IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
APPLICATION FOR PROVISIONAL LETTERS PATENT

PREDICTIVE CODING OF GENOMIC INFORMATION DATA

BY:

Jan Voges
Erichstr. 2
30449 Hannover
Germany
Citizenship: German

Marco Munderloh
Glockenheide 1a
30855 Langenhagen
Germany
Citizenship: German

Joern Ostermann
Karla-Schmidt-Str. 14
30655 Hannover
Germany
Citizenship: German

Contact
{jvoges, munderl, office}@tnt.uni-hannover.de

Table of Contents

1 Preface

It should be understood at the outset that although an illustrative implementation of one or more embodiments are provided below, the disclosed systems and/or methods may be implemented using any number of techniques, whether currently known or in existence. The disclosure should in no way be limited to the illustrative implementations, drawings, and techniques illustrated below, including the exemplary designs and implementations illustrated and described herein, but may be modified within the scope of the appended claims along with their full scope of equivalents.

2 Introduction

Due to novel high-throughput next-generation sequencing (NGS) technologies, the sequencing of huge amounts of genetic information has become affordable. On account of this flood of data, IT costs may become a major obstacle compared to sequencing costs. High-performance compression of genomic data is required to reduce the storage size and transmission costs.

Raw sequencing data (mainly the base sequences – also known as reads - and the quality scores) is stored in so-called FASTQ files, whereas mapped sequencing data is stored in so-called SAM files. SAM files are plain-text human-readable files containing the reads and quality scores as well as mapping information with respect to some reference genome. Mapped sequencing data represented in SAM files contains more redundancy, as typically multiple reads are mapped to the same location on the reference genome. The average amount of reads mapping to the same location is referred to as *coverage*.

It has been shown by Tembe et al. [7] and Deorowicz and Grabowski [8], that splitting the data into separate streams for sequence reads, quality scores etc. (and compressing them independently) yields significant gains over general-purpose (dictionary-based [2]) programs like *gzip* or *bzip2*. All recent contributions to the topic of FASTQ and/or SAM file compression employ this scheme of separately compressing the sequence reads and quality scores (as well as the other field present in FASTQ or SAM files).

Based on statistical analysis performed on a reference data set (issued by the appointed MPEG ad-hoc group on genome compression), new proposals for sequence and quality score compression in SAM files have been developed.

2.1 Outline

We propose a sequence compressor which assumes aligned (position-sorted) data. Our sequence compressor combines the mapping positions, the CIGAR strings, and the actual nucleotide sequences to implicitly assemble local parts of the donor genome to compress the sequence reads. The compressor does not need a reference genome and is able to perform compression “on-the-fly” using solely a *sliding window* (permanently updated short-time memory of arbitrary size) as context for the local assembly.

Regarding the compression of quality scores, we propose predictive compression schemes (which may be based on e.g. Markov models) with a variable or fixed context to boost prediction performance. Due to the large alphabet of quality scores (and thus a huge memory consumption of any prediction model), we furthermore propose employing FIR filter-based compression for quality scores. The proposed FIR filter for quality scores can also be applied on any other data present, e.g. locations, cigar strings, sequence reads, or auxiliary data.

3 Block-based Compression of Nucleotide Sequences

3.1 Prior Work

Current implementations (e.g. *Quip* [6] and *sam_comp* [3]) use an order-2 arithmetic coder (AC) to compress the nucleotide sequences. In order to exploit the redundancy present in the data, an arithmetic coder needs *a priori* knowledge. This practically excludes random access to the compressed data as the prediction performance increases with the amount of data the prediction model can be trained on.

The authors of *DeeZ* [5] made a new approach, called "local assembly". *DeeZ* nevertheless performs the local assembly with respect to an external reference (given in FASTA format) and without subsequent AC.

The authors of *Quip* [6] implemented an assembly-based compression scheme for nucleotide sequences, too. They use by default the first 2.5 million reads to assemble so-called contigs, which are then used in place of a reference sequence to encode aligned reads. This algorithm does not require a reference – this means that the resulting compressed files are entirely self-contained – but has the drawback that the best match of a specific read to the previously assembled reference has to be found in the fixed non-adaptive contig repository.

3.2 Proposed Compression Mechanism

We propose a compression scheme using a short-time memory of arbitrary size which may be implemented as a sliding window to perform a so-called "local implicit assembly". In contrast to *DeeZ*, our compressor does not require a reference genome and is able to work on smaller block granularities as it uses a short-time memory as reference. In contrast to *Quip*, we do not need any reads to assemble so-called contigs in advance and we are able to adapt to changes in data statistics. We perform the compression using only the currently available nucleotide sequences in the short-time memory. This short-time memory can be re-initialized at any time, enabling random access to any part of the compressed data without the need of decoding the preceding compressed data.

Our approach is based on correlations between consecutive reads. As all reads in a SAM file are aligned to their mapping position, all reads mapped to the same position should be similar, except for gene mutations (SNP's) or sequencing errors, respectively mapping errors. These aberrations are coded in the CIGAR string. An example is given in Figure 1. Consecutive reads are plotted line by line. All reads mapped to the same position are aligned. As SAM files are stored sorted by mapping position, a read mapping to a different location than the read before leads to a step in the plot. If plotted over a large block, this leads to the staircase property visible in the plotted figure.

Our approach proposes to encode sequence reads block-wise exploiting the joint coding of the mapping positions, the CIGAR strings and the sequence reads itself, whereas the block size might be fixed or variable and of arbitrary size (which includes the compression of the input file as a single block). The first sequence read in a block is encoded without prediction, as well as the corresponding mapping position and the CIGAR string. However, it might also be coded against some arbitrary reference. Some subsequent read $seq_i, i > 1$, is then aligned to a previous read $seq_j, j < i$, using its CIGAR string and its mapping position.

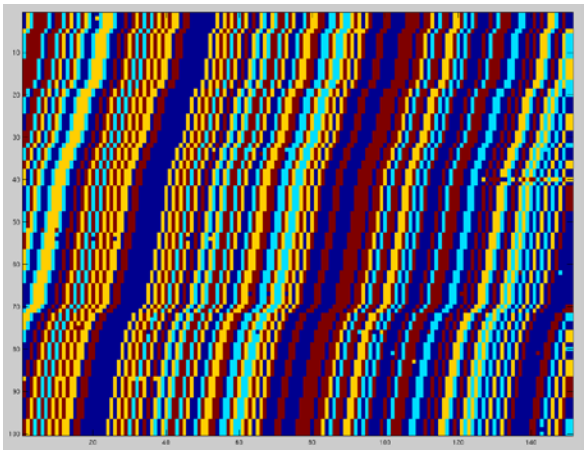

Figure 1: Consecutive nucleotide sequences from a SAM file. The different colors represent the nucleotides.

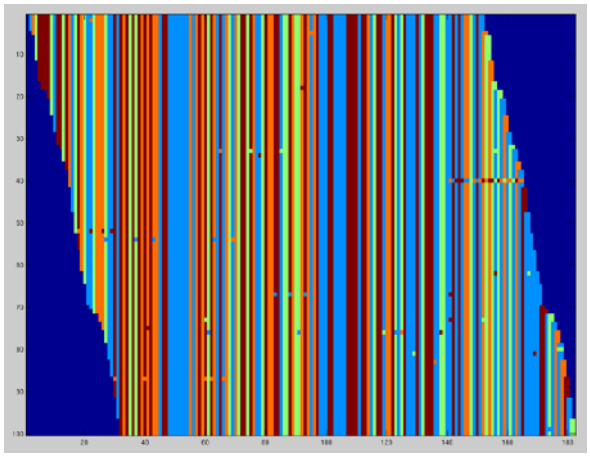

Figure 2: The nucleotide sequences after the expand function, stored as a sliding window in the circular buffer. The noisy 'dots' represent genome mutations or sequencing errors described in the CIGAR string.

Thereafter, we compute the difference d_i (containing the differences of the mapping position and the nucleotide sequence from the aligned read seq_i to read seq_j and pass it to an entropy coder, e.g. an order-0 arithmetic coder. At the decoder, read seq_i can be reconstructed from read seq_j by applying d_i to reconstruct the nucleotide sequence, the CIGAR string, and the mapping position.

A possible encoder structure is shown in Figure 3.

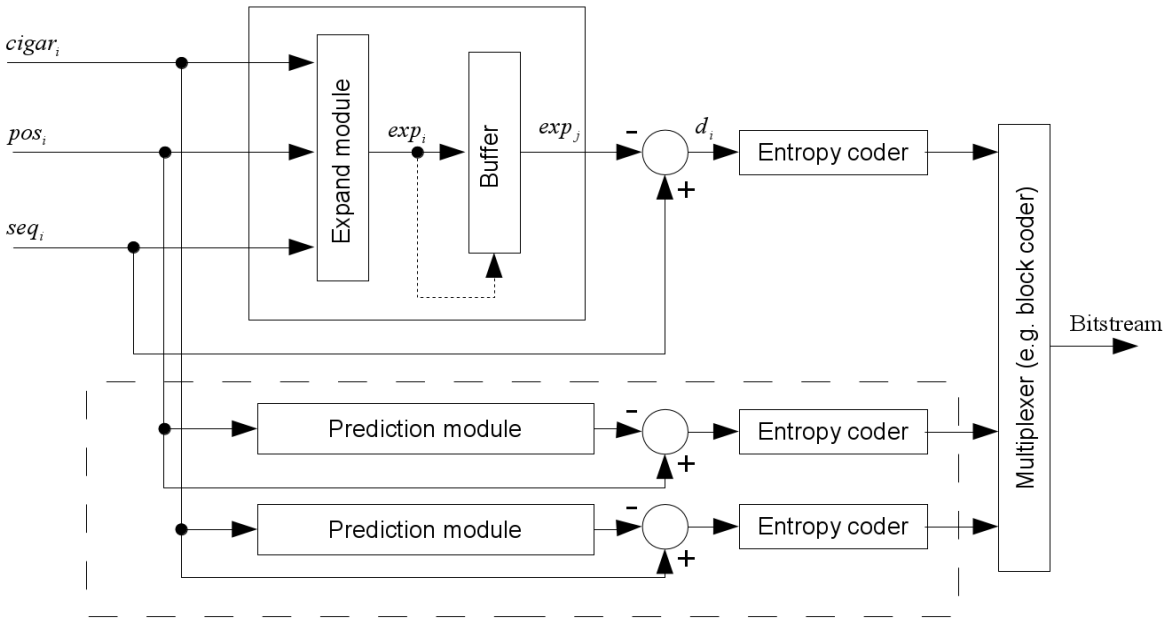

Figure 3: Possible encoder structure of the proposed sequence compression algorithm

The alignment of read seq_i to read seq_j is done with a function we call expand. The output of expand is called exp_i.The CIGAR strings and mapping positions of both reads are used to unwind the sequence such that they would both directly align (please refer to the SAM specification [1], Section 1.4ff for a detailed explanation of the CIGAR string syntax). In general, unwinding the sequence consists of a series of insertion, deletion, and modification operations as described by the CIGAR string. A possible output of the expand function is visualized in Figure 2. Consequently, expand condenses the position, the CIGAR string, and the sequence into a joint representative code word. As an

alternative, the last step could be omitted and these SAM fields could be processed separately. A short example of the working principle of expand is given in Figure 4.

```
POS CIGAR      SEQ
7   8M2I4M1D3M TTAGATAAAGGATACTG < read 1
9   3S6M1P1I4M AAAAGATAAGGATA    < read 2

      6789... < POS
       8M      2I4M  1D3M
       TTAGATAAAGGATA_CTG < read 1 expanded
      AAAAGATAAGGGATA     < read 2 expanded
      3S 6M    1P1I4M
```

Figure 4: Working principle of the function expand

The previous read seq_j is by default read seq_{i-1}. This context read $seq_j = seq_{i-1}$ might be highly erroneous or not aligned to the reference. We address this issue by keeping track of *N* previous reads (sliding window) to be able to select the best matching expanded read exp_{j} to encode read seq_i. Sequence reads which are not aligned to the reference might be directly passed to the entropy coder or retained and encoded separately later, e.g. at the end of each block. The selected matching read might be signaled or estimated at the decoder. A possible implementation of the algorithm in pseudo code is shown in Figure 5.

```
Input: block_size, threshold, pos[L], cigar[L], seq[L], L (no. of SAM records)
Output: binary stream (from entropy coder)
new circbuf(N); // circular buffer of size N (sliding window)

l = 0;          // line count
b = block_size; // block line count

while (l < L)
| if (b == block_size) then // start new block
| | clear_circbuf();
| | reset_entropy_coder();
| | entropy_coder(pos[l], cigar[l], seq[l]);
| | circbuf.push(pos[l], cigar[l], seq[l]);
| | b = 1;
| else
| | if (cigar(l) != '*') then // sequence is mapped
| | | min_entropy = +inf;
| | | do
| | | | seq_exp = expand(pos[l], cigar[l], seq[l]);
| | | | (pos_ref, cigar_ref, seq_ref, ref_index) = circbuf.get_next();
| | | | ref_exp = expand(pos_ref, cigar_ref, seq_ref);
| | | | d = compute_differences(seq_exp, ref_exp);
| | | | if (entropy(d) < min_entropy) then
| | | | | min_entropy = entropy(d);
| | | | | ref_index_best = ref_index;
| | | | | d_best = d;
| | | | endif
| | | while (entropy(d) > threshold) && circbuf.has_next();
| | | entropy_coder(ref_index_best, d_best);
| | else
| | | entropy_coder(pos[l], cigar[l], seq[l]);
| | endif
| | circbuf.push(pos(l), cigar(l), seq(l));
| endif
| b++;
| l++;
endwhile
```

Figure 5: Algorithm outline

4 Predictive Coding of Quality Scores

4.1 Prior Work

The quality scores yield the largest first-order entropy among all field present in a typical SAM file, mainly due to their large alphabet. The authors of *Fqzcomp* [3] state amongst other things the following dependencies for a sequence of quality scores:

- Any score q_i has a strong correlation to the direct previous quality scores $q_{i-1}, q_{i-2}, \dots$, decreasing the further back we go [4].
- Sequences as a whole tend to be good or bad.

Current SAM compression implementations exploit this observations by employing a Markov model with the memory ranging over the direct predecessors in the quality score stream.

4.2 Modeling the Source with a Markov Chain

Common SAM file compression approaches [4, 5] noted, that quality scores usually contain a trailing sequence of low quality values. We made this observations, too, and propose first to trim the trailing low quality values.

Current SAM file compressors model the quality score source as a Markov chain, predicting the current symbol with a maximum likelihood predictor using the *N* direct predecessors.

The maximum likelihood predictor selects in each state $q_1, \dots q_N$ the symbol a_i with the greatest conditional probability *P* as prediction value $q_{N+1}^{(p)}$:

$$P\left(q_{N+1}^{(p)} = a_i \middle| q_1, \dots q_N\right) \geq P(q_{N+1} = a_k | q_1, \dots, q_N) \forall k$$

4.2.1 Drawbacks

The maximum likelihood predictor consumes a huge amount of memory since we have to keep track of $k^N * k$ relative symbol frequencies (having an alphabet of size *k*). This model also needs a large amount of data to train in order to tightly fit the model to the data.

4.3 Proposed Compression Schemes

As an alternative to building the prediction memory during the decoding process, we propose the possibility to store the prediction table (generated during compression) e.g. in the file header to speed up decompression as an alternative or in combination. The prediction tables for the individual blocks may then be encoded, e.g. delta-encoded with subsequent AC. This enables random access to the compressed data while retaining the compression performance of the prediction model (e.g. Markov model) by providing the fully trained prediction memory, or parts of it, (built from the full input data at the encoder) to the decoder, which can then be directly used from the start of the decoding process.

The following subsections describe alternatives to predictive quality score compression which might be combined to yield the best compression performance. As an additional alternative, we propose to directly compress the quality scores without prediction employing solely an entropy coder, e.g. an AC of arbitrary order.

4.3.1 Extended Line Context Approach

We made the observation, that in some datasets subsequent quality score lines are “similar” in terms of a small Levensthein distance between pairs of quality score lines.

We respond to this using a so-called *line context*, i.e. a structure holding some number of previous lines from the current block (this is also known as “sliding window”). The prediction context (i.e. a

Markov model) is divided into one part containing the memory locations on the current line ("intra-line memory") and one part containing the memory locations on some specific line from the line context ("inter-line memory") as depicted in Figure 6.

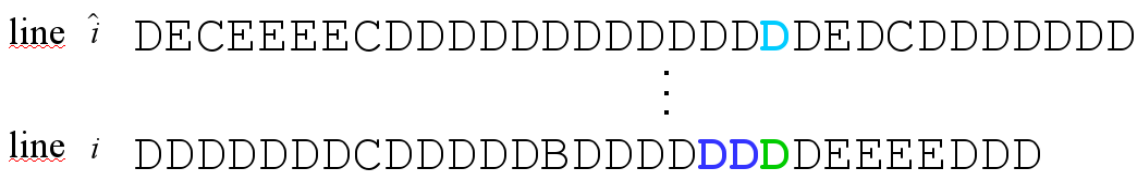

Figure 6: Visualization of the intra-line (dark blue) and inter-line (light blue) context used to predict the current symbol (green).

Choosing the right line from the line context to boost prediction performance is crucial. We propose to compute an identification vector for each quality score line (or parts of it) in the line context. As an example, this could be a three-dimensional vector, containing e.g. the length, the arithmetic mean, and the variance of a quality score line. In the process of encoding line *i*, we first select the one specific line $\hat{\imath}$ from the line context with the least identification vector distance, whereas any known vector distance measurement might be applied, e.g. the three-dimensional Euclidean distance. The vector components might be individually weighted. Consecutively, the intra-line memory is placed on line *i* and the inter-line memory is placed on the selected line $\hat{\imath}$ from the line context.

4.3.2 Dictionary-based Substring Matching Approach

To encode the quality score $q_{i,j}$ in column *j* and line *i*, we propose to select the substring *s* of *N* preceding symbols $s = q_{i,j-N-1}, \dots, q_{i,j-1}$. We then propose to search for a similar substring $\hat{s} = q_{\lambda,\gamma}, \dots, q_{\lambda,\gamma+N} \equiv s$ in the line context and to use $q_{\lambda,\gamma+N+1}$ as the prediction value $q_{i,j}^{(p)}$ for $q_{i,j}$.

Alternatively, we propose to use a prediction value $q_{i,j}^{(p)}$ calculated as a mean or median or as an arbitrary combination from *M* substring matches $\hat{s}_m, 1 \leq m \leq M$, which might be weighted by any reasonable weighting factors, or any other reasonable measure, whereas any suitable norm might be applied. This principle is briefly pointed out in Figure 7.

Figure 7: The green symbol is predicted from the blue symbol, if a matching substring containing M=5 symbols (red) is found in the line context.

This approach for quality score compression can also be applied on any other data present, e.g. locations, cigar strings, sequence reads, or auxiliary data.

4.3.3 Markov Context-Extending Substring Matching Approach

As an alternative, we propose to extend the context of any of the above mentioned predictors with the prediction value ${}^{s}q_{i,j}^{(p)} = q_{\lambda,\gamma+N+1}$ found by substring matching within the line context.

Regarding the prediction model from Section 4.2, or any other prediction model, we therefore propose to extend the state/context $q_1, \ldots, q_n$ with ${}^{s}q_{i,j}^{(p)}$ to form an extended state $q_1, \ldots, q_n, {}^{s}q_{i,j}^{(p)}$. Regarding the prediction model from Section 4.3.1, we propose to extend the context (consisting of the inter- and intra-line memories) with ${}^{s}q_{i,j}^{(p)}$.

4.4 Linear Prediction with FIR Filters

Due to the large alphabet of the quality scores (typically approximately 30 symbols, although more are theoretically possible), Markov models are limited to some low order *N* (typically $N \leq 3$).

As an alternative, we propose to use an FIR filter to predict the current symbol. While it is on the one hand costly to compute the FIR filter coefficients, on the other hand, filter coefficients might be shared between multiple quality score lines.

The basic layout of a predictive encoder can be viewed in Figure 8.

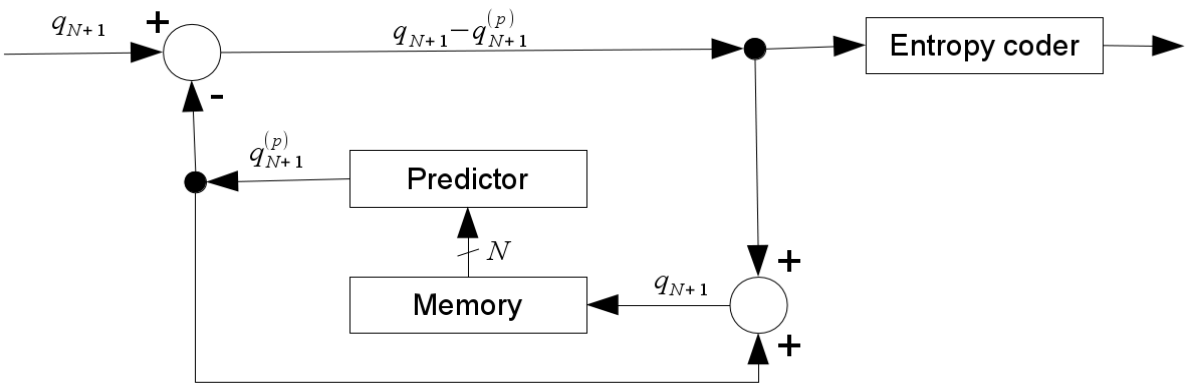

Figure 8: Linear predictive encoder schematic

The linear predictor computes the current prediction value $q_{N+1}^{(p)}$ by evaluating the linear series of previous symbols q_n weighted with predictor coefficients a_n.

$$q_{N+1}^{(p)} = \sum_{n=1}^{N} a_n q_n$$

One possible approach to compute the predictor coefficients is to minimize the mean squared error which yields the following condition to compute the filter coefficients.

$$\frac{\partial}{\partial a_n}\left(E\left[\left(q_{N+1} - q_{N+1}^{(p)}\right)^2\right]\right) = 0$$

Thus, we obtain *l* sets of predictor coefficients for a quality score line of length *l*. Using the line context from Section 4.3.1, it could be beneficial to share previously computed predictor coefficients among multiple lines.

As an alternative to the minimum-mean-square-error filtering it should be possible to employ any other known method to compute the predictor coefficients, e.g. with minimum-variance prediction.

This approach for quality score compression can also be applied on any other data present, e.g. locations, cigar strings, sequence reads, or auxiliary data.

4.5 Entropy Coding of Prediction Errors

The histogram of the prediction errors approximately shows a two-sided geometric distribution. We therefore propose to use subsequent Rice coding [9] as entropy coding step to obtain the binary representation of the data. This reduces the encoder complexity, as recent compressors such as *Quip* [6] employ arithmetic coding.

5 Remark

While several embodiments have been provided in the present disclosure, it should be understood that the disclosed systems and methods might be embodied in many other specific forms without departing from the spirit or scope of the present disclosure. The present examples are to be considered as illustrative and not restrictive, and the intention is not to be limited to the details given herein. For example, the various elements or components may be combined or integrated in another system or certain features may be omitted, or not implemented.

6 References

[1] The SAM/BAM Format Specification Working Group: Sequence Alignment/Map Format Specification.

[2] Ziv., J. & Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on information theory 23, 337-343 (1977).

[3] Bonfield, J.K & Mahoney, M. V.: Compression of FASTQ and SAM Format Sequencing Data. PloS ONE 8, e59190 (2013).

[4] Christos Kozanitis, Chris Saunders, Semyon Kruglayak, Vineet Bafna, and George Varghese: Compressing genomic sequence fragments using SlimGene. Journal of Computational Biology: A Journal of Computational Molecular Cell Biology, 18(3):401-13, March 2011. ISSN 1557-8666. doi: 10.1089/cmb.2010.0253.

[5] Faraz Hach, Ibrahim Numanagic & S. Cenk Sahinalp: DeeZ: reference-based compression by local assembly. Nature Methods, Vol. 11, No. 11, November 2014, pp 1082-1084.

[6] Daniel C. Jones, Walter L. Ruzzo, Xinxia Peng & Michael G. Katze: Compression of next-generation sequencing reads aided by highly efficient de-novo assembly. Nucleic Acids Research, 2012, 1-9. doi: 10.1093/nar/gks/754.

[7] Tembe, W., Lowey, J. and Suh, E.: G-SQZ: compact encoding of genomic sequence and quality data. Bioinformatics, 26, 2192-2194, (2010).

[8] Deorowicz, S. and Grabowski, S.: Compression of DNA sequence reads in FASTQ format. Bioinformatics, 27, 860-862. (2011).

[9] Robert F. Rice and James R. Plaunt: Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data, IEEE Transactions on communication technology, Vol. COM-19, No. 6, December 1971.