Universidad de la República

Tesis de Maestría

Coding of Multichannel Signals with Irregular Sampling

Autor:
Pablo Cerveñansky

Supervisores:
Álvaro Martín
Gadiel Seroussi

Núcleo de Teoría de la Información
Facultad de Ingeniería

16 de septiembre de 2019

Chapter 1

Datasets

1.1 Introduction

Explain why / how every dataset was transformed into a common format.
Show example csv (describe header, data rows, etc.)

Dataset	**#Files**	**#Types**	**Data Types**
IRKIS	7	1	VWC
SST	3	1	SST
ADCP	3	1	Vel
ElNino	1	7	Lat, Long, Zonal Winds, Merid. Winds, Humidity, Air Temp., SST
Solar	4	3	GHI, DNI, DHI
Hail	1	3	Lat, Long, Size
Tornado	1	2	Lat, Long
Wind	1	3	Lat, Long, Speed

Tabla 1.1: Datasets overview.

1.2 IRKIS

1.3 SST

1.4 ADCP

1.5 ElNino

1.6 Solar

1.7 Hail

1.8 Tornado

1.9 Wind

Chapter 2

Coders

2.1 Introduction

2.2 CoderBase

```
input  : file ∈ Datasets: csv file to be coded
output: CoderBase(file): binary file coded with CoderBase
out = new_binary_file()
out.code_integer(CODER_BASE, 8)
out.code_header(file)
out.code_integer(file.data_rows_count(), 24)
foreach column in file.columns do
    foreach entry in column.entries do
        if entry = NO_DATA then
            value = column.no_data_integer
        else
            value = entry + column.offset
        end
        out.code_integer(value, column.total_bits)
    end
end
out.close_file()
```

Figure 2.1: *CoderBase* pseudocode.

```
input  : file: binary file coded with CoderBase
output: DecoderBase(file): csv file decoded with DecoderBase
out = new_csv_file()
coder_value = file.decode_integer(8)
out.decode_header(file)
data_rows_count = file.decode_integer(24)
if coder_value = CODER_BASE then
    foreach column in out.columns do
        foreach entry in column.entries do
            value = file.decode_integer(column.total_bits)
            if value = column.no_data_integer then
                out.write_string(NO_DATA)
            else
                out.write_string(value − column.offset)
            end
        end
    end
else
    . . . // if file was coded with a different coder
end
out.close_file()
```

Figure 2.2: *DecoderBase* pseudocode.

2.3 CoderPCA

2.4 CoderAPCA

2.5 CoderCA

2.6 CoderPWLH and CoderPWLHInt

2.7 CoderGAMPS and CoderGAMPSLimit

2.8 CoderFR

2.9 CoderSF

Chapter 3

Experimental Results

In this chapter we present the experimental results of our project. The main goal of our experiments was to analize the performance of every one of the coders implemented in Chapter 2. To achieve that, we used them to code the different data types of the datasets introduced in Chapter 1. In Section 3.1 we give an overview of the performed experiments. In Section 3.2 we compare the relative performance of the coders with and without mask. In Section 3.3 we compare the performance of the mask coders among each other, while in Section 3.4 we compare them with the gzip algorithm.

3.1 Experiments Overview

The experiments consisted in coding every one of the data types of each dataset taking different parameter combinations. Specifically, we considered the following three parameters:

- 15 coders:
  - *CoderBase*
  - *CoderPCA-NM* and *CoderPCA-M*
  - *CoderAPCA-NM* and *CoderAPCA-M*
  - *CoderCA-NM* and *CoderCA-M*
  - *CoderPWLH-NM* and *CoderPWLH-M*
  - *CoderPWLHInt-NM* and *CoderPWLHInt-M*
  - *CoderGampsLimit-NM* and *CoderGampsLimit-M*
  - *CoderFR-M*
  - *CoderSF-M*
- 7 window sizes: 4, 8, 16, 32, 64, 128 and 256
- 8 error thresholds:
  - 0 (lossless compression)
  - 1, 3, 5, 10, 15, 20 and 30 (lossy compression).

In what follows, we refer to these sets using the notations C, W and E, respectively. Any valid[1] triplet ($c \in C$, $w \in W$, $e \in E$) defines a specific *coding algorithm (CA)*. The experiments consisted in separately coding each data type with each possible coding algorithm $a \in CA$.

It is important to point out that we are making an abuse of notation with the error threshold parameter, since when taking $e \in E$ the actual threshold we consider is the e% of the standard deviation for the data to be coded.

As explained in Section 2.2, *CoderBase* is a low complexity coder[2] which was implemented to facilitate the comparison of the performance of the remaining coders between each other. With that in mind, we calculated the compression rate obtained when coding each data type with each coding algorithm. Next, we define the compression rate.

Definition 3.1.1. The *compression rate (CR)* of a coding algorithm $a \in CA$ for a certain file f is given by the following equation

$$CR(a, f) = 100 \times \frac{|a(f)|}{|CoderBase(f)|}, \tag{3.1}$$

where $|a(f)|$ and $|CoderBase(f)|$ are the sizes of the resulting files obtained when coding f with a and *CoderBase*, respectively.

The performance of algorithm a improves as $|a(f)|$ decreases. Thus, our main goals are to analyze which coding algorithms minimize (3.1) and how do the different parameters influence the result of this equation.

Before continuing, we think it's important to make some additional clarifications regarding our experiments and the coding algorithms considered:

- *CoderBase* is the single coder that only allows to compress in lossless fashion (i.e. parameter e is always 0).
- *CoderBase* and *CoderSF-M* don't use a window (i.e. parameter w is ignored)
- For *CoderPCA-NM* and *CoderPCA-M* the window size parameter determines a fixed window size, while for the remaing coders (except *CoderBase* and *CoderSF-M*) the window size is variable and the parameter specifies its maximum possible value.
- We didn't implement no-mask versions of *CoderFR-M* and *CoderSF-M*. More details on this matter can be found in Sections 2.8 and 2.9.

[1]Not every parameter combination makes sense (see final paragraph of the current section).
[2]According to our definition it is also a *coding algorithm* (see first bullet point).

3.2 Relative Performance of the Coders

In this section we compare the performance of the coders with (*M*) and without mask (*NM*). We are only interested in comparing two algorithms between each other when their implementations arise from the same original algorithm but one uses a mask and the other one doesn't. Therefore, of the array of coders defined in Section 3.1 we are going to compare:

- *CoderPCA-NM* and *CoderPCA-M*
- *CoderAPCA-NM* and *CoderAPCA-M*
- *CoderCA-NM* and *CoderCA-M*
- *CoderPWLH-NM* and *CoderPWLH-M*
- *CoderPWLHInt-NM* and *CoderPWLHInt-M*
- *CoderGampsLimit-NM* and *CoderGampsLimit-M*.

Definition 3.2.1. We define C_M as the subset of C in which the coders use a mask and CA_M as the set of coding algorithms in which parameter $c \in C_M$. We define C_{NM} and CA_{NM} in an analogue manner for the coders that don't use a mask.

To compare the performance of two coders, $c_M \in C_M$ and $c_{NM} \in C_{NM}$, when coding a certain file, we consider a pair of coding algorithms $a_M \in CA_M$ and $a_{NM} \in CA_{NM}$, with the same w and e but different c parameters, and calculate the relative difference between each other, which we define next.

Definition 3.2.2. The *relative difference (RD)* between a pair of coding algorithms $a_M \in CA_M$ and $a_{NM} \in CA_{NM}$ for a certain file f is given by the following equation

$$RD(a_M, a_{NM}, f) = \begin{cases} 100 \times \frac{|a_{NM}(f)| - |a_M(f)|}{|a_{NM}(f)|}, & \text{if } |a_{NM}(f)| \neq |a_M(f)|, \\ 0, & \text{if } |a_{NM}(f)| = |a_M(f)|, \end{cases} \tag{3.2}$$

where $|a_M(f)|$ and $|a_{NM}(f)|$ are the sizes of the resulting files obtained when coding f with a_M and a_{NM}, respectively.

Algorithm a_M achieves a better compression rate than algorithm a_{NM} when the result of equation (3.2) is positive. As the result increases, the relative performance of a_M improves with respect to a_{NM}.

When comparing the performance of a pair of coders c_M and c_{NM} when coding a file with a given error threshold $e \in E$, we will only consider the window sizes $w_M, w_{NM} \in W$ for which the associated coding algorithms $a_M = (c_M, w_M, e)$ and $a_{NM} = (c_{NM}, w_{NM}, e)$ obtain the minimum compression rates. We refer to w_M and w_{NM} as the optimal window sizes and formally define them next.

Definition 3.2.3. The *optimal window size (OWS)* of a coder $c \in C$ and threshold $e \in E$ for a certain file f is given by the following equation

$$OWS(c, e, f) = w^* \in W \text{ such that } CR((c, w^*, e), f) = \min_{w \in W} \Big\{ CR((c, w, e), f) \Big\}, \tag{3.3}$$

where we take the smallest window in the event more than one value satisfies the equation[3].

[3]This was never the case on our experiments.

Table 3.1 illustrates the summary results obtained when comparing the relative performance of pairs of coders c_{NM} and c_M for each dataset. In the third and fourth columns we display the percentage of the combinations $<c \in C,\ e \in E>$ for which each coding algorithm obtains the best compression rate. The last column shows the range for the relative difference values for said combinations.

Dataset	**Information**	**Best c_{NM}**	**Best c_M**	**RD Range**
IRKIS	Many gaps	-	100%	(0; 36.88]
SST	Many gaps	-	100%	(0; 50.60]
ADCP	Many gaps	-	100%	(0; 17.35]
ElNino	Many gaps	-	100%	(0; 50.52]
Solar	Few gaps	49%	51%	[-0.25; 1.77]
Hail	No gaps	100%	-	[-0.04; 0)
Tornado	No gaps	100%	-	[-0.29; 0)
Wind	No gaps	100%	-	[-0.12; 0)

TABLA 3.1: Relative performance of the c_{NM} and c_M coders. In the last column we highlighted the maximum relative difference values for c_M (blue) and c_{NM} (red).

On datasets with many gaps RD is always positive, and so in every case we achieve better compression rates when using c_M coders. On the other hand, on gapless datasets RD is always negative, which means that c_{NM} coders always perform better. On the dataset with few gaps, approximately on each half of the combinations the best results are obtained with different coders.

We notice that in every case in which c_{NM} performs best the relative difference is close to zero. In the graphs in Figure 3.1 we display the case in which c_{NM} obtains the most significant relative difference. This occurs for the "Longitude" data type of the Tornado dataset, when comparing *CoderAPCA-NM* and *CoderAPCA-M* taking an error threshold equal to 30. In Table 3.1 we can verify that in such case the relative difference is -0.29.

On the other hand, when c_M performs best the relative difference reaches higher absolute values, with a maximum of 50.60 for the "VWC" data type of the SST dataset. In the graphs in Figure 3.2 we can observe that said result is obtained when comparing *CoderPCA-NM* and *CoderPCA-M* taking an error threshold equal to 30.

The experimental results presented in this section suggest that if we were interested in coding a dataset with many gaps, we would benefit by using a c_M algorithm. However, even if the dataset didn't have any gaps, the performance gain obtained by using a c_{NM} algorithm instead would be negligible. Since the c_M algorithm is more robust and performs better in general, in the next sections we will focus on its study.

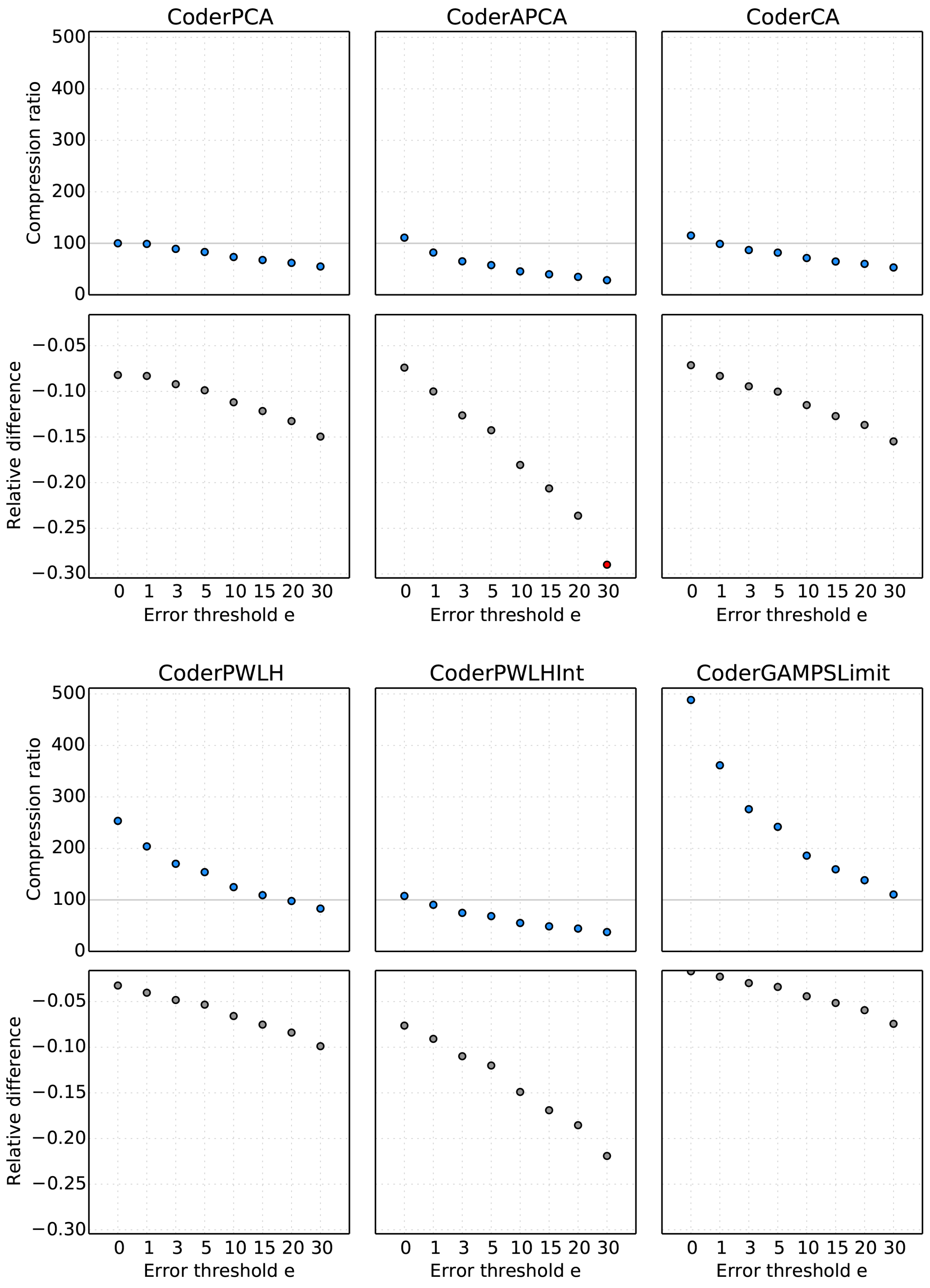

FIGURE 3.1: Compression rate and relative difference for the combinations $<c \in C,\, u \in U>$ for the “Longitude” data type of the Tornado dataset. In the relative difference plot for CoderAPCA we marked with red color the case in which c_{NM} obtains the most significant relative difference (-0.29).

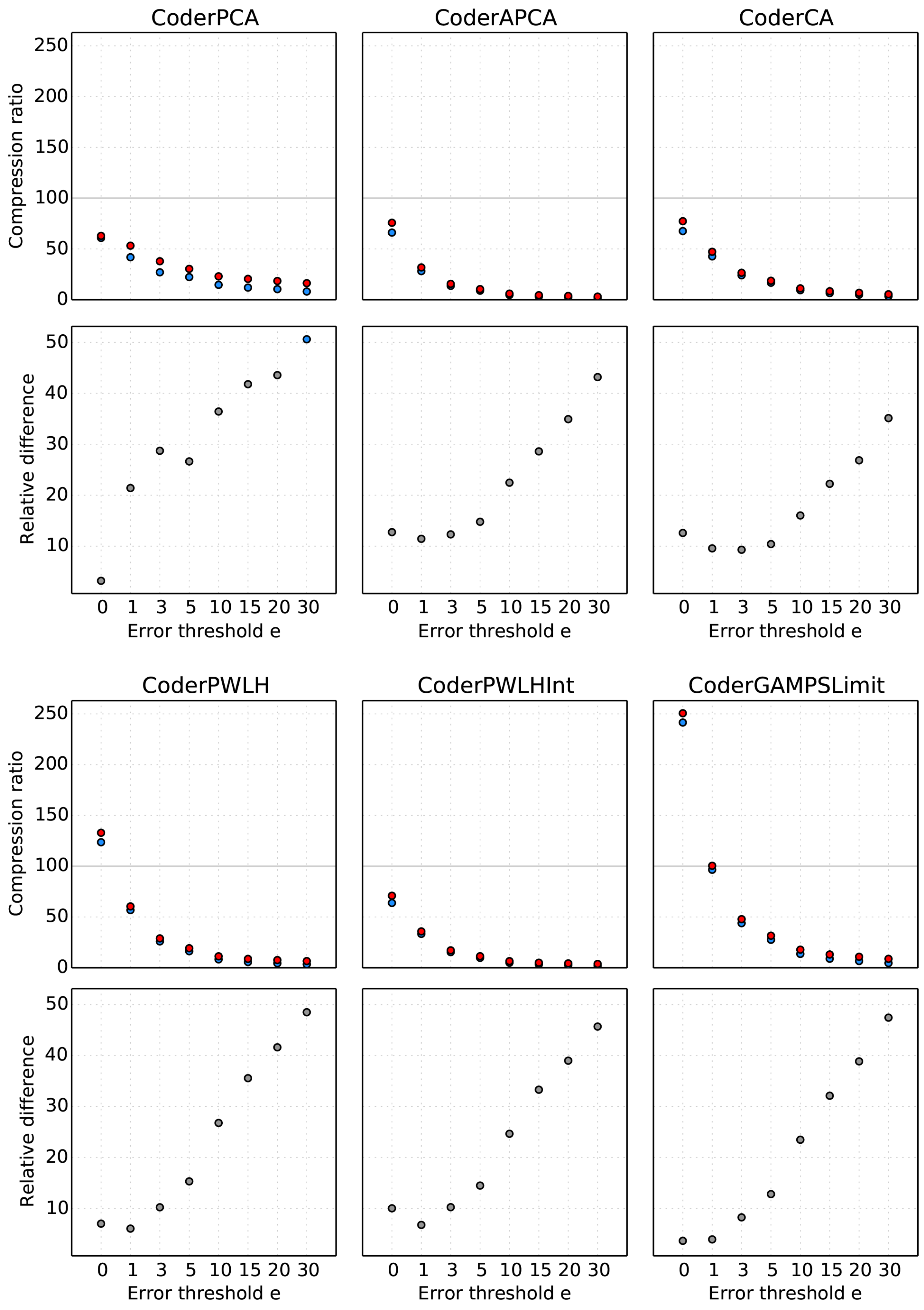

FIGURE 3.2: Compression rate and relative difference for the combinations $<c \in C,\ u \in U>$ for the "VWC" data type of the SST dataset. In the relative difference plot for CoderPCA we marked with blue color the case in which c_M obtains the most significant relative difference (50.60).

3.3 Mask Coders Performance

In this section we analyze the performance of every one of the mask coders implemented in Chapter 2. Once again, the compression rate defined in equation (3.1) will be the metric we use for comparing the coders between each other.

We considered the results obtained when coding the different data types of the datasets introduced in Chapter 1. For example, in Figure 3.3 we can see the graphs obtained for the "VWC" data type of the IRKIS dataset. For each $<c \in C,\ e \in E>$ combination we plot two values: the window size which minimizes the compression rate and said compression rate.

Easily, after observing the plots we noticed that in general the compression rate for coders *CoderPWLH-M*, *CoderGAMPSLimit-M* and *CoderSF-M* was worst than the rest.

Analyze the data and discard these

PONER OTRA GRAFICA DE OTRO TIPO DE DATO

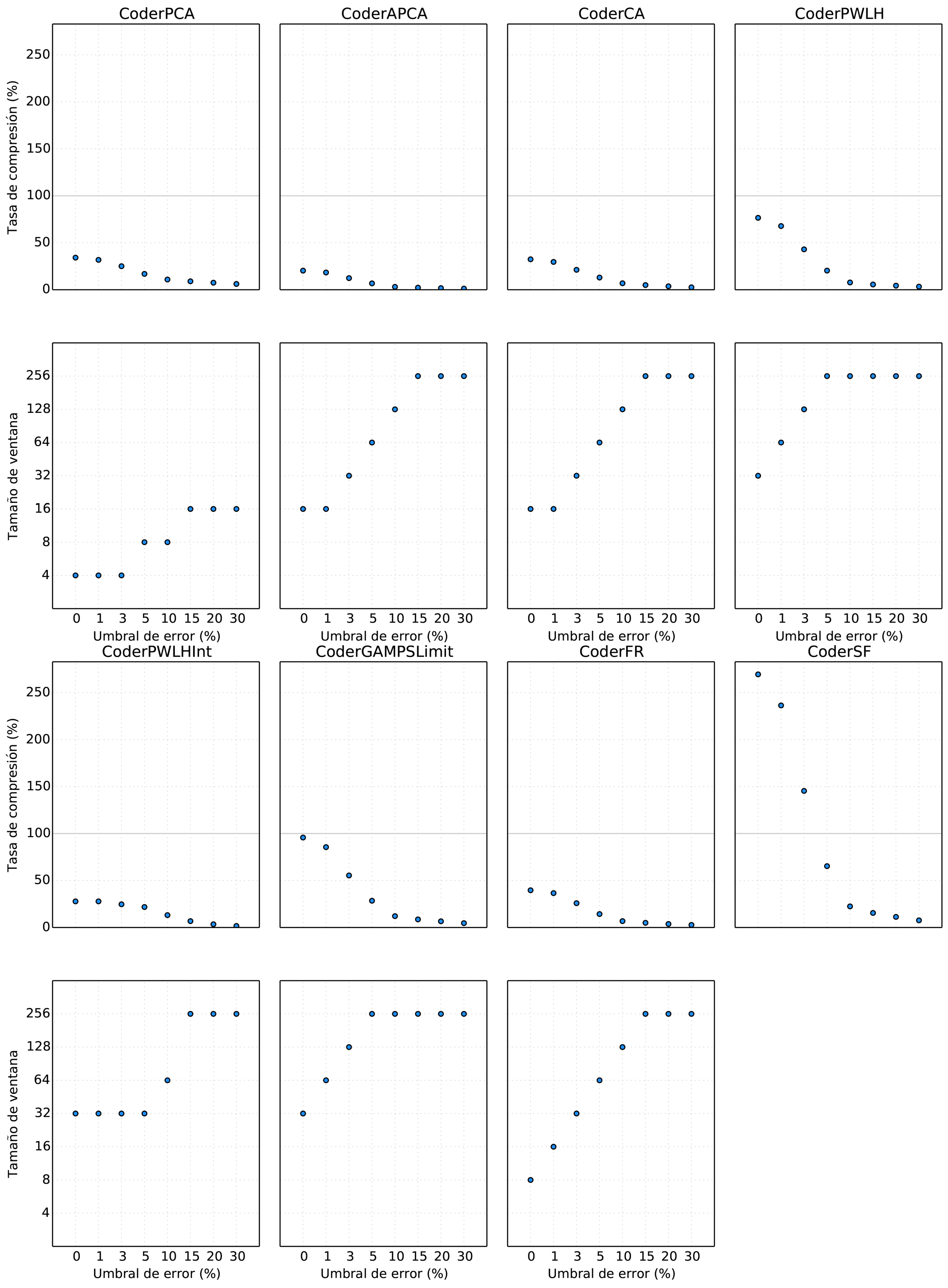

FIGURE 3.3: Compression rate and Window size graphs for the different combinations $<c \in C,\, w \in W, e \in E>$ for the "VWC" data type of the IRKIS dataset.

3.4 Comparison with the gzip Algorithm

HECHO INFORME:

- Elegir nomenclatura para los dos distintos modos de ejecución => CoderPCA-NM (sin máscara) y CoderPCA-M (con máscara).

- Realizar un análisis cuantitativo para saber qué tanto mejor comprime el modo MM=0 en los pocos casos en los que funciona mejor que el modo MM=3. Vimos que esos casos se dan en los datasets con pocos o ningún gap, y la diferencia en las tasas de compresión es mínima. En cambio, cuando hay gaps en los datasets, la diferencia relativa de rendimiento a favor del modo MM=3 es mayor. Escribir un párrafo con dicho análisis, incluyendo alguna gráfica como ejemplo.

- Agregar tabla con resumen de los datasets - ver AVANCES / DUDAS (13)

- Poner las gráficas horizontales, 3 arriba y 3 abajo.

- Mencionar que CoderSlideFilter no tiene en cuenta el parámetro con el tamaño máximo de la ventana.

TODO INFORME:

- Vimos que en los datasets sin gaps, en general para todas las combinaciones <tipo de dato, algoritmo> la diferencia relativa no crece al aumentar el umbral de error. Escribir un párrafo explicando el por qué de este comportamiento.

- Mencionar experimentos ventana local vs ventana global. (ver minuta de la reunión del lunes 10/06/2019).

- Mencionar relación de compromiso entre el umbral y la tasa de compresión: al aumentar el umbral mejor la tasa de compresión (lógico).

- Agregar tabla con resumen de los algoritmos.

- Subir todo el material complementario en un link (después referirlo en el informe)

TODO CÓDIGO:

- Para los experimentos sin máscara no se están considerando los datos para los algoritmos CoderFractalRestampling y CoderSlideFilter.

- Agregar tests para MM=3.

- Al ejecutar los algoritmos GAMPS/GAMPSLimit sobre el dataset de ”El Niño” (546 columnas) tengo problemas de memoria en Ubuntu, pero no en la Mac.

- Universalizar algoritmo

- Modificar GAMPS/GAMPSLimit para que utilice floats (4 bytes) en vez de doubles (8 bytes). De todas maneras, no creo que esto cambie los resultados de manera significativa, ya que aun si la cantidad de bits utilizados al codificar con GAMPS/GAMPSLimit fuera la mitad, en ningún caso superaría la tasa obtenida con el mejor codificador.