TREIZIÈME COLLOQUE GRETSI - JUAN-LES-PINS DU 16 AU 20 SEPTEMBRE 1991

UNIVERSAL DATA COMPRESSION WITH GENERALIZED ALGORITHM CONTEXT

G. Furlan - J. Rissanen

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120

Résumé

L'algorithme du Contexte pour la modélisation universelle de source, introduit par J. Rissanen [1] dans le cas binaire, est généralisé au cas non-binaire, ce qui permet son application pour la modélisation de différents processus aléatoires comme ceux rencontrés en compression d'image, avec ou sans perte d'information, dans les systèmes chaotiques, et, plus généralement, partout où une prédiction est nécessaire. Cette généralisation comprend deux améliorations majeures: le contrôle de la taille du modèle et une modification des règles de sélection du contexte afin d'accroître les performances et la vitesse d'exécution, qui sont combinées lors de cette implémentation. En simples termes, on peut décrire l'algorithme du Contexte comme essayant simultanément tous les modèles appropriés, disons de type Markovien, et ne retenant que le plus performant pour la prédiction de chaque symbole.

Abstract

A universal modeling algorithm Context, introduced in Rissanen [1] for binary strings, is generalized for nonbinary strings, which makes it applicable to modeling many types of random processes, such as encountered in image compression, both lossless and lossy, chaotic systems, and generally speaking whenever prediction is needed. This generalization includes two major improvements, the control of the size of the required tree and a modification of the original context selection rule to improve accuracy and speed, which in the current implementation are combined. In broad terms, one can view the algorithm as running simultaneously all the relevant models, say of Markov type, and, for the prediction of each symbol, choosing the most appropriate of them.

I. Introduction

The first universal data compression algorithms were capable of encoding strings, generated by independent information sources, with asymptotically optimum mean per symbol length without a priori given source probabilities. Clearly, such algorithms estimate either directly or indirectly the statistics with increasing accuracy while the string is being encoded. The same approach can be applied, at least in principle, to all stationary sources by means of gathering the statistics of longer and longer segments. However, in practice there is an obvious difficulty of exponentially growing number of items to be stored, and new ideas are needed to do the job in a practically meaningful manner. One of the most powerful universal algorithms published to date is due to Ziv and Lempel [4]. Their elegant algorithm achieves asymptotically optimum compression for strings generated by any stationary ergodic source, and it does the job in many cases in a quite practicable manner.

At the same, there is a major problem with an incremental block parsing algorithm such a Ziv-Lempel algorithm, and with all block models for that matter, namely, that they capture random dependencies between symbols that fall within one and the same block. Also, for the large class of strings, where symbols interact in two or higher dimensional neighborhoods, any technique producing one dimensional parsed segments is necessarily ineffective and often impossible to utilize in a practicable manner. In order to obtain more powerful models, the requirement that the collected segments only partition the string must be relaxed. In the currently described solution this is done so that each symbol's occurrence counts, or more general statistical properties, are collected conditional to other nearby symbols, defining the symbol's "context". As shown in [5], such conditioning allows for a more efficient way to take advantage of statistical regularities. Indeed, even the segments found by the incremental parsing algorithm define contexts for each of their constituent symbols, and such an interpretation offers a uniform and more fundamental explanation of the modeling efficiency of block models. In broad terms, the idea behind the algorithm Context is to gather statistics of each symbol in ever growing contexts as the string is being processed, while shifting through the contexts to find the optimal one. One can view the algorithm as running simultaneously all the relevant Markov models, and, for the prediction of each symbol of the string, choosing the most appropriate of them.

In Section II we introduce algorithm Context in its most advanced form, which combines a stochastic complexity,

[7], based rule for optimal state space selection with a control of the tree size. Section III gives some practical results when the Context algorithm is combined with an arithmetic coder, [5], [6]. Finally, Section IV includes a summary.

II. New implementation of algorithm Context.

Algorithm Context is an efficient statistics gathering and storing device. It collects in a tree all the numbers of times each symbol occurs in various "contexts" or, synonymously, "states". A "context" is a set of past symbols which are thought to have an influence on the current symbol occurrence. A simple example is a Markovian model, where a symbol's context is a certain number of immediately preceding symbols. Such a context tree is capable of representing all the contexts of a certain agreed type that actually occur in a string or an image, and it provides about the most powerful "universal" model we can think of. To understand how this is accomplished, one must understand the idea of "Stochastic Complexity" [7] and how it is used to determine the optimal context for each symbol. Stochastic Complexity generalizes Shannon's information of data in that it is defined as the negative logarithm of the greatest probability we can construct for the considered data string within a broad class of distributions, rather than just one given distribution as in Shannon's case. This amounts to adding to Shannon's information a term reflecting the complexity of the task required to estimate, one way or another, the required distribution. The principle of searching for a model class which permits the shortest total code length for the data; ie, the stochastic complexity, is called the Minimum Description Length (MDL) principle. With this principle one can select in the context tree for each new symbol that node as the optimal context which gives the smallest Stochastic Complexity for the past symbol occurrences at this node.

To encode a symbol at its optimal node (a state), a multiplication-free Arithmetic Coder, [6], may be used, which does the encoding with help of the statistics of the symbol provided by its optimal context. As well known, in an Arithmetic Coder each symbol's "codeword", as it were, is instantaneously redesigned, so that a near-optimal code length results whether or not the string is modeled as a stationary or a non stationary source. By contrast, redesigning a traditional Huffman code tree for each symbol with its changing statistics would be hopelessly complex and out of question. Even in its multiplication-free version the efficiency of an Arithmetic Coder is 97-99 percent or better. For the reader unfamiliar with arithmetic coding, we mention that an Arithmetic Coder constructs the code string as a cumulative probability of the string that precedes the symbol to be encoded in the lexical order of the strings. When calculating this cumulative probability, the code uses a certain approximation of the probability of the string defined by the model, the Context Algorithm in this case. This approximation satisfies all the properties of an information source, which makes the decoding possible.

Before giving the new implementation of Algorithm Context, we need a few notations. We consider a growing string $x(t) = x_1, \ldots, x_t$ of symbols taken from the alphabet $A = \{a_1, \ldots, a_d\}$. The computations are done adaptively for each symbol x_t in the string. Let $z_1 = x_{t-f(1)}$, $z_2 = x_{t-f(2)}$, ... denote a renumbering of the past symbols in the string defined by a permutation $1 \leq f(k) \leq t-1$, $k = 1, \ldots, t-1$. This reordering is based upon our belief in that z_1 has the greatest influence on the symbol x_t, the next greatest being z_2, and so on. For example, for a string of letters, we may take the function f as the identity function so that $z_1 = x_{t-1}, z_2 = x_{t-2}$ and so on. In the case where the symbols represent the pixels of an image, an appropriate choice is $x_{t-1} = x(i,j-1)$, $x_{t-2} = x(i-1,j)$, $x_{t-3} = x(i-1,j-1)$ and so on, where i and j represent the line and the column indices of $x_t = x(i,j)$, respectively.

In these notations the algorithm, which recursively collects in a tree essentially all the occurrence counts of the k+1-tuples of the values $x_{t-f(k)}, \ldots, x_{t-f(1)}, x_t$ that occur in the string, is as follows:

Algorithm Context

1. *Initialization:*
   Start with the tree consisting of a single root node, which has a counter for each of the symbols in the alphabet, all set to 0. Read the first symbol x_1, encode this symbol as described in step 3) and increment the count of this symbol by 1. Denote the resulting, still 1-node tree by $T(1)$.
2. *Choice of the coding node (optimal context):*

   Each node of the tree has a list of the d symbol counts together with a Relative Efficiency Counter, called REC, whose sign indicates its efficiency relative to its father node for encoding a symbol, provided that a father node exists; we take the root to have a negative REC. Recursively, let $T(t-1)$ be constructed after $t-1$ first symbols have been processed. Climb the tree, starting at the root according to the substring $z_1, z_2, \ldots$ into the past. Let $z(i) = z_1, z_2, \ldots, z_i$ be such a path in the tree. This path defines a node which is a possible context for the "next" symbol x_t. Define the unique encoding node as the first node $z(i)$ with a negative value for its REC which precedes a node $z(i+1)$ with a nonnegative REC, if one exists.
3. *Encoding:*
   Use the multiplication-free arithmetic coder to encode the symbol x_t. For that, feed the arithmetic coder with the distribution formed by the d symbol counts at the encoding node.
4. *Update of the tree:*
   Climb the tree, starting at the root, according to the substring $z_1, z_2, \ldots$. Then, for each node $z(j) = z_1, \ldots, z$ visited do the following:
   - Compute the ideal code length $l(x_t | z(j))$ of the symbol x_t at the considered node $z(j)$

$$l(x_t | z(j)) = -\log_2 P_{est}(x_t),$$

where $P_{est}(x_t)$ denotes the probability of symbol x_t as defined by the counts at the considered node using a suitable estimator. As an example, we can choose the classical Markovian estimator defined as:

$$P_{est}(x_t \mid z(j)) = \frac{n(x_t \mid z(j))}{n(z(j)) + 1} \quad if \quad n(x_t \mid z(j)) \neq 0;$$

and

$$P_{est}(x_t \mid z(j)) = \frac{1}{(n(z(j)) + 1)(d - q)}$$
$$if \quad n(x_t \mid z(j)) = 0;$$

where $n(x_t \mid z(j))$ denotes the count of symbol x_t at this node and $n(z(j))$ is the sum of all such symbol counts while q is the number of symbols which have not yet occured at node $z(j)$

- Except for the root, add to the REC at the node $z(j)$ the difference $I(x_t \mid z(j)) - I(x_t \mid z(j-1))$. If the new value of the REC is greater than a threshold RECMAX, then set REC to RECMAX. Similarly, if REC < RECMIN, then set REC to RECMIN.
- Increment by 1 the count corresponding to the symbol x_t.

5. *Growing the tree:*

   If the count of the symbol x_t at the last node visited after update is greater than 1, and the REC of the node is negative, then form a new node with a branch defined by the symbol extending the substring one step deeper into the past, namely, $z_m = x_{t-f(m)}$. Set all the symbol counts at this new node to 0 except the one corresponding to x_t which is set to 1. Initialize the REC value to RECMIN.

Comments

A crucial task in this algorithm is the choice of the encoding nodes. To clarify the thinking imagine that we have already determined the context tree $T(t-1)$ from the so far processed string $x = x_1, \dots, x_{t-1}$, but have not yet decided which context to use for encoding of x_t; ie, how far into the past string $z(i) = z_1, \dots, z_i$ we should look for the context, which also defines a path from the root of the tree to a unique node. In order to get guidance we calculate how well the various nodes on this path would have been able to compress the past symbol occurrences. To calculate the per symbol entropy at each node is not the answer, because the deeper we go the smaller the entropy gets, which would force us always to select the leaf as the coding node. Rather, we should somehow include the complexity of the node distribution in the code length calculation, which is what was done in [1] and [2]. Rather, here we do the code length calculation predictively, which automatically includes the "cost" of the model associated with each node, as it were. To increase the speed and simplify this process, we evaluate the predictive code lengths in a relative manner with help of the cumulative counter called REC [8] at each node,

$$REC_t(z(i)) = REC_{t-1}(z(i)) + I(x_t \mid z(i)) - I(x_t \mid z(i-1)),$$
$$i > 0,$$

which allows us to judge the efficiency of the considered node as compared with its father node. Because it works in a relative rather than absolute manner (ie, it accumulates the differences of the code lengths between two adjacent nodes) there is no need to take into account all the son nodes in the decision.

When the value of the REC increases, it means that the node associated with this REC is less efficient than its father node. Conversely, when the value decreases, it means that the node is more efficient than its father node. Hence, in principle we should test the sign of the derivative of the REC counter, which however is cumbersome and somewhat unreliable because of the stochastic nature of the code length differences. Instead we do the job in the indicated manner using the maximum and the minimum values of the REC counter. Because the strings require nonstationary models, the REC counters allow the constructed encoding nodes to regress and become a non-encoding node. The use of the REC counters also allows a more selective development of the tree and a more efficient usage of the memory space. In fact, since we permit encoding nodes to have son nodes, some of which in turn being turned into encoding nodes, the size and the shape of the tree get carefully matched to the information content in the source string, and the memory space allowed for the tree will be optimally used. Even when the memory space gets filled up, some nodes may still be dropped and new ones created according to the information carried by them. This way a degree of adaptation in the tree construction is achieved.

III. Results

An arithmetic coder in connection with a universal model provided by the Context Algorithm forms currently the most powerful lossless data compression package we are aware of. The compression rates are better than those obtainable with, say, a Ziv-Lempel algorithm, in particular for short and medium length files. When applied to the common grey scale test image "Lena" sampled into 256 x 256 pixels, one gets a code length of 5.1 bits/pixel. This compares favorably with the result in Ho and Gersho [9], who obtained 5 bits/pixels by applying a DCT on the same image but sampled at the resolution 512 x 512 which for the more coarsely sampled image corresponds to a greater bit rate. Furthermore, for binary image compression, algorithm Context is more efficient than the classical Q-Coder adapter with some fixed Markov model.

Algorithm Context provides also a new foundation for lossy coding, which the authors discuss in another paper. However, it can also be applied to enhance current quantization algorithms. For example, when applied to scalar quantized values, it allows a further improvement of the compression rate by a factor of about 30%, without any loss of information in the quantized string. In vector Quantized images, the algorithm Context should not be applied to the pixel components of the quantized vectors but to the 2-D field of the resulting indices. Then the compression rate improvement over ordinary VQ is typically in the range of 25-30 percent. Further, when combined with transform coding, it provides quasi-lossless coding of images for a bit rate close to 1 bit/pixel with an average deviation 0.1 bits/pixel for most parts of the image.

IV. Conclusions

In this paper we have presented a new version of the powerful algorithm Context that allows a real time implementation for multi-symbol sources. This version allows: 1) an optimal choice of the encoding nodes which no longer need to satisfy the property that all son nodes are encoding nodes; 2) linking of the size of the required tree to the subtree defined by encoding nodes. Combined with a multiplication free multi-symbol Arithmetic Coder, it represents the most efficient lossless data compression method we are aware of. Further generalizations of the algorithm are under development which permit universal modeling of all time series.

V. Bibliography

[1] J.Rissanen - "A Universal Data Compression System", IEEE Trans. Info. Theory, vol IT 29, No 5, September 83.

[2] J.Rissanen - "Complexity of Strings in the Class of Markov Sources", IEEE Trans. Info. Theory, vol IT 32, No 4, July 86.

[3] G.Furlan - "Contribution a l'Etude et au Developpement d'Algorithmes de Traitement du Signal en Compression de Donnees et d'Images", Ph.D Dissertation, Universite de Nice-Sophia Antipolis, April 90 (unpublished).

[4] J.Ziv, A.Lempel - "Complexity of Individual Sequences via Variable Rate Coding" IEEE Trans. Info. Theory, vol IT 24, No 5, September 78.

[5] J.Rissanen, G.Langdon - "Universal Modeling and Coding" IEEE Trans. Info. Theory, vol IT 27, No 1, January 81.

[6] J. Rissanen, K. Mohiuddin - "A multiplication-free multi-alphabet arithmetic code" - IEEE - Trans. Com., Vol 37, No 2, February 1989.

[7] J. Rissanen - "Stochastic Complexity in Statistical Inquiry" - World Scientific Publishing Co., Singapore, 1989.

[8] G. Furlan - "An Enhancement to Universal Modeling Algorithm Context for Real-Time Applications to Image Compression" - IEEE - Proc. ICASSP 91

[9] Y. S. Ho, A. Gersho - "Classified Transform Coding of Images Using Vector Quantization" - IEEE - Proc. ICASSP 89