Entropy gradient: a technique for estimating the entropy of finite time series

Adel S. El-Atawy and Ahmed A. Belal
Dept. of Computer Science and Automatic Control, Faculty of Eng, Alexandria University, Alexandria, Egypt

Several techniques are used to estimate the entropy of a finite sequence of symbols taken from a finite alphabet. Here we will present a new technique based on using varying levels of approximations for the entropy. Fitting the different approximations into one of a set of functions will result in an estimate to the absolute entropy. New applications are presented for the entropy gradient technique; estimating the real memory span of finite memory machines as a special case of finite state machines and investigating the degree of compositeness of numbers.

توجد طرق متعددة لتقدير الأنتروبية لسلسلة محدودة الطول من الرموز المأخوذة من أبجدية محدودة. سيتم عرض تقنية جديدة تعتمد على استخدام درجات متعددة من التقريبات للأنتروبية. و سيؤدى وضع تلك التقريبات المختلفة فى صورة واحدة من مجموعة من الدوال إلى تقدير الأنتروبية المطلقة. سيعرض تطبيقان جديدان لطريقة تدرج الأنتروبية؛ تقدير عمق الذاكرة الحقيقي للآلات محدودة الذاكرة كحالة خاصة للآلات محدودة الحالات، و دراسة قدر التركيب للأرقام.

Keywords: Entropy estimation, Finite memory machines, Finite state machines, Number properties, Compositeness

1. Introduction and preliminaries

The Information Theoretic version of entropy was first proposed in its modern form by C.E. Shannon [1] in a statistical form and by Kolmogorov in an algorithmic theoretic view [2].

The Shannon Entropy of a data sequence is a highly used figure to describe the complexity, compressibility [3], amount of information, weight of noise component, effectiveness of random data generators, and many other properties of the analysed data. Various applications exist for the estimation of entropy in quite distant fields as Biology [4], and Medicine [5,6]. A good collection of various applications is presented in [7].

The amount of information per symbol extracted from a data source (i.e., entropy) was prior to Shannon considered to be $\log_2(|S|)$ bits of information where S is the alphabet from which the data source selects its output.

Shannon introduced the effect of statistical properties of the source on its entropy level. Given the probabilities of various symbols of the alphabet, the general form of this relation is:

$$H(S) = -\sum_{S} p.\log(p) \,. \qquad (1)$$

It extends to extract the information of a data source, realizable as a state machine, by averaging the entropy of the data source over all its states;

$$H(S) = -\sum_{States} P(State) \sum_{S|State} p.\log(p).$$

Or the detailed notation as will be used in the calculations;

$$H_m^e(S) = -\sum P(s_1 s_2 \ldots s_{e+m}) \log(s_1 s_2 \ldots s_e \mid s_1 s_2 \ldots s_m)\,,$$

where e is the source extension level (i.e., symbol block size), and m is the memory span (i.e., order of the machine's Markov model) [8].

As we go further in analysis; adding up states, estimating probabilities of symbols more reliably, and observing the conditional probabilities using longer correlation into the history of the data sequence, we achieve better approximations of the absolute entropy of the data source.

By studying the way our approximations get better, we can get a better inside view of the data source under consideration, and extrapolate to find out the real entropy of the source.

To estimate the probabilities needed for the entropy calculations, we can make use of the following general form; $P(A)=\frac{n_A+\beta}{N+\beta d}$, where n_A is the frequency of event A among total samples, and d is the cardinality of the alphabet set (i.e., $|S|$). β is a constant to be chosen according to each specific case. β=0 is the normal maximum likelihood estimation form, β=1 is the one proposed by Laplace, but others used values like $\beta=1/d$ [14]. We will show that negative values of β have given amazing results in some applications.

We will first discuss the technique broadly, then show how to realize it explaining some of the implementation problems. In section four, we will show how this technique was verified using some standard data sources, and give comparisons to other techniques. In the last section we will focus on a couple of new applications that make use of the technique, and entropy estimation in general.

2. Technique overview

We can divide the technique into three phases: data sequence processing (data collection phase), probability estimation and entropy calculation (calculation phase), and finally curve fitting and result extraction (estimation phase).

2.1. Data collection phase

Assuming the data source is providing its output on a symbol per symbol basis (online), namely; $s_1s_2s_3 \ldots s_N$ from a finite alphabet S, $|S|$ = d. N is finite also.

In order to be able to calculate all entropies based on a range of extensions and memory depths, we need to save all the frequencies (or what is enough to deduce it) of such blocks of symbols [10]. We will be using a d-ary suffix tree (or a prefix tree, they are both equivalently useful to our work as is shown later) where d is the cardinality of our source alphabet.

The height of the tree is strictly bounded by a certain Depth. This *Depth* is assumed by the user of the technique depending on his application, suspected complexity of the data source, and/or length of available data sequence. However, *Depth* should generally be less than $log_d(N)$ for the frequencies of symbols to be a reliable representation of the data source's real probabilities.

The tree (the suffix type) holds the following data: each node has the symbol and the number of times (its frequency) being the predecessor of its parent symbols (the root node, carries the *NULL* character, and its frequency value is the total symbol count N). In the prefix tree case, each node stores the count showing how many times it was its parents' successor.

The process for building up the suffix tree proceeds as follows: For each new symbol, increment the root node, and increment (or create with count 1) all its successive children as the current symbol and previous symbols dictate, as shown in fig. 1.

The same procedure applies in the case of a prefix tree moving down the tree according to the future symbols, not past ones. Of course the future *Depth* symbols should be known, this can be achieved by delaying the processing of any symbol until *Depth* symbols are observed. See fig. 2.

The only cases that we cannot handle are the start-up state when using suffix trees, and the finalizing state when using prefix trees. For these cases, the logical statement that the sum of counts of all nodes should be equal to the count of their common parent will be unsatisfied. This problem can be attributed to the information hidden in the initial (or final)

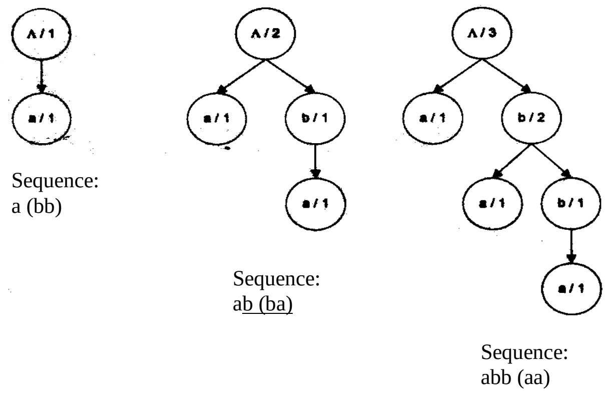

Fig. 1. Suffix tree after processing the sequence "abb". Showing the first steps and the final result.

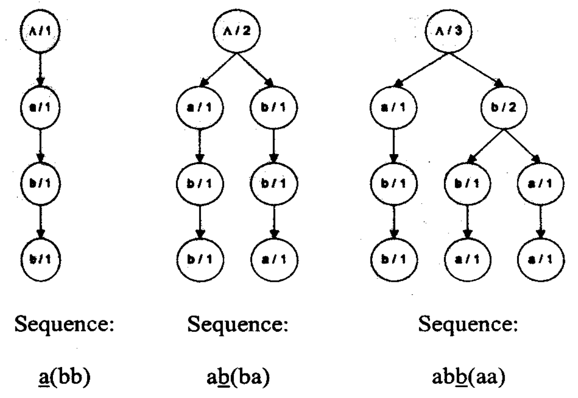

Fig. 2. Prefix tree after processing the sequence "abb(aa)", "aa" is still to come. Showing steps after each symbol.

state of the data source itself. This exactly-one error in the frequency should be normally discarded when the size of data processed increases, but for very short sequences it can make a difference in the final entropy value. This will be handled in the calculation phase.

2.2. Calculation phase

The constructed tree can be used to calculate the frequency of all symbols and states as follows (using examples from fig. 3).

Using prefix trees: The value of any node corresponds to the frequency of the symbols subsequence from the root. For example; Freq(ab) = 3, by moving from root→*a*→*b*. Also, the conditional frequencies can be calculated as well, by moving according to state symbols, then specific symbol sequence. For example; Freq(ab|*a*) = 1, root→*a*→*a*→*b*. Hence, we can calculate all the various probabilities needed for the entropy calculation using a single path from root to leaf. For example; taking the path root→*a*→*a*→*b* we can get all the following values: P(*a*)=5/9, P(*aa*)=2/9, P(*a*|*a*)=2/5, P(*aab*)=1/9, P(*ab*|*a*)=1/5, P(*b*|*aa*)=1/2.

Using suffix trees: It will be a little bit harder than the prefix tree to use, but with the same complexity. The same holds for all unconditional frequencies and probabilities But conditional forms will need to extract the total count of the state under consideration. This can be done by traversing the tree, keeping track of multi-pointers with various depths. For example; Freq(*b*|*aa*) = Freq(*ba*|*a*) =Freq(*aab*) = 1., P(*b*|*aa*) = Freq(*aab*)/Freq(*aa*) = 1/2.

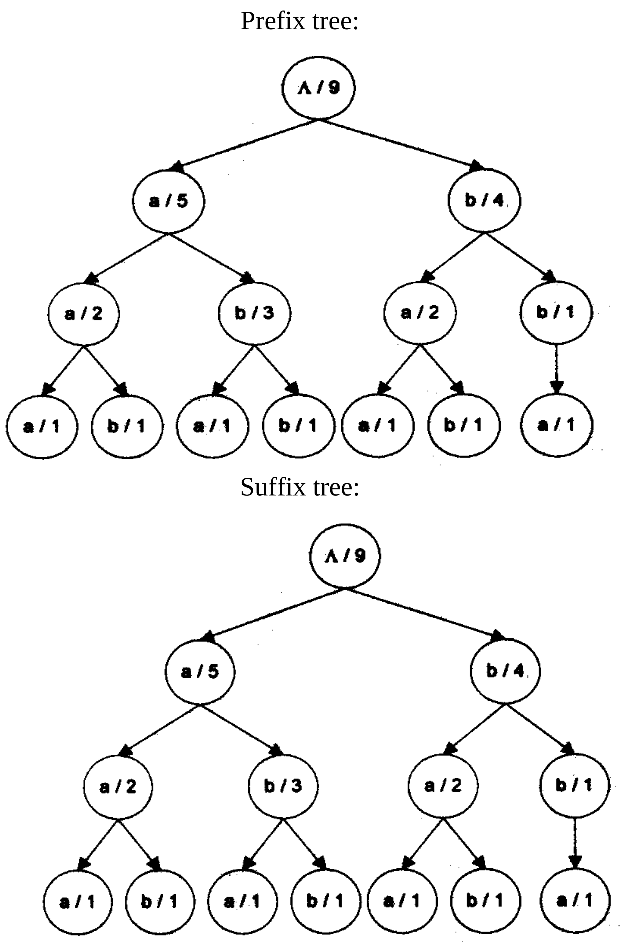

Fig. 3. Prefix and Suffix trees for the sequence "abbaaabab".

An algorithm can be designed easily for traversing both types of trees, to accumulate terms needed for the calculation of all the entropies with *e*+*m* ≤ *Depth*, starting with zero entropies, and adding up entropy terms (i.e., Plog(P)) as we traverse down the trees. It can be shown that both trees will be traversed with the same complexity, since for every node the amount of work done is similar, and the number of nodes in the whole tree will be essentially identical in both cases.

Next, we show how to handle the few problems that arise in the case of very short sequences: Biased Estimation, and Initial/ Final state effect.

Biased Estimation: It was shown in various related work [11, 12] that the entropy estimated from the naïve form of estimating the probabilities using frequencies (MLE) results in a biased value;

$$\langle H(S)\rangle = H(S) - \frac{M-1}{2N} + \frac{1}{12N^2} \times (1 - \sum_{p_i>0} \frac{1}{p_i}) + O(\frac{1}{N^3}).$$

where M is the different number of symbols (or blocks of symbols) monitored. This bias term can be also decreased by using the Laplace form of estimator, or by adjusting the β parameter accordingly, for example, it was shown [9] that $\beta = 1/d$ is best suited for data with long correlations (e.g., English text). Also, we can treat the bias by adding up the first term to the calculated entropy (M can be obtained for each different $e+m$ by counting the number of nodes in each level in the tree while traversing the tree to get the entropy terms themselves).

Initial/Final state effect: The problem appears for the last (or first, in the case of suffix trees) symbols analysed, as they won't have future (past) symbols. This can be seen in fig. 3 for nodes ($\Lambda \rightarrow b$, $\Lambda \rightarrow a \rightarrow b$) (or in the suffix tree: $\Lambda \rightarrow a$, $\Lambda \rightarrow b \rightarrow a$) as the node count does not equal the sum of its children nodes. This unity-difference affects the final value of entropy calculated, but disappears very quickly by processing more symbols. To compensate for its effect, we should add this unity discrepancy into the child nodes, and there are various ways to distribute it among child nodes:

Equal Share: safest method.

In Ratio: optimistic, we assume the initial state takes the same trend as the average seen till now.

Worst case: distribute it in a way to increase the estimated entropy (by adding it to the least count, and on ties, distribute evenly).

These methods assume that there might be no different symbols other than those observed, but consider the case in fig. 3 $\Lambda \rightarrow b \rightarrow b$) (same case in the suffix tree) where we observed only a's after the 2 b's. In such cases it will be useful to add up fake nodes, with zero count for missing symbols, then use the distribution strategies mentioned above.

2.3. Estimation phase

For some purposes the process will stop here, with a table of entropy values for the ranges e=1 to *Depth*, and m=0 to *Depth-e* (i.e., an upper triangular matrix). For other - mostly automated - processing cases, we will need to fit these values into a surface that gives us an overview for the behaviour of the source. A curve fitting strategy might look direct and straightforward, but it will not be very successful if not done carefully. First, we should choose a function form of some logical meaning related to the samples (i.e., entropy values) we have, see table 1. We can enumerate a few problems with the direct least square error technique

The fitting cannot be symmetric with respect to negative and positive errors. If the fitted curve floats over the samples, that will be safer than getting below them. Being conservative in entropy estimation is most probably safer than giving underestimations.

The bias - not fully removable - increases with increasing the $e+m$ value, this dictates that the fitting technique have to be less sensitive in some regions (i.e., high $e+m$) than others.

Some constraints may be placed over the parameters of the functions to be used.

Formalizing our problem into a generic optimization problem will go as follows:

$$\min: \quad \text{error} = \sum_{e=1}^{D} \sum_{m=0}^{D-e} \varepsilon_{e,m,\upsilon}^2 ,$$

where;

$$\varepsilon_{e,m,\upsilon} = \begin{cases} \lambda(H^*(e,m)\Big|_{\upsilon} - H(e,m)) & , H^*(e,m)\Big|_{\upsilon} < H(e,m) \\ (H^*(e,m)\Big|_{\upsilon} - H(e,m)) & , H^*(e,m)\Big|_{\upsilon} > H(e,m), \end{cases}$$

and $\lambda > 1$ (υ is the parameter vector of the EGF, Entropy Gradient Function, to free the objective function from their exact representation. The components of this vector are the target for optimization). The effect of the added λ parameter is to make the error in the negative direction more important to avoid than positive error. Other alternative formulations of the problem are:

$$\varepsilon_{e,m,v} = Exp(\lambda(H^*(e,m)\Big|_{v} - H(e,m))) + Exp(H(e,m) - H^*(e,m)\Big|_{v}),$$

or

$$\varepsilon_{e,\,m,\,v} = \begin{cases} (H^*(e,m)\big|_v - H(e,m))^{\alpha} & ,H^*(e,m)\big|_v < H(e,m) \\ (H^*(e,m)\big|_v - H(e,m))^{\beta} & ,H^*(e,m)\big|_v > H(e,m), \end{cases}$$

where $\alpha > \beta$.

Many optimization techniques [13] can be used to solve this problem, as it is a well behaved objective function whatever the values of the errors.

3. Technique verification

In order to verify the correctness of data collection and entropy calculation schemes (as well as the suitability of the proposed EGF's), a few test sequences were used. The used sequences have statistical forms that allow for an exact calculation of their entropy.

3.1. Test 1: Exponential distributed data

The alphabet consists of n symbols with the following probability distribution:

$$P(x=i) = \begin{cases} 2^{-i} & ,0 < i < n \\ 2^{-(n-1)} & ,i = n\,. \end{cases}$$

The entropy of this source;

$$\sum_{i=1}^{n} p.\log(p^{-1}) = \sum_{i=1}^{n-1} 2^{-i}.\log(2^{i}) + 2^{-(n-1)}.\log(2^{n-1})$$

$$= \sum_{i=1}^{n-1} i.2^{-i} + 2^{-(n-1)}.(n-1) = 2 - 2^{-(n-2)}.$$

Comparing the results obtained by using the EG method with a context-based technique like Lempel-Ziv [14] and a statistical technique like Huffman (i.e., first step of entropy approximation) we obtained the following results shown in fig. 4.

Fig. 5, shows how close the results of Huffman (a memory-less technique) and the EG technique were. EG is slightly lower at some points as it is not bounded to encode such data as the Huffman technique does.

EG gives better results than Lempel-Ziv that tries to get contextual relations between successive symbols, while the data - by definition - is memory-less.

3.2. Test 2: Single memory source (locality referenced states)

The output of the data source is the label of the current state of a specific probabilistic finite state machine.

Each state is labelled with two parameters; a group number and an inter-group number. At each state transition, the group number changes to next value with probability ε, and stays still with probability $1\text{-}\varepsilon$ (In a cyclic fashion over 1 to *M*). Next inter-group number is selected uniformly from 1 to *N*.

The entropy of this source can be calculated independently from the two label components;

$H(S)= H(grp\#) + H(inter\text{-}grp\#)$
$\quad = H(\varepsilon) + \log(N)$,

where;

$H(\varepsilon) = -\ \varepsilon.\log(\varepsilon) - (1\text{-}\varepsilon).\log(1\text{-}\varepsilon)$

The next graph fig. 6, shows the result of applying EG, LZ, and Huffman (compared to the exact value) on the output sequence of the defined data source. As was in the case of Test 1, LZ gives a steady higher estimate than EG and Exact value. The Depth needed can be as low as 2 levels, although higher values can be used without affecting the final result (Depth is still subject to the limits mentioned before).

This source is an example of a statistical source with finite context dependency length (memory span = 1 in this case).

3.3. Test 3: English text

Using the EG technique (with EGF-3 and Depth = 5), and with application to English text (various novels, textbooks,... and their combinations, sum up to about 6x10^7 chars) gave results that are within 4% from that

Table 1
Some proposed EGFs and their corresponding constraints

Examples of possible EGFs:	Constraints
$H^*(e,m) = h + \frac{r}{e+m}$	h, r ≥ 0 h ≤ log(d)
$H^*(e,m) = h + \frac{1}{ae+bm}$	a, b ≥ 0
$H^*(e,m) = h + (ae+bm)^{-c}$	c ≥ 0

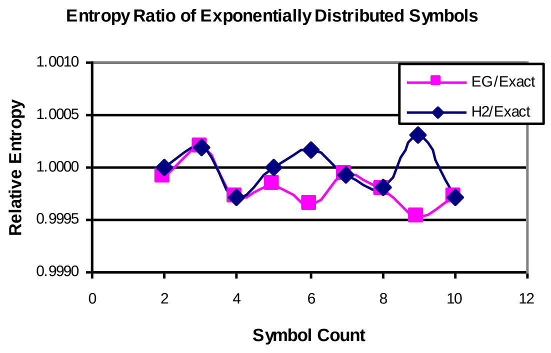

Fig. 4. Entropy results for exponential distributed data, using EG, Huffman, LZ, compared to the exact value.

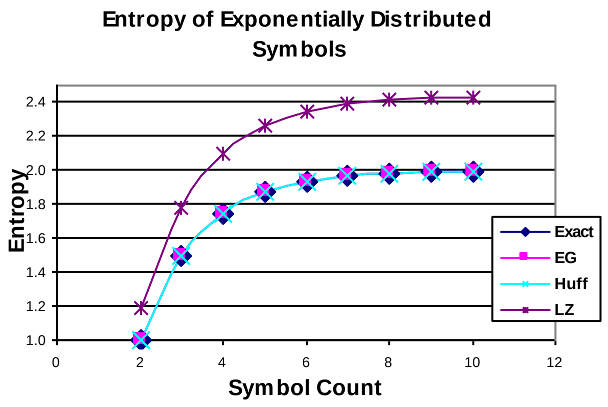

Fig. 5. Zoomed fig. 4, comparing Huffman and EG technique results.

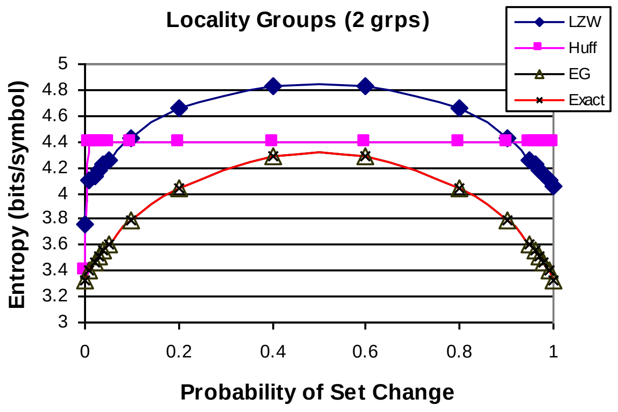

Fig 6. Entropy curves for Locality based state machines. Comparing EG to Huffman, LZ and the exact values. (Huffman is seen as a steady line).

obtained by previous researchers [9] (EG achieved 1.796 bits/char and Schurmann's extrapolated estimate was 1.860 bits/char, and 1.781 bits/char for plain 27 characters text).

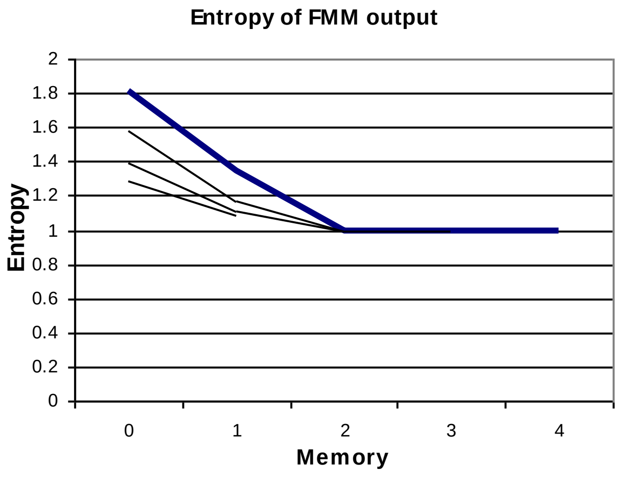

Fig. 7. Entropy gradient over memory, for a specific finite memory machine with memory span equals to 2.

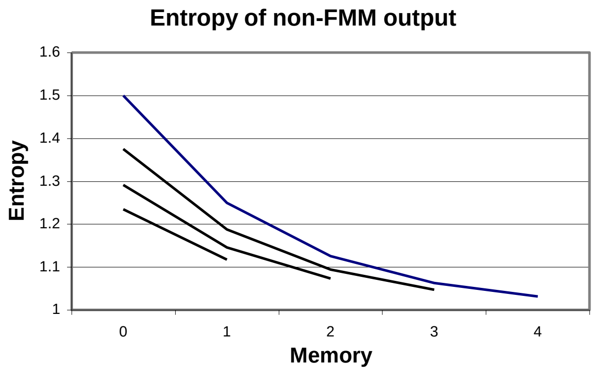

Fig. 8. Entropy gradient of an example FSM, over different memories. The multiple lines represent the entropy for different extensions.

This test ensures the applicability of our technique to data with high contextual content.

4. Applications

Many applications that use entropy estimation exist. Here, two new applications for which the EG technique may be specifically useful are proposed.

4.1. Finite memory machines

A finite memory machine is a state machine that given the previous μ inputs/outputs pairs you can identify the current state [15].

Given a finite state machine, it is generally not trivial to decide whether it is a finite memory machine or not.

The problem is with finding all the possible input sequences and guessing the next state (by studying previous sub-sequences of an assumed length). If the guess was successful for all possible sequences then their length is the machine memory span. Carrying this experiment is practically hard, and another method is proposed in this paper based on entropy analysis.

The identification process goes as follows; Generate a random sequence X (a pseudo random sequence will do), and apply it to the machine and record its output sequence Y. Calculate the entropy approximations using different memory depths (i.e., $H(1,m)$ for m=0, 1,... Depth-1) for both sequences; X and $Z = \{(x_i,y_i)\}$ (i.e., each symbol is an ordered pair of the input and its corresponding output).

As X is taken to be a random sequence; its entropy will stay steady at 1 bit/symbol (in case of a binary input/output machine). The entropy of Z will start high (up to 2 bits/ symbol for, as assumed, binary input/output) and decays with increasing the memory of the estimation till it reaches the entropy level of X. The reason is that the information hidden inside the machine (the knowledge of the current state) is decreased till it vanishes completely when we use μ previous states in the estimation.

So the test concludes that the given FSM is a FMM with memory span equals to m, where $H(1,m)(X) = H(1,m)(Z)$. This is shown in fig. 7.

A thing to note is the effect of using different values of β in the estimation of probabilities. Negative values of β result in a very useful result namely, the approximation values achieve a minimum at the required memory level (at $m=\mu$), instead of stabilizing at the entropy level of X. This can reduce the effort to search for the appropriate m.

One last point, if the machine is not a FMM in the first place; then we can know this from the entropy gradient of the Z sequence as it will never home into that of the input sequence, but keeps on approaching it in a decreasing rate as shown in fig. 8.

4.2. Number properties

Here we try to study the properties of numbers using the average of the entropy of their representation in more than one radix.

The reason behind the idea is that if a number is composite, then we expect that we might see the pattern of its factors appearing in the representation of the number itself. For example; 1111,0101,0101 (3925) contain the pattern 101 (5) more than once, and 3925 is divisible by 5. So, we can say that there is some probability that highly composite numbers will contain some repeated patterns. Repeated patterns lead eventually to low entropy.

We proposed some properties that can be quantitatively measured for a given number to study and to be compared with the entropy of such a number, namely; factor count, divisor count, distinct factors count, factor average, minimum factor, maximum factor.

In the next graph, fig. 9, the entropy of a set of numbers is compared to the number of factors of such numbers, showing the trend of the overall set of points.

The correlation was measured between each measure and the entropy of the number using binary representation, and by using the average entropy over a number of representations. More than one range of numbers was used, and the findings were consistent and conforming with logic. Factors/Divisor Count when increased means smaller patterns can be pointed out more easily, resulting in decreased entropy. Also, Average/min/max factor value increases when factors become

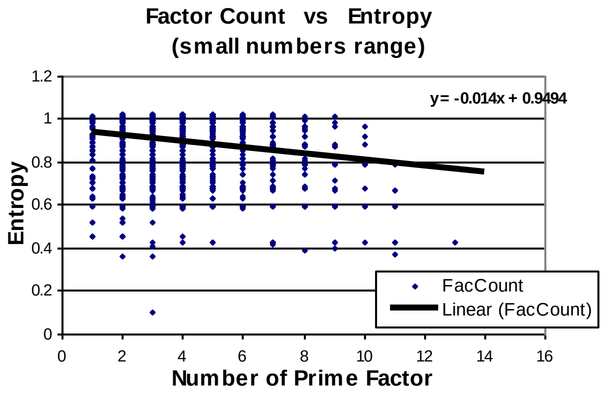

Fig. 9. Factor count vs. entropy.

Table 2
Correlations between some of the number properties selected and entropy of number representations (10^4 numbers around the 10^6 range)

Measure	Correlation
Factor count	-0.087
Divisor count	-0.075
Distinct factors count	-0.024
Factor average	0.108
Minimum factor	0.080
Maximum factor	0.140

longer and harder to find in the final representation. See the next table for a sample of correlation values for one of the tested ranges (10^4 numbers around the 10^6 range).

5. Conclusions

We have here presented a better form to keep statistical information about a data source, and that allows estimating probabilities with different block sizes of symbols and different conditional levels. Thus, the same data is used to estimate entropies with all estimating powers instead of saving costly exhaustive listings for combinations of symbols and states as other techniques might do.

Problems facing probability estimation were mentioned and solutions were proposed; bias, zero probability problem, initial/final data source state problem.

Basically a prefix or - equivalently - a suffix tree is used to carry multilevel frequency information. And it was shown that there is no difference in the amount of work needed to extract entropy values from both data structures.

Proposed curves were shown, with increasing degrees of freedom, to be used to fit to the entropy values with some modifications to the fitting techniques normally used.

Two new applications were investigated; finite memory machine identification, and studying the degree of compositeness of numbers. The first used the entropy gradient directly, and the other went to the final state of the estimation to use the estimated entropy itself.

References

[1] C.E. Shannon, “Mathematical Theory of Communication”, Bell System Tech. Journal, Vol. 27, pp. 379-423, 623-656 July, October (1948).

[2] M.L. and P. Vitanyi, An Introduction to Kolmogorov Complexity and Its Applications, Springer (1997).

[3] T. Bell, I.H. Witten, and J.G. Cleary, “Modelling for Text Compression”, ACM Computing Surveys, Vol. 21 (4) pp. 557-591 (1989).

[4] D. Loewenstern and P. Yianilos, “Significantly Lower Entropy Estimates for Natural DNA Sequences”, Data Compression Conf, p. 151, March (1997).

[5] H. Bo; Y. Fusheng, T. Qingyu, and Tin-C. Cheung, “Approximate Entropy and Its Preliminary Application in the Field of EEG and Cognition”, 20th Annual Conf of the IEEE Engineering in Medicine and Biology Society, Vol. 20 (4) (1998).

[6] L. Quanzheng and G. Xiaorong “Subsection Approximate Entropy and Its Application In Sleeping Staging”, Proceeding of 1st Joint BMES/EMBS Conference Serving Humanity, Advancing Technology, Atlanta, pp. 13-16 (1999).

[7] S. Verdu, “Fifty Years of Shannon Theory”, IEEE Trans. Info. Theory, Vol. 44 (6), pp. 2057-2078 (1998).

[8] N. Abramson, Information Theory and Coding, McGraw-Hill, New York (1963).

[9] T. Schurmann, and P. Grassberger, ‘Entropy Estimation of Symbol Sequences”, American Institute of Physics: CHAOS, Vol. 6 (3), pp. 414-427 (1996).

[10] T. Poschel, W. Ebeling W. and H. Rose, “Guessing Probability Distribution From Small Samples”, Journal of Stat. Phys., Vol. 80, pp. 1443-1452 (1995).

[11] G.P. Bashrain, “On a Statistical Estimate for the Entropy of a Sequence of Independent Random Variables”, Theory Probability Appl., Vol. 4 (3), pp. 333-336 (1959).

[12] R. Moddemeijer, “The Distribution of Entropy Estimators Based on Maximum

Mean Log-Likelihood", 21st Sym Inf Theory, pp. 231-238 (2000).

[13] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty, Non-Linear Programming, second edition, John Wiley & Sons (1993).

[14] Lempel and J. Ziv, "A Universal Algorithm for Sequential Data Compression", IEEE Trans. Inf. Theory, Vol. 23 (3) pp. 337-343 (1977).

[15] F.J. Hil and G.P. Peterson, Introduction to Switching Theory and Design, third edition, John Wiley & Sons, sec 10.4, (1981).

Received May 4, 2003
Accepted November 30, 2003