



# **Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity**

**S.C. Evans and S.F. Bush**

**2001CRD159, November 2001**

**Class 1**

**Technical Information Series**

Copyright © 2001 General Electric Company. All rights reserved.

## Corporate Research and Development

### Technical Report Abstract Page

**Title** Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity

**Author(s)** S.C. Evans **Phone** (518)387-7014  
S.F. Bush 8\*833-7014

**Component** Information and Decision Technologies

**Report Number** 2001CRD159 **Date** November 2001

**Number of Page** 11 **Class** 1

**Key Words** Kolmogorov Complexity, Compression, Minimum Message Length, Bio-informatics, Biotechnology, DNA sequence analysis

A new compression algorithm is derived that computes and encodes the Minimum Message Length (MML) near optimal partition of symbols in a string for compression. Using Symbol Compression Ratio (SCR) as a driving function this algorithm produces a binary tree model of the data that introduces a fundamental parameter of information related to Kolmogorov Complexity – the size of the alphabet in the near optimal partition.

Manuscript received October 25, 2001

Published at: <http://www.crd.ge.com/~bushsf/ftn>

# Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity

S.C. Evans and S.F. Bush  
<http://www.crd.ge.com/~bushsf/ftn>

## 1. Introduction

Kolmogorov Complexity ( $K(x)$ ) is the optimal compression bound of a given string  $x$ . This incomputable yet fundamental property of information has vast implications and applications in the areas of network and system optimization, security, bioinformatics, and emergence (see [1], [2] for an introduction to Kolmogorov Complexity and [2], [3] and [7] for some applications). An ideal approach for compression of information with a known distribution is Huffman Coding, which approaches the entropy of the source distribution. A useful notion of Kolmogorov Complexity is described according to Minimum Message Length or Minimum Data Length principles (MML) [4]. Under MML,  $K(x)$  is estimated as the smallest possible combination of model plus description of string under a model. Our approach is to combine these two notions to achieve a partition of a string into symbols that provides the near optimal compression among possible partitions. We introduce the Symbol Compression Ratio (SCR) as a test for an individual symbol's contribution to the overall compression ratio when evaluating partitions. Our new compression technique estimates Kolmogorov Complexity under the MML principle with a string modeled as the concatenation of a finite set of symbols. This technique not only provides a useful compression technique, but also results in a simple binary tree model of a string that can be used to generate a typical data set to which the string belongs. The size of the partition alphabet as output by this algorithm is put forward as a computable fundamental property of information that is related to the Kolmogorov Complexity of a string.

## 2. The Model

We consider compression of a finite binary string  $S$  of length  $L$ . We seek the optimal partition of  $S$  into  $I$  symbols that can be encoded using a near optimal encoding strategy such as Huffman coding such that the combination of the descriptive cost of the model (in this case the partition or definition of symbols) plus the encoding of the data under the model are minimized. The following parameters are defined:

**Table 1: Parameters**

Parameter	Meaning
$I$	Length of finite string S
$I$	Total number of symbols in partition $i \in [1, I]$
$l_i$	Length of symbol i
$r_i$	Number of repetitions of symbol i in S If repetitions for all symbols are equal then $r_i = r$
$R$	Total number of repetitions, $R = \sum_i r_i$
$L_i$	Length of S consumed by symbol I $L_i = l_i r_i$ , $L = \sum_i L_i$
$D_p$	Total Descriptive cost of S under partition p. Equal to the sum of the model description M plus the encoding of the data under the given model.
$d_i$	Descriptive Cost of Symbol i. This parameter will be derived in section 4
$\tilde{\lambda}_i$	Symbol Compression Ratio (SCR) $\tilde{\lambda}_i = \frac{d_i}{L_i}$

### 3. The Effect of a Partition on MML

The entropy of a distribution of symbols ( $H_s$ ) defines the average per symbol compression bound in bits per symbol for a prefix free code. For a distribution p of I symbols:

$$H_s = -\sum_i p_i \log_2(p).$$

Huffman coding and other strategies can produce an instantaneous code approaching the entropy when the distribution p is known. But what is the best encoding possible when the source distribution is not known? One way to proceed is to measure the empirical entropy of the string, that is the entropy defined inherently by the input string itself. However, empirical entropy is a function of the partition and depends on what sub-strings are grouped together to be considered symbols. See [6] for a consideration of some of the inadequacies of the well-known Lempel-Ziv algorithms in dealing with higher order empirical entropies.

Our goal is to optimize the partition (the number of symbols, their length, and distribution) of a string such that the compression bound for an instantaneous code, which is equal to  $R \cdot H_s$ , plus the codebook size is minimized according to the MML criteria. We

estimate the codebook size (model descriptive cost  $M$ ) to be the sum of the lengths of unique symbols:

$$M = \sum_i l_i .$$

Thus we estimate the total descriptive cost  $D_p$ :

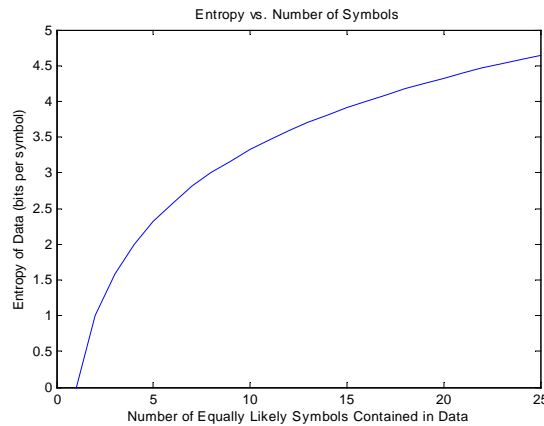
$$D_p = M + R \cdot H_s$$

Consider for now that all symbols are equally likely and of equal length. Thus:

$$r_i = \frac{R}{I} = r , l_i = \frac{L}{R} = l \text{ and } H_s = I \log_2(I)$$

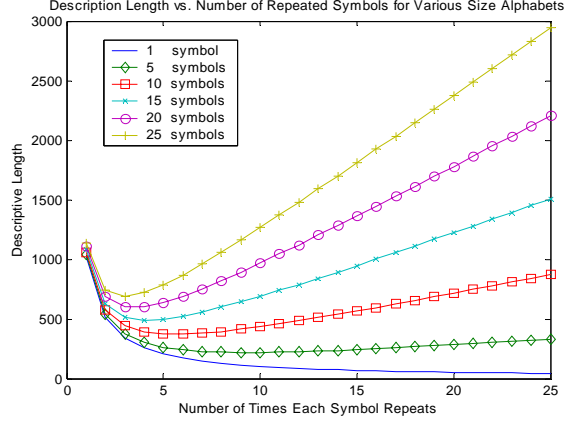
Figure 1 describes how entropy changes as unique symbols are added to the partition; each added symbol increasing the number of bits required to encode each symbol in a less than linear fashion. This increased descriptive cost per symbol must be traded against symbol length and number of repetitions.

For a given number of unique symbols, more repetitions will at first tend to decrease the overall description length, since the fixed length string of size  $L$  will now be divided into shorter words of size  $l$  and the codebook for the string will now be shorter to describe.



**Figure 1: More equally likely symbols in a partition cause the Entropy to increase – raising the bits per symbol descriptive cost in a less than linear manner.**

The description length decreases (the reduction in model size dominates) until a minimum occurs where the benefit from a decreased codebook size is offset by the fact that more symbols of a fixed average encoded length (on the order of  $H_s$ ) must be appended to the description. Figure 2 plots description length vs. number of repeats for various size equally likely alphabets based on a 1024 bit string. The knee in the curve for each number of symbols represents the optimal number of repetitions for a certain symbol alphabet size.



**Figure 2: Symbol length and number of repetitions of equal length equally likely symbols comprising a string of finite length produce competing affects in total string descriptive cost.**

As shown in Figure 2, the benefit of repeated patterns in fixed size data is overcome more quickly in a large equally likely alphabet. For the case where the entropy is less than one (i.e. there is only a single symbol) the benefit increases until all symbols are repeats, as expected. The above analysis shows that when partitioning a string both length of symbols and number of repetitions of symbols must be traded off and optimized in order to minimize descriptive length. We develop a method to treat non-uniform distributions in the next section.

#### 4. Symbol Compression Ratio

In seeking to partition the string so as to minimize the total string descriptive length  $D_p$ , we consider the length that the presence of each symbol adds to the total descriptive length and the amount of coverage of total string length  $L$  that it provides. As described in section 3, the descriptive cost of the model is on the order of the sum of the lengths of unique symbols in the partition. The descriptive cost of the encoded data is on the order of  $R \cdot H_s$ , where  $H_s$  is the entropy of the symbol partition. The probability of each symbol,  $p_i$ , is a function of the number of repetitions of each symbol:

$$p_i = \frac{r_i}{R}.$$

Thus, we have:

$$\begin{aligned} H_s &= -\sum_i p_i \log_2(p_i) = -\sum_i \frac{r_i}{R} \log_2\left(\frac{r_i}{R}\right) \\ &= \frac{-1}{R} \sum_i r_i [\log_2(r_i) - \log_2(R)] \end{aligned}$$

Since  $R = \sum_i r_i$ ,

$$H_s = \log_2(R) - \frac{1}{R} \sum_i r_i \log_2(r_i) .$$

$H_s$  is a measure of the ability to encode the distribution  $p$  of  $I$  symbols. The smaller  $H_s$  the fewer bits per symbol required to encode each symbol. For a fixed  $I$ ,  $H_s$  is maximized when all  $I$  symbols are equally likely.

Thus, descriptive length of the string under partition  $p$  is equal to:

$$D_p = R \log_2(R) + \sum_i l_i - r_i \log_2(r_i)$$

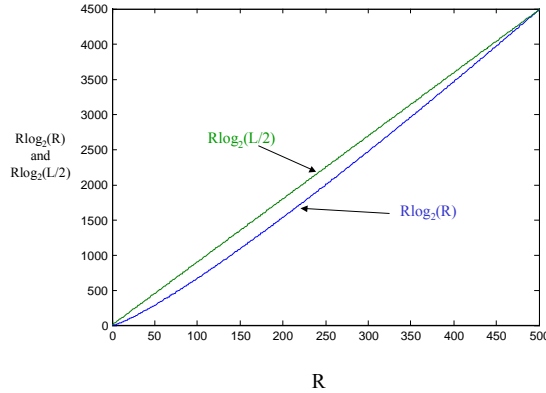
with

$$R \log_2(R) = \sum_i r_i \log_2(R) = c \sum_i r_i$$

where  $c$  is a constant estimating  $\log_2(R)$ . For  $R$  that can vary between 2 and  $L/2$  for symbols of size 2 bits or greater,  $\log_2(R)$  can be estimated to enable an incremental, per symbol formulation for  $D_p$ . Estimating  $c$ :

$$c = \log_2\left(\frac{L}{2}\right) ,$$

results in a conservative approximation for  $R \log_2(R)$  over the likely range of  $R$  as shown in Figure 3 for partitions of strings having length equal to 1000 bits.



**Figure 3: Estimate of  $R \log_2(R)$**

The per-symbol descriptive cost can now be formulated:

$$d_i = r_i [\log_2(L/2) - \log_2(r_i)] + l_i$$

$D_p$  is less than the sum of the individual code words due to the conservative approximation for  $\log_2(R)$ . A lower bound on the estimate for  $d$  can be formed as well to create upper and lower bounds on the descriptive length  $D_p$ :

$$d_{\min_i} = r_i \log_2(r_i) + l_i - r_i \log_2(r_i) = l_i$$

$$\sum_i d_{\min_i} \leq D_p \leq \sum_i d_i$$

This bound can be tightened, as better estimates of the total number of repetitions in a partition become known.

Thus we have a metric that conservatively estimates the descriptive cost of any possible symbol in a string. A measure of the compression ratio for a particular symbol is simply the descriptive length of the string divided by the length of the string “covered” by this symbol. We define the compression ratio of a symbol (SCR) to be:

$$\tilde{\lambda}_i = \frac{d_i}{L_i} = \frac{r_i \left[ \log_2\left(\frac{L}{2}\right) - \log_2(r_i) \right] + l_i}{l_i r_i}$$

Thus we have a metric to describe the effectiveness for compression of a particular candidate symbol in a possible partition of a string that can be used for comparison in forming a partition. The overall compression ratio is bound by the following:

$$\frac{D_p}{L} \leq \frac{\sum_i d_i}{\sum_i L_i} = \frac{\sum_i L_i \tilde{\lambda}_i}{\sum_i L_i}$$

Examining SCR above it is clear that good symbol compression ratio arises in general when symbols are long and repeated often. But clearly, selection of some symbols as part of the partition is preferred to others. Figures 4 and 5 show how symbol compression ratio varies with the length of symbols and number of repetitions for a 1024 bit string. In both figures the discontinuities reflect when  $l_i \cdot r_i$  exceeds  $L$  and SCR is undefined.

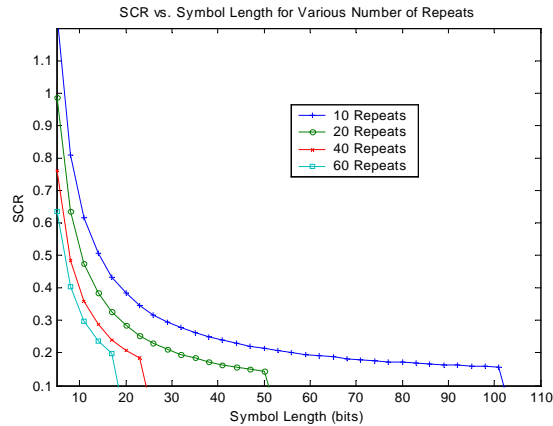
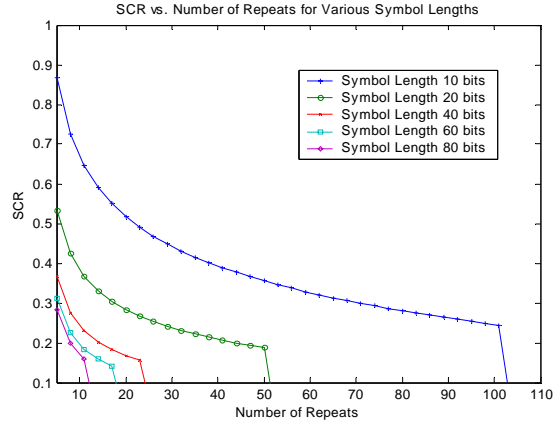


Figure 4: SCR vs. Symbol Length for 1024 bit String



**Figure 5: SCR vs. Repeats for 1024 bit String**

## 5. Optimal Symbol Compression Ratio (OSCR) Algorithm

The Optimal Symbol Compression Ratio (OSCR) algorithm forms a partition of string  $S$  into symbols that have the best symbol compression ratio among possible symbols contained in  $S$ . The algorithm is as follows:

### OSCR Algorithm

1. Form a binary tree of all non-overlapping sub-strings contained in  $S$  that occur  $\geq 2$  times and note the frequency of occurrence.
2. Calculate the SCR for all nodes (sub-strings). Select the sub-string from this set with the smallest SCR and add it to the model  $M$ .
3. Replace all occurrences of the newly added symbol with a unique character to delineate this symbol. Repeat steps 1 and 2 with the remaining binary string elements until no binary elements remain.
4. When a full partition has been constructed, use Huffman coding or another coding strategy to encode the distribution,  $p$ , of symbols.

The following comments can be made regarding this algorithm:

1. This algorithm progressively adds symbols that do the most compression “work” among all the candidates to the code space. Replacement of these symbols left-most-first will alter the frequency of remaining symbols.
2. SCR is at first based on the crude estimate for  $R$  discussed in Section 4 and Figure 3. Justification of this estimate and possible iteration of the algorithm can be achieved by performing the algorithm again upon completion with the computed value for  $R$



A1AA00A01A01A1AA1A1A

Iterating the algorithm shows that the second symbol candidate, 01 which has the smallest compression ratio, does not promote compression among the nodes in Figure 6 (SCR > 1). Thus the remaining symbols simply substitute for 1 and 0:

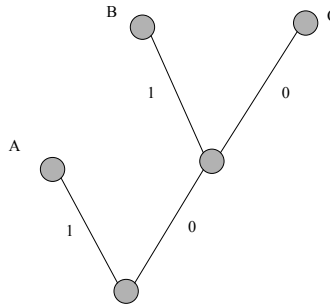
ABAACCACBACBABAABABA

This provides the following distribution of symbols:

**Table 2: Symbol Distribution**

Symbol	Probability
A	.5
B	.3
C	.2

The entropy of this symbol distribution is 1.48 bits per symbol. This can be approximated by the Huffman tree shown in Figure 7, which achieves an expected encoded length of 1.5 bits per symbol.



**Figure 7: Huffman Tree for Symbol Partition**

Encoded with this binary tree the original string is mapped to:

$$S' = 101110000100011000110111011011$$

Thus the encoded message has been reduced to 30 bits from the original 40 bits. The descriptive cost of the codebook is greater than the sum of the lengths of symbols, which is equal to 5 bits. Depending on the strategy for delineating the separation between code words and defining the prefix free encoding of the codebook, descriptive cost could increase by as much as  $\lceil \log_2(I) \rceil$  bits.

The previous example illustrates the concept of the OSRC algorithm. As is the case with Lempel-Ziv and other compression algorithms, greater compression is realized on strings of longer length. In addition to compression, the algorithm provides the following benefits:

- 1) The model in Figure 7 can be used to produce a typical set of strings to which S belongs.

- 2) The symbol alphabet size of 3 symbols is an inherent parameter associated with this string that can be used to compare it with other strings. The symbol size measurable parameter related to complexity that reflects the number of variables address by the string.

## 6. Applications and Implications

One implication of this algorithm is that descriptive complexity can be used recursively to improve the compression algorithm. The potential of reducing the descriptive length of the codebook by compressing individual code words or the codebook as a whole will be pursued in future work.

The model ascribed to the data by this algorithm is a binary tree constructed using a Huffman code that can be used to generate a typical set of data to which the string belongs. See [5] for an exposition of the relationship between Kolmogorov complexity and typical sets. The Huffman tree is an attractive model because one progresses through the tree creating sub strings with random coin flips at each node in the tree until a codeword is reached. This model relates directly to the Solomonoff notion of  $K(x)$  as the most probable (shortest) random program in a distribution of programs that could produce a given string on a given Turing machine [2]. The end nodes of the binary tree represent potential halt states for a very simple machine.

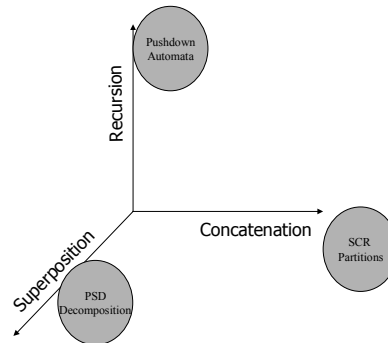
This algorithm provides an ideal mechanism for studying the tradeoff among complexity, entropy, compression ratio, and processing and memory resources and provides the capability to control the tradeoff among compression ratio, processing time, and memory usage. Clearly, compression of large strings results in a large model space, and thus large amount of processing and memory resources. By pruning the binary tree at a given depth,  $2^k - 2^d$  leafs can be saved given  $k$  total leaves and pruning at depth  $d$ . This will remove the possibility of finding symbols of length  $2^d$  or longer resulting in less compression. Furthermore, growing the tree only on certain high frequency branches can serve to limit resources while still identifying the most likely symbols with high compression ratio.

This algorithm can be used to generate the corresponding estimated minimal Turing Machine by mapping symbol types to states and symbol contents to output. The algorithm can also be used as to generate strings of a given complexity. The compression ratio or complexity estimator in this case, can be set to the desired value. Working backwards in the development of the algorithm, various values can be constrained, or left as open variables in order to generate strings of a given complexity. For example, by specifying the length ( $L$ ), complexity ( $D/L$ ), and total repetitions  $R$ , one of many possible Turing Machines of complexity  $D/L$  can be generating with the corresponding properties of complexity.

Clearly other models may provide smaller descriptive lengths. But this technique may prove to be a useful model to include in a set of models, especially where a string is composed of the concatenation of phrases. One can consider the MML determination of Kolmogorov complexity a 3 (or higher) dimensional problem as shown in Figure 7. The OSCR algorithm may provide a metric for string compressibility as a concatenation of

other sub strings. Other models may be more conducive to discerning recursive or superposition affects (such as power spectral density based metrics as discussed in [3]).

### Dimensions of Models for MML



**Figure 8: MML Vector Space**

Key areas for applications for this work include bioinformatics and biotechnology (see [8] for one example of how complexity theory intersects with the field of biotechnology). Compression algorithms such as Lempel Ziv are used to analyze DNA sequences - thus improvement in this area will be a direct benefit for DNA research. Additionally, modeling of genetic disorders and immune system response is strongly linked with complexity theory and will benefit from better estimators of Kolmogorov complexity. Our future work will explore this and other natural relationships between biotechnology and algorithmic information theory.

## 7. Conclusions

Symbol compression ratio is a useful method for determining optimally compressible partitions of strings. The algorithm provides a computable parameter related to Kolmogorov Complexity that enables discussion regarding the size of the likely alphabet discussed in a string. This provides a stepping-stone in seeking the optimal (smallest) description for the string. Further research is necessary (and is anticipated to be included in the final version of this paper) to benchmark this algorithm against well-known compression algorithms such as Lempel-Ziv, as well as other Kolmogorov complexity estimation and MML techniques.

## 8. References

- [1] Cover, T.M. and Thomas, J.A. *Elements of Information Theory*. Wiley, NY, 1991.
- [2] Li, M. and Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications*, Springer, NY 1997
- [3] Evans, S., Bush, S.F., and Hershey, J., "Information Assurance through Kolmogorov Complexity," DARPA Information Survivability Conference & Exposition II, 2001, Proceedings Vol. 2, pp. 322-331.

- [4] Wallace C.S. and Dowe, D.L., “Minimum Message Length and Kolmogorov Complexity,” *The Computer Journal*, Vol. 42, No. 4. 1999.
- [5] Gacs, P., Tromp, J.T., and Vitanyi, P. “Algorithmic Statistics,” *IEEE Transactions on Information Theory*, Vol. 47, No. 6, September 2001, pp. 2443-2463.
- [6] Kosaraju, S.R. and Manzini, G. “Compression of Low Entropy Strings with Lempel-Ziv Algorithms” *SIAM J. COMPUT.* Vol. 29, No. 3. pp. 893-911.
- [7] Kulkarni, A.B. and Bush, S.F., *Active Network Management and Kolmogorov Complexity OpenArch 2001*, Anchorage Alaska, April 27-28, 2001.
- [8] Ming Li, Jonathan H. Badger, Chen Xin, Sam Kwong, Paul Kearney, Haoyong Zhang, “An Information Based Sequence Distance and Its Application to Whole Mitochondrial Genome Phylogeny,” *Bioinformatics*, Vol. 17, No. 2, pp. 149-154, 2001.

### **Patent Dockets**

- RD29828      Self-Healing Faults Via Reversible Code
- RD29534      Method for Receptor Design in Self-Healing Complexity-based Fault Mitigation System
- RD29533      Self-Healing Systems via Complexity-based Algorithmic Active Packet Fusion Receptors
- RD29824      BioInformatic Algorithmic In-Route Model Composition using Active Network Support

### **Acknowledgments**

The work discussed in this paper was funded in part by DARPA, under the auspices of the Fault Tolerant Networks program. Our thanks go to Doug Maughan, the Active Networks program manager and Scott Shyne, Air Force Rome Labs for their generous support.

**S.C. Evans**  
**S.F. Bush**

**Symbol Compression Ratio for String Compression and  
Estimation of Kolmogorov Complexity**

**2001CRD159**  
**November 2001**