, $n_{\mathtt{0}}$ continue with symbol `0`, $n_{\mathtt{1}}$ continue with symbol `1`, and $n_{\intercal}$ terminate here. Given the sequence length distribution L, the probability for a sequence that has at least d symbols to have no more than d symbols is given by a Bernoulli distribution with bias $\theta_{\intercal}(d)$, where:

$$\theta_{\intercal}(d) \;:=\; \frac{L(d)}{1 - \sum_{k<d} L(k)} \tag{3}$$

Consequently, the number of terminations $n_{\intercal}$ at depth d (out of n possible sequences) is binomially distributed with:

$$n_{\intercal} \;\sim\; \text{Binomial}(n,\; \theta_{\intercal}(d)) \tag{4}$$

Writing $\theta_{\intercal}$ for the probability of termination at the local node, and $\theta_{\mathtt{1}}$ and $\theta_{\mathtt{0}}$ for the occurrence probabilities of `1` and `0`, the joint distribution over $n_{\mathtt{0}}$, $n_{\mathtt{1}}$ and $n_{\intercal}$ can be written as follows:

$$(n_{\intercal}, n_{\mathtt{0}}, n_{\mathtt{1}}) \;\sim\; \text{Mult}(\theta_{\intercal},\; \theta_{\mathtt{0}}\,(1-\theta_{\intercal})\,,\; \theta_{\mathtt{1}}\,(1-\theta_{\intercal})) \tag{5}$$

where $\theta_{\mathtt{1}} = 1 - \theta_{\mathtt{0}}$. The encoding procedure for this tree needs to encode a ternary (rather than binary) choice, but the basic principle of operation remains the same. Algorithm 1 can be modified to encode $(n_{\intercal}, n_{\mathtt{0}}, n_{\mathtt{1}})$ using a multinomial code.

Note that, as described above, $\theta_{\intercal}$ is a function of the length distribution L and the current node depth d. In principle, it is possible to use a conditional length distribution that depends on the prefix of the node, as the node's prefix is available to both the encoder and the decoder. Similarly, $\theta_{\mathtt{0}}$ and $\theta_{\mathtt{1}}$ could in principle be functions of depth or prefix.

4 Conclusions

We proposed a novel and simple data compression algorithm for sets and multisets of sequences, and illustrated its use on collections of cryptographic hash sums. Our approach is based on the general principle that one should encode a permutation-invariant representation of the data, in this case a tree, with a code that matches the distribution induced by the data's generative process. When the distribution of the source sequences is known, the tree is optimally compressed with a nested binomial coding scheme; otherwise, a Beta-binomial coding scheme can be used. The Beta-binomial code is universal in that it learns the symbol distribution of the sequences in the multiset (even for symbol distributions that are position or prefix dependent).

One might regard the coding algorithms presented in this paper either as lossless compression for sets and multisets, or as lossy compression methods for lists: when the order of a list of elements isn't important, bandwidth can be saved.

Future work could address multisets of sequences whose elements are not independent and identically distributed, by combining the above approach with probabilistic models of the elements.

Appendix

A A binomial code

The binomial distribution describes the number of successes in a set of N independent Bernoulli trials. It is parametrised by natural number N and success probability θ, and ranges over positive integers $n \in \{0 \dots N\}$. A binomial random variable has the following probability mass function:

$$\text{Binomial}(n \mid N, \theta) = \binom{N}{n} \cdot \theta^n (1-\theta)^{N-n} \tag{6}$$

Encoding a binomial random variable with an arithmetic coder requires computing the cumulative distribution function of the binomial distribution. A method for doing this efficiently might utilise the following recurrence relation:

$$\text{Binomial}(n+1 \mid N, \theta) \;=\; \frac{N-n}{n+1} \cdot \frac{\theta}{1-\theta} \cdot \text{Binomial}(n \mid N, \theta) \tag{7}$$

The cumulative binomial distribution can then be computed as follows. Initialise $B_\Sigma \leftarrow 0$, and $B \leftarrow (1-\theta)^N$. To encode a binomially distributed value n, repeat for each k from $1 \dots n$:

$$B_\Sigma \;:=\; B_\Sigma + B \tag{8}$$

$$B \;:=\; \frac{N-k}{k+1} \cdot \frac{\theta}{1-\theta} \cdot B \tag{9}$$

The interval $[B_\Sigma,\ B_\Sigma + B)$ is then a representation of n that can be used with an arithmetic coder.

B A Beta-binomial code

The Beta-binomial compound distribution results from integrating out the success parameter θ of a binomial distribution, assuming θ is Beta distributed. It is parametrised by an integer N and the parameters α and β of the Beta prior:

$$\text{BetaBin}(n \mid N, \alpha, \beta) \;=\; \int \text{Binomial}(n \mid N, \theta) \cdot \text{Beta}(\theta \mid \alpha, \beta) d\theta \tag{10}$$

$$= \binom{N}{n} \cdot \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \cdot \frac{\Gamma(\alpha+n)\Gamma(\beta+N-n)}{\Gamma(\alpha+\beta+N)} \tag{11}$$

Just like for the binomial distribution, there is a recurrence relation which can speed up the computation of the cumulative Beta-binomial distribution:

$$\text{BetaBin}(n+1 \mid N, \alpha, \beta) \;=\; \frac{N-n}{n+1} \cdot \frac{\alpha+n}{\beta+N-n-1} \cdot \text{BetaBin}(n \mid N, \alpha, \beta) \tag{12}$$

The method from appendix A can be modified accordingly, yielding a Beta-binomial coding scheme.

Acknowledgments

The author would like to thank David MacKay, Zoubin Ghahramani, Jossy Sayir and anonymous reviewers for helpful feedback. This work was supported by EPSRC Grant EP/I036575 and a Google Research Award.

References

[1] C. S. Wallace, "Classification by minimum-message-length inference," in *Advances in Computing and Information — ICCI '90*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1990, vol. 468, pp. 72–81.

[2] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and Helmholtz free energy," in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. Morgan Kaufmann, 1994.

[3] B. J. Frey and G. E. Hinton, "Efficient stochastic source coding and an application to a Bayesian network source model," *The Computer Journal*, vol. 40, no. 2, 3, pp. 157–165, 1997.

[4] L. R. Varshney and V. K. Goyal, "Ordered and disordered source coding," in *Proceedings of the Information Theory & Applications Inaugural Workshop*, Feb. 2006.

[5] ——, "Toward a source coding theory for sets," in *Proceedings of the Data Compression Conference*. IEEE Computer Society, 2006, pp. 13–22.

[6] ——, "On universal coding of unordered data," in *Information Theory and Applications Workshop 2007, Conference Proceedings*. IEEE, 2007, pp. 183–187.

[7] Y. A. Reznik, "Coding of sets of words," in *Proceedings of the Data Compression Conference*. IEEE Computer Society, 2011, pp. 43–52.

[8] S. Zaks, "Lexicographic generation of ordered trees," *Theoretical Computer Science*, vol. 10, no. 1, pp. 63–82, 1980.

[9] T. M. Cover, "Enumerative source encoding," *IEEE Transactions on Information Theory*, vol. 19, no. 1, pp. 73–77, Jan. 1973.

[10] V. Gripon, M. Rabbat, V. Skachek, and W. J. Gross, "Compressing multisets using tries," in *Information Theory Workshop (ITW)*. IEEE, 2012, pp. 642–646.

[11] NIST, "Secure hash standard," Federal Information Processing Standards (FIPS), Publication 180-1, Apr. 1995, Information Technology Laboratory, National Institute of Science and Technology (NIST), USA.

[12] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[13] W. H. Kautz, "Fibonacci codes for synchronization control," *IEEE Trans. Inf. Theory*, vol. 11, no. 2, pp. 284–292, Apr. 1965.

[14] A. Apostolico and A. S. Fraenkel, "Robust transmission of unbounded strings using Fibonacci representations," *IEEE Trans. Inf. Theory*, vol. 33, no. 2, pp. 238–245, Mar. 1987.

[15] J. Teuhola, "A compression method for clustered bit-vectors," *Inf. Process. Lett.*, vol. 7, pp. 308–311, 1978.

[16] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. Inf. Theory*, vol. 21, no. 2, pp. 194–203, Mar. 1975.