A New Low-Complexity Entropy Coding Method

Ilya V. Brailovsky, Dmitry A. Plotkin
Department of Computational Mathematics and Cybernetics
Moscow State University, Moscow, Russia
Evgeny M. Kravtsunov
The Institute of Microprocessor Computer Systems,
Russian Academy of Science (IMCS RAS), Moscow, Russia

Abstract

In this paper we introduce a new method for low-complexity entropy coding, which we call Generalized Interval Transformations coding. It gives high compression rates (very close to the entropy) without using complex arithmetic operations like divisions or multiplications.

We will discuss two *algorithms* derived form the Generalized Interval Transformations *method*. The first algorithm is a low-complexity online algorithm, which proved to be very efficient for compression of Intermediate Representation data for E2k optimizing compiler. The second algorithm has the same low computation complexity however works at a greater algorithmic complexity and allows reaching coding speed (encoded bits on input byte) 2,15 on Calgary Corpus.

Keywords: *data compression, inversion frequencies,Generalized Interval Transformations, Rice-Golomb codes.*

1. INTRODUCTION

The coding technique of Generalized Interval Transformations (GITs) is based on generalization of the known “inversion frequencies” algorithm [1]. GITs operate on blocks of data and convert data to sequences of geometrically distributed integer numbers. Integer numbers are further compressed by the famous Rice-Golomb [2] codes and outputted for forming the bitstream. The point is that for Rice-Golomb codes it’s necessary to know some statistical information. Instead of using Rice-Golomb codes we can use some universal codes without these requirements. This strategy has been studied in [3] and it gives good results. In this paper, however, we propose an algorithm for online parameter estimation of Rice-Golomb codes resulting in better compression efficiency as compared to a universal integer numbers coding.

The paper is structured in the following way: first we give a definition for the whole family of Generalized Interval Transformations. Then we confine our study to online Adaptive Interval Coding algorithm and show the results and possible applications of this algorithm. After that we define Binary Interval Coding (a subfamily of GITs) and show the results of the coding on Calgary Corpus test suite.

2. GENERALIZED INTERVAL TRANSFORMATIONS

To define Generalized Interval Transformations (GIT) we first define Partial Generalized Interval Transformations and using the definition of Partial Generalized Interval Transformations we also define Full Interval Transformations as presented in [4].

2.1 Partial Generalized Interval Transformations

Let **S** be a message in alphabet $\mathbf{A} = \{a_i\}$, and let us split $\mathbf{A}$ into two nonempty subsets $\mathbf{A}_1$ and $\mathbf{A}_2$ so that $\mathbf{A} = \mathbf{A}_1 \cup \mathbf{A}_2$.

Definition 1. We will call Partial Generalized Interval Transformations (or PGITs) the transformation of **S** into three other sequences:

1) Sequence $\mathbf{S}_1$, which consists of letters from subset $\mathbf{A}_1$ in message **S**; in other words, $\mathbf{S}_1$ is a message **S** with letters from the subset $\mathbf{A}_2$ “struck out”.
2) Sequence $\mathbf{S}_2$, which consists of letters from subset $\mathbf{A}_2$ in message **S**; in other words, $\mathbf{S}_2$ is a message **S** with letters from the subset $\mathbf{A}_1$ “struck out”.
3) Sequence $\mathbf{S}_I$ of intervals between sequential entries of letters from $\mathbf{A}_1$ in **S**.

For example, let $\mathbf{A} = \{a,b,c\}$, **S** = *aabcbbccabbaca*, $\mathbf{A}_1 = \{a\}$ and $\mathbf{A}_2 = \{b,c\}$. Then the Partial Interval Transformation produces the following sequences: $\mathbf{S}_1$ = *aaaaa*, $\mathbf{S}_2$ = *bcbbccbbc* and $\mathbf{S}_I$ =0,0,6,2,1.

2.1 Full Generalized Interval Transformations

With the aid of PGITs it is possible to construct many different coding algorithms by choosing different splits of the original alphabet and applying different algorithms for further compression (transformation) of the sequences $\mathbf{S}_1$, $\mathbf{S}_2$ and $\mathbf{S}_I$. For instance, it is possible to encode sequences $\mathbf{S}_1$, $\mathbf{S}_2$ by applying a PGIT to them and splitting subsets $\mathbf{A}_1$ and $\mathbf{A}_2$ until they contain only 1 letter.

Definition 2. Full Generalized Interval Transformation (or FGIT) is a composition of Partial Interval Transformations where both of $\mathbf{S}_1$ and $\mathbf{S}_2$ are transformed by a Partial Interval Transformation if the corresponding $\mathbf{A}_i$, *i*=1,2 contains more than 1 letter.

We will call both Partial and Full Generalized Interval Transformations as Generalized Interval Transformations (GITs).

3. ADAPTIVE INTERVAL CODING

If we take $\mathbf{A}_1$ consisting of a single letter for each stage of a FGIT we will get exactly the “inversion frequencies” transformations [1]. By definitions 1 and 2, GITs (and “inversion frequencies” in particular) operate on blocks of data. In [5] for “inversion frequencies”, however, a fast online algorithm for forward and backward transformations is proposed. We take this algorithm as a basis for our coding algorithm. But instead of arithmetic encoding which is proposed in [5] as the final stage of compression, we use fast multiplication-free algorithm of Rice-Golomb codes, which exploits only addition and shift arithmetic operations. For maintaining online property of the fast transformation algorithm we propose to use the following strategy of estimating Rice-Golomb codes parameter k for intervals corresponding to the given letter: for encoding next interval use $k = \lceil \log_2 m \rceil$ where m is the average length of all previous intervals for this letter. We prove that this algorithm of adaptation gives in average $O(\frac{1}{N})$ increase in code length against the optimal Rice-Golomb code selection (N is the length of **S**). The combination of the fast algorithm for "inversion frequencies" transformation and adaptation for Rice-Golomb codes and some adaptation on the order of choosing letters (adaptation on choosing $\mathbf{A}_1$ subsets) will be referred to as Adaptive Interval Coding (AIC).

One of the possible implementations of AIC algorithm was made for the E2k optimizing compiler [6]. Intermediate Representation (IR) of this compiler for a typical task consists of stored language-specific structures, binary and text data with non-homogeneous statistics. IR data typically are not very big, but for some tasks it is necessary to built large IR files for global analysis. In order to keep working with IR data in memory instead of swapping data to disk the compiler needs a very fast and very efficient compression algorithm. With the aid of AIC it's possible to achieve coding speed 3.57 for SPECint92 IR data. "gzip", for instance, reaches coding speed 2.66 on the same files but works several times slower which is unacceptable from the practical point of view. Comparison with several other algorithms leads us to conclusion that AIC gives the best balance between coding efficiency and compression rate among a wide range of coding methods for the task of compression of E2k IR data.

Another possible use of AIC is to apply it to the output from Barrows-Wheeler Transformation (BWT) [1]. In this case coding speed 2.49 can be reached on Calgary Corpus test suite. This compression rate is smaller (about 10%) than that for a typical BWT based encoder, but AIC has less complexity than any other *online* BWT based algorithms.

4. BINARY INTERVAL CODING

Let $\mathbf{A} = \{0,1\}$ be the binary alphabet and **S** be a message in this alphabet. Let $\mathbf{A}(n)$ be all binary words under $\mathbf{A} = \{0,1\}$ with the length n. Now we can construct a split of alphabet $\mathbf{A}(n)$ into $(n+1)$ subsets of $\mathbf{A}_i(n)$ in the following way: $\mathbf{a} \in \mathbf{A}(n)$ belongs to $\mathbf{A}_i(n), i = 0,...,n$ if and only if the number of entries of "0" in $\mathbf{a} \in \mathbf{A}(n)$ is equal to i. We will call the GIT, which corresponds to the abovementioned split of the alphabet $\mathbf{A}(n)$, Binary Interval Coding (BIC) with parameter n. It is proved in [4] that the redundancy of such coding decreases while n grows in case of sources without memory.

We've implemented BIC algorithm in software and tested it on Calgary Corpus test suite. With n=2 it gives coding speed 5,06, with n=16 - already 2,35, and with n=24 - 2,15. Algorithmic complexity of the computations is of course growing with the increase of n. Today's best coding speed 2,08 [7] is better than the best results of BIC. But these first results are very promising because a good many improvements in the statistical modeling for source data are left for future.

3. CONCLUSION

Generalized Interval Transformations method proves to be efficient both for building low-complexity algorithms and for building high efficient universal compression algorithms. An example of low complexity method is Adaptive Interval Coding algorithm, which is the best-balanced compression algorithm for E2k compiler IR data. Binary Interval Coding gives promising high compression rates at the first implementation resulting coding speed 2,15 on Calgary Corpus and comes close to the world's best compression results.

4. REFERENCES

[1] Arnavut Z., Magliveras S. S. Block Sorting and Compression // Proc. IEEE Data Compression Conference, Snowbird, Utah, 1997.

[2] Golomb S. W. Run length encoding // IEEE Transactions on Information Theory, 1966, vol. IT-12, №7. – P.399-401.

[3] Brailovsky I. Flag Inversion Frequency Coding // Information Technologies, Moscow, published by New Technologies, 2002, №.11. - P. 19-25 (in Russian)

[4] Brailovsky I. Coding binary sources without a memory with a generalized interval transformation. // Chebyshevskii spornik, 2003, v.4, № .11. - P. 19-25 (in Russian)

[5] Kadach A. Efficient algorithms of lossless text data compression. PhD thesis, Siberian RAS department. - Novosibirsk, 1997 (in Russian).

[6] Babayan B. A. E2k Technology and Implementation. // Proceedings of the Euro-Par 2000 - Parallel Processing: 6th International, Jan. 2000, v. 1900/2000. - P. 18-21.

[7] Shkarin D. Improving the efficiency of PPM algorithm // Problems of Information Transmission, 2001, 34(3). – P. 44-54

About the authors

Ilya V. Brailovsky is a researcher at Moscow State University, Department of Computational Mathematics and Cybernetics. His contact email is brail@mcst.ru.

Evgeny M. Kravtsunov is a Ph.D. student at Institute of Microprocessor Computer Systems, Russian Academy of Science. His contact email is ekrav@mcst.ru.

Dmitry A. Plotkin is a 5-th year student at Moscow State University, Department of Computational Mathematics and Cybernetics. His contact email is darksun@mail.ru.