Lossless compression of Poisson distributed variable for AMIGA/Auger

Gonzalo de la Vega[a], Ricardo Sato[c], Fernando Contreras[a], Beatriz García[a,b], Manuel Platino[a], Agustin Lucero[a], Federico Suarez[a], Mariela Videla[a], Oscar Wainberg[a]

[a] *Instituto de Tecnologías de Detección de Astropartículas*
[b] *Universidad Tecnológica Nacional Facultad Regional Mendoza*
[c] *Pierre Auger Observatory*

Abstract

A new wireless communications system is being developed for the AMIGA project, an extension to the Pierre Auger Observatory. In order to minimize the required data rate and data frame size we explore the possibility of compressing part of the data using a simple method with low memory and CPU requirements. The method proves to be capable of saving more than 20% of the required transmission speed. By reducing the data frame size, we are able to minimize the number of frames per second reducing the overall error rate and allowing for the use of standard commercial technology with minimal customization.

Keywords: information theory, lossless compression, Rice coding, Golomb coding, Poisson process

1. Introduction

The Auger Muon Infill for the Ground Array (AMIGA) [2] project is an enhancement to the Pierre Auger Observatory (PAO) [1] that requires a new communications system in order to avoid overloading the PAO original communications system. AMIGA will generate data that is similar to that of PAO except that some extra information will be sent when an event is detected.

The Pierre Auger Observatory can be viewed as a large, sparse coincidence detector. The detector is comprised of a large number of Surface Detectors (SD) capable of precisely determine the moment at which a particle passes through them. This detected particles are sub-products of the collision of a High Energy Particle (HEP) with the

Email addresses: gonzalo.delavega@iteda.cnea.gov.ar (Gonzalo de la Vega), rsato@auger.org.ar (Ricardo Sato), fcontreras@auger.org.ar (Fernando Contreras), beatriz.garcia@iteda.cnea.gov.ar (Beatriz García), manuel.platino@iteda.cnea.gov.ar (Manuel Platino), agustin.lucero@iteda.cnea.gov.ar (Agustin Lucero), federico.suarez@iteda.cnea.gov.ar (Federico Suarez), mariela.videla@iteda.cnea.gov.ar (Mariela Videla), oscar.wainberg@iteda.cnea.gov.ar (Oscar Wainberg)

Earths atmosphere components. It is the coincidence between three or more SD what may indicate a HEP event.

The coincidence detection is performed in a Central Data Acquisition System (CDAS) and because of that some data must be transferred to this centralized facility in order to evaluate the coincidence. The signal detected by the SD is filtered so that only a signal with the proper shape is considered as a coincidence candidate, this candidates being called T2. The data corresponding to the candidate event is stored temporarily and a time tagged. Every second a list of T2 time tags is collected and sent to CDAS for coincidence evaluation.

Although the centralized coincidence detection is the way PAO currently works, the results presented here may be used for a distributed system, as long as the same data is being transferred.

Potentially interesting events or candidates occur at random intervals independent from each other so it is possible to consider the arrivals of T2 as a Poisson process and the probability distribution would be represented by a decaying exponential, or as the data is being discretely sampled, a geometric distribution.

In order to take advantage of this it can be useful to consider the difference between time tags instead of the whole offset within the second as they are used currently in PAO communication system.

Once the statistical model is parameterized it is possible to use information theory to analyze possible compression strategies.

2. Data structure

Each T2 can be uniquely identified by its time tag and a SD identification. Additionally some extra information is included describing some characteristic of the event. For this reason all the SD has to do is send a list of time tags with the corresponding information. Each event is described in the list by a two field entry [3]:

Energy	Microsecond time tag
4 bits	20 bits

The four ”energy” bits include some description of the event occurred at the listed microsecond. The time tag indicates the amount of microseconds since the beginning of the last second according to the GPS timing. To represent this number 20 bits are needed. If the difference between a time tag and the previous is stored, the number would be in average 20 times smaller as there are 20 entries to the list in average. This would mean 16 bit in average, with a maximum of 20 bits. The 20 bits of the time tag could be split in 4 parts, lets call them Most Significant Nibble (MSN), Middle Byte (MB) and Least Significant Byte (LSB):

MSN	MB	LSB
4 bits	8 bits	8 bits

These timestamps are then packed in a list once a second. This list is comprised by the corresponding GPS second followed by the time stamp entries:

$GPSSecond = TTag_0$
$TTag_1$
$TTag_2$
...
$TTag_n$

The list is sent once per second, within frame along with protocol headers.

3. Information content

As a first step we eliminate the redundancy that comes from the data being stored as an offset from the beginning of the GPS second. If the time elapsed between an event and the next is considered, then it is a Poisson process we are looking at.

Each entry to the list becomes then:

$$\Delta TT_n = TTag_n - TTag_{n-1} \tag{1}$$

When discretized by sampling, the exponential distribution becomes a geometrical one. Considering that, the probability distribution is given by:

$$P(n) = (1-p)p^n, n \geq 0, 0 < p < 1 \tag{2}$$

The information entropy of a discrete variable n, in bits is:

$$H = -\sum_n P(n) log_2[P(n)] \tag{3}$$

In the case of the geometrical distribution, the entropy can be computed from the closed form expression:

$$H = -log_2(1-p) - \frac{p}{1-p} log_2(p) \tag{4}$$

To obtain the value of p it is necessary to express ΔTS in a way that it represents the variables as seen by the algorithm, i.e. not in actual time, but in sample time. This is done by considering that at 2^{20} samples per second, the average ΔTS represent:

$$\lambda = \frac{1}{\Delta TS \times 2^{20}} \tag{5}$$

and

$$p = e^{-\lambda} \tag{6}$$

From the available data, the average $\Delta TS = 45.9ms$ is obtained.

Filling all the values the information entropy is $H = 17.05bit$, which is the target to be approached to, in the sense that an ideal compression method would result in an average code length of 17.05 bits instead of 20 bits.

4. Optimal compression with Golomb coding

The Golomb code [5] is a statistical method for data compression. If a number n is to be compressed, it is separated into a quotient $q = \lfloor \frac{n}{m} \rfloor$ and a remainder $r = n - qm$ so that $n = mq + r$. The value of q is coded in unary while r is coded with the help of a third value $c = \lceil log_2 m \rceil$. The first $2^c - m$ values ar coded as unsigned integers with $c - 1$ bits while the rest are coded in c bits.

For this particular case it is possible to generate optimal Golomb code [4] for p and m satisfying:

$$p^m + p^{m+1} \leq 1 < p^m + p^{m-1} \tag{7}$$

In order to produce the Golomb code it is necesary split the input value in two parts, as briefly explained previusly. The way to split the input is defined by the m parameter, following the conventions in [5], as we will from now on:

$$m = \left\lceil -\frac{log_2(1+p)}{log_2(p)} \right\rceil \tag{8}$$

The resulting value is $m = 33434$, with which, an optimal code should be obtained.

5. Simplified alternative with Rice coding

Golomb coding uses a tunable parameter m to divide an input value into two parts: q, the result of a division by m, and r, the remainder.

In the case of a Golomb code where $m = 2^k$, for any integer k, it is called a Rice code, which is much simpler to code and decode. In short, in Rice coding, the most significant part of the variable is coded using unary coding while the least significant is stored unmodified. In the particular case of the data we are dealing with, we found that $m = 33434 = 2^{15.03}$ which turns out to be very convenient, because it is almost a power of two. In fact, as will be shown later, using $m = 2^{15}$ has negligible impact over the compression efficiency.

For the Rice code, the two parts, q which is coded, and the reminder r, are represented by:

$$q(n) = \left\lfloor \frac{n}{m} \right\rfloor \tag{9}$$

$$r(n) = n - q(n)m \tag{10}$$

The greatest advantage of using a value of m that is a power of two, is not only that makes the algorithm simpler, but also that such division is achieved by shifting bits and the remainder is left untouched.

In practice, Rice coding can be implemented by just taking the 5 most significant bits, and storing them using unary coding, and just storing the rest of the bits unmodified.

With unary coding, for any symbol n, the code length L_n can be easily calculated as $L_n = n + 1$ and the code lengths of all symbols along with its probability $P(n)$

of occurrence will give the expected size S for a large sample. If the alphabet can be represented with b bits, then $N = 2^b$, codes can be generated, and:

$$b = \lceil log_2(m) \rceil \tag{11}$$

The general formula for the compressed size a given number of bits is given by:

$$S = \sum_{n=0}^{N-1} L(n)P(n) = \sum_{n=0}^{N-1} (n+1)(1-p)p^n \tag{12}$$

Where now:

$$\lambda = \frac{1}{\Delta TS \times 2^b} \tag{13}$$

It is very easy to compute the summation with a mathematical software, but with a little algebra, and using the formula for the differentiation of the geometric series, it is possible to get a closed form expression:

$$S = \frac{1 - p^{N+1}}{1-p} - (N+1)p^N \tag{14}$$

For $m = 2^{15}$ and p computed with λ from equation 13, then $S = 2.078$ which added to the 15 uncompressed bits, gives 17.078 for a variable with an information entropy of 17.050 bits. Clearly, Rice coding is very close to the ideal.

5.1. Further simplification

There is an option to code q with a shorter unary code. For Huffmans algorithm to generate a unary code for an exponentially distributed variable, it is necessary that $p < \frac{1}{2}$, which gives:

$$P(n) > \sum_{k=n+1} P(k) \tag{15}$$

This means that for any given value, its probability is higher than the sum of all the higher values. This holds already for m as we have calculated before, and also for larger values.

If we were willing to sacrifice some compression efficiency, the compression algorithm could be even simpler if only the 4 most significant bits are to be compressed. This choice would allow to just copy the 16 least significant bits, which are two whole bytes, and store the 4 most significant bits, or half byte, with unary coding. In terms of the Golomb-Rice algorithm, taking the 4 most significant bits means shifting 16 bits to the right, also equivalent to dividing by $m = 2^{16}$.

Using equation 14 and taking $b = 4$, those 4 bits can be compressed to 1.37 bits. Adding the uncompressed 16 bits, the result is 17.37 bits instead of 17.05, which is not bad considering the great simplification.

6. Near optimal compression with Rice coding

We have so far analyzed optimal compression using Golomb coding, and alternatively Rice coding taking advantage of the fortunate situation that $m \simeq 2^{15}$. But as we are going to show, using:

$$m \leq \left\lceil -\frac{log_2(1+p)}{log_2(p)} \right\rceil \tag{16}$$

also leads to results very close to optimal, with the advantage that the results are good even for geometrical distributions where $m \neq 2^k$ for integer k.

We have stated that the Huffman algorithm generates a code of length $n+1$ for n if $p < \frac{1}{2}$. If $p \leq \frac{1}{2}$ then the Golomb algorithm can be used, but we have also seen that Rice coding is simpler to implement.

For $p \leq \frac{1}{2}$ the Huffman algorithm generates a code of the same length $s(n)$ for various values. This length for a given value n is given by:

$$s(n) = \left(\left\lceil \frac{n+1}{2^l} \right\rceil + l \right) (1-p)p^n \tag{17}$$

$$l = b - B + \lfloor log_2 m \rfloor \tag{18}$$

with $B = 20$ being the amount of bits used for uncompressed coding and b the amount of bits taken for compression.

If $l = 1$ then both $n = 0$ and $n = 1$ will be coded with length $c(n) = 2$ and $n = 2$ and $n = 3$ with $c(n) = 3$ and so on.

7. Compression of the "Energy" nibble

The field comprised by the 4 most significant bits of the T2 list entry is called "Energy". These are in fact tag bits for the time stamp that cannot be easily modeled. In any case, this nibble has very simple statistics:

Table 1: probability for the values of the "Energy" nibble

Nibble value	Probability
1	0.878
9	0.122
other	$< 10^{-5}$

We will only consider the compression in the case of the values 1 and 9, since not using compression otherwise should have no sensible impact due to the extremely low probability. Also note that although in both 9 and 1, the least significant bit is 1, masking it is of no use, as there ares still only 2 values considered.

The actual entropy of this field, as computed with equation 3, is $H(Energy) = 0.519$, about half a bit. This suggests that using a bit to code the nibble, is already a waste. We will consider then packing more than one entry and coding those groups of bits with Huffman's algorithm.

Table 2: coding of the "Energy" nibble taken in pairs

Energy pair	Probability	Permutations	Code	Code length	Code frequency
1,1	0.771	1	1	1	0.771
1,9	0.107	2	011,010	3	0.214
9,9	0.015	1	00	2	0.015

Considering the code length and its probability as before, we can expect a coded size of $1.44bit$ for every 2 entries, giving an average of $0.72bits/entry$ which is much better than 4 but still high compared to the information entropy.

By packing in groups of four, the result is as follows:

Table 3: coding of the "Energy" nibble taken in groups of 4

Energy pair	Probability	Permutations	Code length	Code frequency
1,1,1,1	0.595	1	1	0.595
1,1,1,9	0.082	4	4	0.330
1,1,9,9	0.011	6	5.67	0.069
1,9,9,9	0.002	4	6	6.3×10^{-3}
9,9,9,9	0.002	1	4	2.0×10^{-4}

The expected average code size is $2.34bit$ every 4 entries, giving $0.585bit/entry$ which can be considered reasonably efficient.

8. Method comparison

With a proper metric it is possible to compare the presented methods to evaluate their efficiency. We should bear in mind that the full efficiency is given by the entropy, which is $H = 17.05bit$ in this case, representing an efficiency $\eta = 1$, and that $20bit$ means $\eta = 0$. To get a clear representation we will consider how many bits in average, from the compressible are effectively compressed by each method. The compression efficiency η_c is then given by:

$$\eta_c = \frac{uncompressed\_size - compressed\_size}{uncompressed\_size - entropy} \tag{19}$$

The metric can then be used to compare the methods for a large number of entries. The result is shown in table 4 showing that the "overrated"($m \leq 2^{14}$) is as good as the "strict" ($m = 2^{15}$) Rice, with the advantage of being more flexible, and both are better than "underrated" ($m = 2^{16}$) Rice.

A similar comparison can be made for the "Energy" nibble.

8.1. Performance for short T2 lists

As the actual T2 lists are short, it should be considered that the final compressed list must be stored in an integer number bytes, and the padding bits will represent a non-negligible part. It has been stated before that the arrival of potential events (T2) is

Table 4: compression method comparison for "large" T2 lists

Method	m	Expected size	Efficiency
Rice	2^{15}	17.078	0.9905
Underrated Rice	2^{16}	17.370	0.8915
Overrated Rice	2^{14}	17.078	0.9905

Table 5: compression method comparison for "large" Energy nibble lists

Method	Expected size	Efficiency
Singlets	1.122	0.8267
Pairs	0.722	0.9418
Quartets	0.586	0.9807

a Poisson process and considering a fixed interval of 1 second, the probability density function is a Poisson distribution.

It can be seen in figure 1 that the Poisson distribution fits very well the data. It is important to notice that in order to make the comparison it is necessary to consider an extra T2 generated by the cut of the list at the beginning of each second.

$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{20}$$

It is clear that the probability of the list being 21 entries long is essentially as likely as 19 or 23, and actually not very much likely that being 15 or 25. In order to properly analyze all these possibilities the compression efficiency must be considered for the average arrival time corresponding to each list size.

In order to fully account for the compression of various list lengths, the previously described method can be used. Lets call the length of a T2 list k, then its probability $P(k)$ of occurrence is given by equation 20. From now on Rice coding will be considered so its compressed size $S(k)$ is given by equation 14, and equation 13 should be modified to reflect the arrival interval for that second, so it is a function of k:

$$\lambda(k) = \frac{k}{2^b} \tag{21}$$

The size S_T of a large number of compressed short T2 lists would be given by:

$$S_T = \sum_{k=1}^{\infty} P(k)S(k) \tag{22}$$

It is also possible to do this using a different value of m for different list length, making b also a function of k, so that for equation 14:

$$N(k) = 2^{b(k)} \tag{23}$$

A comparison in table 6 showing that $b = 5$ is the best solution, in concordance with what was calculated in section 5.

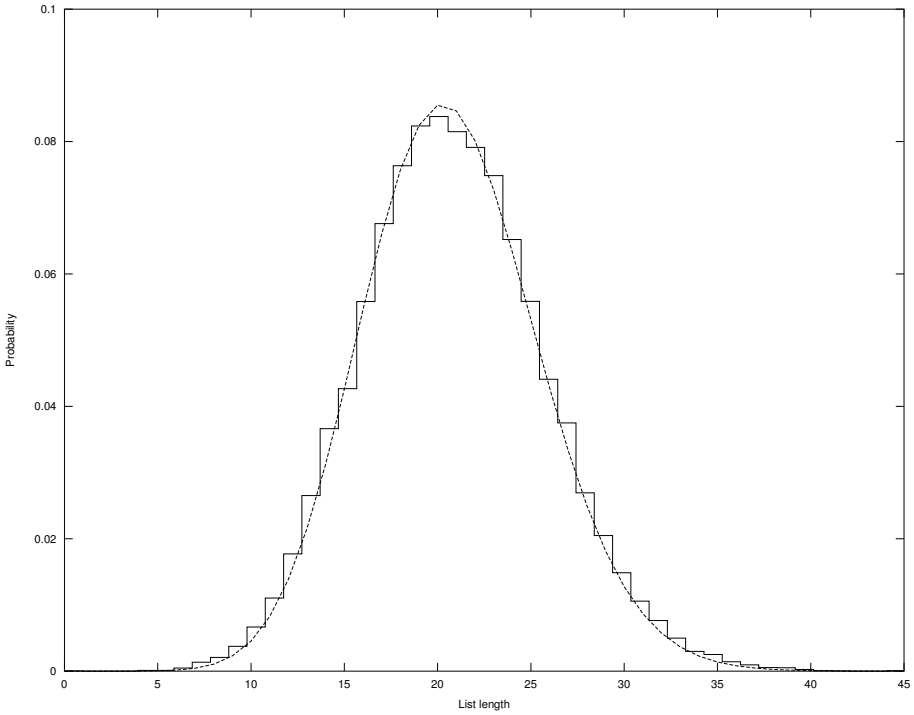

Figure 1: Poisson probability distribution function and histogram of the real data.

Table 6: Compression performance of Rice coding for short T2 lists with different values of m

m	Expected size	Efficiency
4	17.382	0.8876
5	17.100	0.9830
6	17.616	0.8081
m(k)	17.195	0.9508

The overall performance with Rice coding for the time stamps and taking pairs for compressing the "Energy" nibble, the total 24 bits can be compressed, in average, to $17.100bits + 0.722bits = 17.822bits$ from an information entropy of $17.050bits + 0.519bits = 17.569bits$ and an overall efficiency of $\eta_c = 0.9607$. This represents a reduction of about 25% of the T2 payload.

9. Conclusions

We have studied the possibility of compressing the T2 lists that comprise the bulk of the PAO data carried by the communications system. Several methods were studied focusing in compression performance and implementation simplicity, and showing that with very low computational cost it should be possible to save about 25% of the

payload traffic, increasing the number of stations per channel or freeing radio time for retransmissions. The strength of the method relies on the lack of need of a compression dictionary, so it can be used for small pieces of data for which the size of a dictionary would be too large in comparison. Additionally, the packet size could be reduced to help the implementation of a communications system based on the IEEE 802.15.4 protocol which allows for a payload of 125 Bytes, smaller than the 150 Bytes from the original PAO design. The described method can be also used for any exponentially distributed variable to assess the convenience of utilizing compression.

[1] J. Abraham et al. [Pierre Auger Collaboration], Properties and performance of the prototype instrument for the Pierre Auger Observatory, Nuclear Instruments and Methods, A523 (2004), 50.

[2] A. Etchegoyen [Pierre Auger Collaboration], AMIGA, Auger Muons and Infill for the Ground Array, 30th ICRC (2007)

[3] CDAS software source code, http://www.auger.org.ar/CDAS/, 2009.

[4] R. G. Gallager, D. C. Van Voorhis, Optimal Source Codes for Geometrically Distributed Integer Alphabets, IEEE Transactions on Information Theory 21, 228 (1975).

[5] D. Salomon, Data Compression: The Complete Reference, Fourth Edition, Springer-Verlag, 2007.