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

July 27, 2020

Cambios de la versión anterior (13/7/2020) a esta versión (27/7/2020)

Chapter 1: Datasets

- Agregué una introducción especificando lo que va a ir en cada section.
- En la Section 1.1 agregué una tabla con los datasets. Incluye la columna Dataset Characteristic, que también se muestra en la Tabla 3.1 del Chapter 3. Creo que está bueno que esa misma información se presente de la misma manera cuando se habla de los datasets.

Chapter 2: Algorithms

- Agregué una introducción especificando lo que va a ir en cada section.
- En la Section 2.1 agregué una tabla con los algoritmos. Creo que ayuda a entender el tema de los distintos conjuntos de algoritmos y variantes que se definen en el Chapter 3.
- En la segunda página de la Section 2.1 agregué el pseudocódigo genérico para todos los algoritmos de modelo constante y lineal. El algoritmo GAMPS también considera la correlación entre columnas, así que el pseudocódigo es diferente a los demás (todavía no lo hice, lo pensaba poner en la section de GAMPS).
- En la Section 2.2 agregué los pseudocódigos de codificación y decodificación del algoritmo Base.
- En la Section 2.4 agregué los pseudocódigos de codificación del algoritmo PCA, variantes con y sin máscara. Esta semana termino los de decodificación y sigo con APCA, CA, etc.

Chapter 3: Experimental Results

- Introducción: Hice las correcciones marcadas y la separé en párrafos. Marqué un par de dudas con negrita.
- Al principio de la Section 3.1 agregué varias definiciones en un párrafo. No estoy seguro si debería hacer definiciones formales o si está bien que estén todas juntas en un párrafo.
- En las leyendas de las figuras 3.3 y 3.4 está bien poner LOWS y OWS, o debería poner w^*_{global}y w^*_{local}?
- En todas las figuras después de la Section 3.2 creo que debería poner PCA$_M$, APCA$_M$, etc. en vez de PCA, APCA, etc.
- Agregué la Section 3.5 con conclusiones. Algunas cosas quedaron similares a la introducción, pero intenté de que nunca hubiera dos frases iguales.
- Agregué la Section 3.6 con un punteo de ideas de trabajo futuro. Conclusions and Future Work debería ser un capítulo aparte?

Chapter 1

Datasets

In this chapter we present all of the datasets that were compressed in our experimental work, which is described in Chapter 3. In Section 1.1 we give an overview of the different datasets, describing their characteristics in terms of the amount of gaps, and the number of files and data types that each have. In the remaining sections we present in detail each of the datasets.

1.1 Overview

TODO:

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

Dataset	**Dataset Characterstic**	**#Files**	**#Types**	**Data Types**
IRKIS [1]	Many gaps	7	1	VWC
SST [2]	Many gaps	3	1	SST
ADCP [2]	Many gaps	3	1	Vel
Solar [3]	Many gaps	4	3	GHI, DNI, DHI
ElNino [4]	Many gaps	1	7	Lat, Long, Zonal Winds, Merid. Winds, Humidity, Air Temp., SST
Hail [5]	No gaps	1	3	Lat, Long, Size
Tornado [5]	No gaps	1	2	Lat, Long
Wind [5]	No gaps	1	3	Lat, Long, Speed

Table 1.1: Datasets overview. The second column indicates the characteristic of each dataset, in terms of the amount of gaps. The third column shows the number of files. The fourth and fifth columns show the number of data types and their names, respectively.

1.2 IRKIS

1.3 ADCP

1.4 ElNino

1.5 Solar

1.6 Hail

1.7 Tornado

1.8 Wind

Chapter 2

Algorithms

In this chapter we present all of the coding algorithms implemented in the project. In Section 2.1 we give an overview of the different algorithms, describing their features, parameters and masking variants. In Section 2.2 we present algorithm Base, which is a trivial algorithm that serves as a base ground for the compression performance comparison developed in Chapter 3. In Section 2.3 we detail the implementation of the masking variant, using arithmetic coding and the KT estimator. In the remaining sections we present in detail each of the coding algorithms, including implementation specifics with pseudocode for each of their variants.

2.1 Overview of the Algorithms

TODO:

- Add two main papers as bibliography. [6, 7]
- Add references to C implementation of the coders
- Add references to each coder
- Explain MASK_MODE macro in the C++ code.
- Explain that we iterate in the columns, then in the values VS GAMPS
- Mention that the Timestamps column is always compressed lossless

Algorithm	**Model**	**Lossless**	**Lossy**	**M**	**NM**	**w**
Base	Constant	x			x	
PCA	Constant	x	x	x	x	x
APCA	Constant	x	x	x	x	x
CA	Linear	x	x	x	x	x
PWLH / PWLHInt	Linear	x	x	x	x	x
SF	Linear	x	x	x		
FR	Linear	x	x	x		x
GAMPS / GAMPSLimit	Correlation	x	x	x	x	x

Table 2.1: Coding algorithms overview. For each algorithm, the second column shows its model category, the third and fourth columns indicate whether they support lossless and lossy compression, the fifth and sixth columns indicate if the masking (M) and non-masking (NM) variants are valid, and the last column specifies if the window size parameter (w) is supported.

Figures 2.1 and 2.2 show the coding and decoding pseudocode, respectively, for every one of the Constant Model and Linear Model algorithms presented in Table 2.1. Both pseudocodes have the same length, and lines with the same number perform equivalent actions.

The inputs for the coding routine are a csv data file of any of the datasets presented in Chapter 1, an integer value (*algo_key*) that uniquely describes the selected algorithm, and the error threshold and window size parameters. The output is a binary file, which represents the input file coded by the selected algorithm.

```
input  : in: csv data file to be coded
         algo_key: integer value for the selected algorithm
         e: error threshold parameter (ignored by Base)
         w: window size parameter (ignored by Base and SF)
output: out: binary file encoded with the selected algorithm
 1 out = new_binary_file()
 2 out.code_integer(algo_key, 8)
 3 algo = get_algorithm(algo_key)
 4 if algo.has_window_param? then
 5   | out.code_integer($w - 1, 8$)
 6 end
 7 out.code_header(in.header)
 8 out.code_integer(in.count_data_rows(), 24)
 9 foreach column in in.columns do
10   | algo.code_column(column, out, e, w) // e and w arguments are optional
11 end
12 out.close_file()
```

Figure 2.1: Coding pseudocode for the Constant Model and Linear Model algorithms presented in Chapter 2.

The aforementioned binary file is the only input for the decoding routine. This routine also requires the integer value *algo_key* and the window size parameter, but they are implicit inputs since they are encoded in the binary file (lines 2 and 5). The output is a csv data file with the same headers as the original file (line 7), were the maximum difference allowed between a pair of original and decoded data values depends on the error threshold parameter passed to the coding algorithm.

```
input  : in: coded binary file
output: out: decoded csv data file
 1 out = new_csv_file()
 2 algo_key = in.decode_integer(8)
 3 algo = get_algorithm(algo_key)
 4 if algo.has_window_param? then
 5   | w = in.decode_integer(8) + 1
 6 end
 7 out.decode_header(in)
 8 count_data_rows = in.decode_integer(24)
 9 foreach column in out.columns do
10   | algo.decode_column(column, out, w)
11 end
12 out.close_file()
```

Figure 2.2: Decoding pseudocode for the Constant Model and Linear Model algorithms presented in Chapter 2.

2.2 Base

```
input  : column: column of the csv data file to be coded
         out: binary file encoded with the selected algorithm
foreach entry in column.entries do
    if entry == NO_DATA then
        value = column.no_data_int
    else
        value = entry + column.offset
    end
    out.code_integer(value, column.total_bits)
end
```

Figure 2.3: Algorithm Base coder pseudocode.

```
input  : in: binary file coded with algorithm Base
output: out: decoded csv data file
foreach entry in column.entries do
    value = in.decode_integer(column.total_bits)
    if value == column.no_data_int then
        out.write_string(NO_DATA)
    else
        out.write_string(value − column.offset)
    end
end
```

Figure 2.4: Algorithm Base decoder pseudocode.

Mention: Some of the implementation details are not shown in the pseudocode.

2.3 Mask Modes

TODO:

- Explain the Arithmetic Coder. [8, 9]
- Explain KT estimator

2.4 PCA

```
input : column: column of the csv data file to be coded, out: binary file encoded with the
        selected algorithm, e: error threshold parameter, w: window size parameter
err = column.error_for_threshold_parameter(e)
win = new_window()
foreach entry in column.entries do
    win.push(entry)
    if win.size < w then
        continue
    end
    if |win.max − win.min| ≤ 2 ∗ err then
        out.code_bit(0)
        value = (win.min + win.max)/2
        out.code_integer(value + column.offset, column.total_bits)
    else
        out.code_bit(1)
        foreach win_val in win.values do
            value = win_val + column.offset
            out.code_integer(value, column.total_bits)
        end
    end
    win = new_window()
end
```

FIGURE 2.5: Algorithm PCA_M coder pseudocode.

```
input : column: column of the csv data file to be coded, out: binary file encoded with the
        selected algorithm, e: error threshold parameter, w: window size parameter
err = column.error_for_threshold_parameter(e)
win = new_window()
foreach entry in column.entries do
    win.push(entry)
    if win.size < w then
        continue
    end
    if win.all_entries_are_no_data then
        out.code_bit(0)
        out.code_integer(column.no_data_int, column.total_bits)
    else if win.all_entries_are_integers and $|win.max - win.min| \leq 2 * err$ then
        out.code_bit(0)
        value = (win.min + win.max)/2
        out.code_integer(value + column.offset, column.total_bits)
    else
        out.code_bit(1)
        foreach win_val in win.values do
            if win_val == NO_DATA then
                value = column.no_data_int
            else
                value = win_val + column.offset
            end
            out.code_integer(value, column.total_bits)
        end
    end
    win = new_window()
end
```

FIGURE 2.6: Algorithm PCA$_{NM}$ coder pseudocode.

2.5 APCA

2.6 CA

2.7 PWLH and PWLHInt

2.8 SF

2.9 FR

2.10 GAMPS and GAMPSLimit

Ver los siguientes documentos:

- [08] AVANCES / DUDAS
- [09] AVANCES / DUDAS
- [10] AVANCES / DUDAS
- [11] AVANCES / DUDAS
- [12] AVANCES / DUDAS

Chapter 3

Experimental Results

In this chapter we present our experimental results. The main goal of our experiments is to analyze the performance of each of the coding algorithms presented in Chapter 2, by encoding the various datasets introduced in Chapter 1.

In Section 3.1 we describe our experimental setting and define the evaluated combinations of algorithms, their variants and parameter values, and the figures of merit used for comparison.

In Section 3.2 we compare the compression performance of the masking and non-masking variants for each coding algorithm. The results show that on datasets with few or no gaps the performance of both variants is roughly the same, while on datasets with many gaps the masking variant always performs better, in some cases with a significative difference. These results suggest that the masking variant is more robust and performs better in general.

In Section 3.3 we analyze the extent to which the window size parameter **impacts/affects** the compression performance of the coding algorithms. We compress each dataset file, and compare the results obtained when using the optimal window size (i.e. the one that achieves the best compression) for each file, with the results obtained when using the optimal window size for the whole dataset. The results indicate that the **impact/effect** of using the optimal window size for the whole dataset instead of the optimal window size for each file is rather small.

In Section 3.4 we compare the performance of the different coding algorithms among each other and with the general purpose compression algorithm gzip. Among the tested coding algorithms, for larger error thresholds APCA is the best algorithm for compressing every data type in our experimental data set, while for lower thresholds the recommended algorithms are PCA, APCA and FR, depending on the data type. If we also consider algorithm gzip, **there isn't an algorithm that is better for compressing every data type for any threshold value / for no threshold value there exists an algorithm that is better for compresing every data type**. Depending on the data type, the recommended algorithms are APCA and gzip for larger error thresholds, and PCA, APCA, FR and gzip for lower thresholds.

3.1 Experimental Setting

We denote by A the set of all the coding algorithms presented in Chapter 2. For an algorithm $a \in A$, we denote by a_v its variant v, where v can be M (masking) or NM (non-masking). There exist some $a \in A$ for which either a_M or a_{NM} is invalid (recall this information from Table 2.1). We denote by V the set of variants composed of every valid variant a_v for every algorithm $a \in A$. Also, we denote by A_M the subset of algorithms from A composed of every algorithm for which both variants, a_M and a_{NM}, are valid.

We evaluate the compression performance of every algorithm $a \in A$ on the datasets described in Chapter 1. For each algorithm we test every valid variant a_v. We also test several combinations of algorithm parameters. Specifically, for the algorithms that admit a window size parameter w (every algorithm except *Base* and *SF*), we test all the values of w in the set $W = \{4, 8, 16, 32, 64, 128, 256\}$. For the encoders that admit a lossy compression mode with a threshold parameter e (every encoder except *Base*), we test all the values of e in the set $E = \{1, 3, 5, 10, 15, 20, 30\}$, where each threshold is expressed as a percentage fraction of the standard deviation of the data being encoded. For example, for certain data with a standard deviation of 20, taking $e = 10$ implies that the lossy compression allows for a maximal per-sample distortion of 2 sampling units.

Definition 3.1.1. We refer to a specific combination of a coding algorithm variant and its parameter values as a *coding algorithm instance (CAI)*. We define CI as the set of all the CAIs obtained by combining each of the variants $a_v \in V$ with the parameter values (from W and E) that are suitable for algorithm a. We denote by $c_{<a_v,w,e>}$ the CAI obtained by setting a window size parameter equal to w and a threshold parameter equal to e on algorithm variant a_v.

We assess the compression performance of a CAI mainly through the compression ratio, which we define next. For this definition, we regard *Base* as a trivial CAI that serves as a base ground for compression performance comparison (recall the definition of algorithm *Base* from Section 2.2).

Definition 3.1.2. Let f be a file and z a data type of a certain dataset. We define f_z as the subset of data of type z from file f. For example, for the dataset Hail, the data type z may be Latitude, Longitude, or Size.

Definition 3.1.3. Let f be a file and z a data type of a certain dataset. Let $c \in CI$ be a CAI. We define $|c(z, f)|$ as the size in bits of the resulting bit stream obtained by coding f_z with c.

Definition 3.1.4. The *compression ratio (CR)* of a CAI $c \in CI$ for the data type z of a certain file f is the fraction of $|c(z, f)|$ with respect to $|\text{Base}(z, f)|$, i.e.,

$$CR(c, z, f) = \frac{|c(z, f)|}{|\text{Base}(z, f)|}. \tag{3.1}$$

Notice that smaller values of CR correspond to better performance. Our main goals are to analyze which CAIs yield the smallest values in (3.1) for the different data types, and to study how the CR depends on the different algorithms, their variants and the parameter values.

To compare the compression performance between a pair of CAIs we calculate the relative difference, which we define next.

Definition 3.1.5. The *relative difference (RD)* between a pair of CAIs $c_1, c_2 \in CI$ for the data type z of a certain file f is given by

$$RD(c_1, c_2, z, f) = 100 \times \frac{|c_2(z,f)| - |c_1(z,f)|}{|c_2(z,f)|}. \tag{3.2}$$

Notice that c_1 has a better performance than c_2 if (3.2) is positive.

In some of our experiments we consider the performance of algorithms on complete datasets, rather than individual files. With this in mind, we extend the definitions 3.1.3–3.1.5 to datasets, as follows.

Definition 3.1.6. Let z be a data type of a certain dataset d. We define $F(d, z)$ as the set of files f from dataset d for which f_z is not empty.

Definition 3.1.7. Let z be a data type of a certain dataset d. Let $c \in CI$ be a CAI. We define $|c(z, d)|$ as

$$|c(z,d)| = \sum_{f \in F(d,z)} |c(z,f)|. \tag{3.3}$$

Definition 3.1.8. The *compression ratio (CR)* of a CAI $c \in CI$ for the data type z of a certain dataset d is given by

$$CR(c, z, d) = \frac{|c(z,d)|}{|\mathrm{Base}(z,d)|}. \tag{3.4}$$

Definition 3.1.9. The *relative difference (RD)* between a pair of CAIs $c_1, c_2 \in CI$ for the data type z of a certain dataset d is given by

$$RD(c_1, c_2, z, d) = 100 \times \frac{|c_2(z,d)| - |c_1(z,d)|}{|c_2(z,d)|}. \tag{3.5}$$

3.2 Comparison of Masking and Non-Masking Variants

In this section, we compare the compression performance of the masking and non-masking variants of every coding algorithm in A_M. Specifically, we compare:

- PCA_M against PCA_{NM}
- APCA_M against APCA_{NM}
- CA_M against CA_{NM}
- PWLH_M against PWLH_{NM}
- PWLHInt_M against PWLHInt_{NM}
- GAMPSLimit_M against GAMPSLimit_{NM}

For each algorithm $a \in A_M$ and each threshold parameter, we compare the performance of a_M and a_{NM}. For the purpose of this comparison, we choose the most favorable window size for each variant a_v, in the sense of the following definition.

Definition 3.2.1. The *optimal window size (OWS)* of a coding algorithm variant $a_v \in V$, and a threshold parameter $e \in E$, for the data type z of a certain dataset d, is given by

$$OWS(a_v, e, z, d) = \underset{w \,\in\, W}{\arg\min} \left\{ CR(c_{<a_v, w, e>}, z, d) \right\}, \tag{3.6}$$

where we break ties in favor of the smallest window size.

For each data type z of each dataset d, and each coding algorithm $a \in A_M$ and threshold parameter $e \in E$, we calculate the RD between $c_{<a_M, w^*_M, e>}$ and $c_{<a_{NM}, w^*_{NM}, e>}$, as defined in (3.5), where $w^*_M = \text{OWS}(a_M, e, z, d)$ and $w^*_{NM} = \text{OWS}(a_{NM}, e, z, d)$.

As an example, in figures 3.1 and 3.2 we show the CR and the RD, as a function of the threshold parameter, obtained for two data types of two different datasets. Figure 3.1 shows the results for the data type "SST" of the dataset SST, and Figure 3.2 shows the results for the data type "Longitude" of the dataset Tornado. In Figure 3.1 we observe a large RD favoring the masking variant for all tested algorithms. On the other hand, in Figure 3.2 we observe that the non-masking variant outperforms the masking variant for all algorithms. We notice, however, that the RD is very small in the latter case.

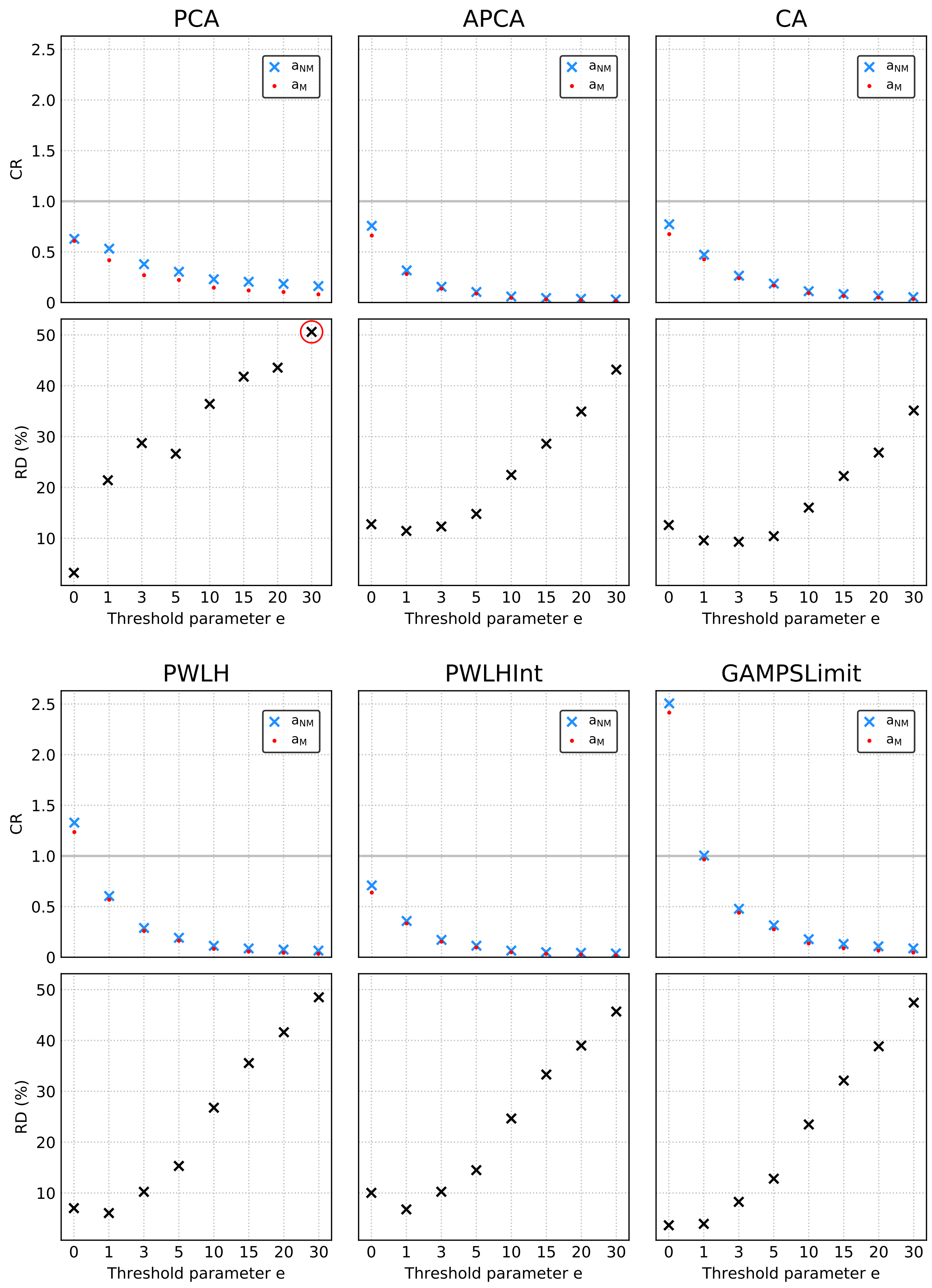

FIGURE 3.1: CR and RD plots for every pair of algorithm variants $a_M, a_{NM} \in A_M$, for the data type "SST" of the dataset SST. In the RD plot for algorithm PCA we highlight with a red circle the marker for the maximum value (50.60%) obtained for all the tested CAIs.

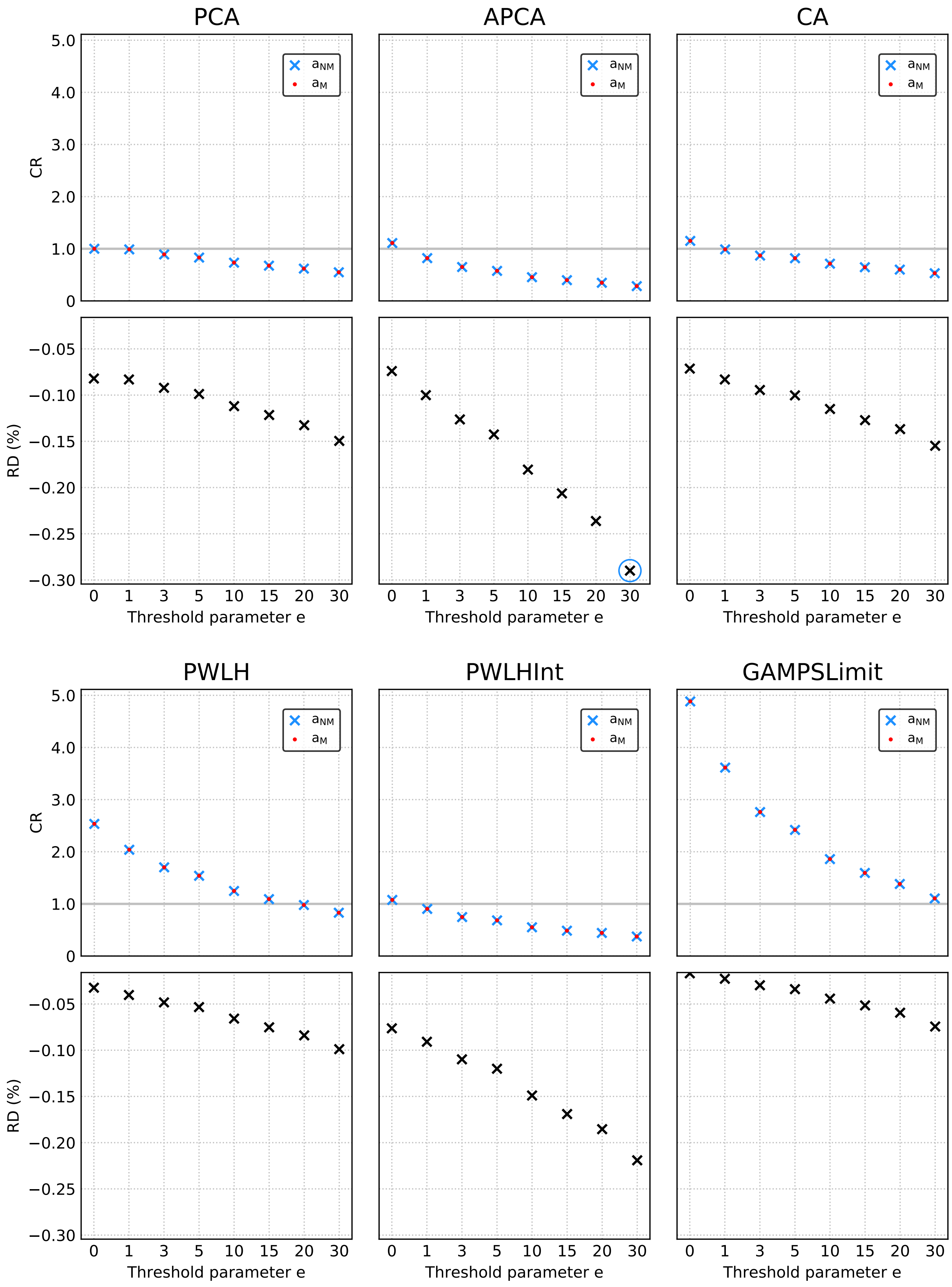

FIGURE 3.2: CR and RD plots for every pair of algorithm variants $a_M, a_{NM} \in A_M$, for the data type "Longitude" of the dataset Tornado. In the RD plot for algorithm APCA we highlight with a blue circle the marker for the minimum value (-0.29%) obtained for all the tested CAIs.

We analyze the experimental results to compare the performance of the masking and non-masking variants of each algorithm. For each data type, we iterate through each algorithm $a \in A_M$, and each threshold parameter $e \in E$, and we calculate the RD between the CAIs $c_{<a_M, w^*_M, e>}$ and $c_{<a_{NM}, w^*_{NM}, e>}$, obtained by setting the OWS for the masking variant a_M and the non-masking variant a_{NM}, respectively. Since we consider 8 threshold parameters and there are 6 algorithms in A_M, for each data type we compare a total of 48 pairs of CAIs. Table 3.1 summarizes the results of these comparisons, aggregated by dataset. The number of pairs of CAIs evaluated for each dataset depends on the number of different data types it contains.

Dataset	**Dataset Characterstic**	**Cases where a_M outperforms a_{NM} (%)**	**RD (%) Range**
IRKIS	Many gaps	48/48 (100%)	(0; 36.88]
SST	Many gaps	48/48 (100%)	(0; 50.60]
ADCP	Many gaps	48/48 (100%)	(0; 17.35]
ElNino	Many gaps	336/336 (100%)	(0; 50.52]
Solar	Few gaps	73/144 (50.7%)	[-0.25; 1.77]
Hail	No gaps	0/144 (0%)	[-0.04; 0)
Tornado	No gaps	0/96 (0%)	[-0.29; 0)
Wind	No gaps	0/144 (0%)	[-0.12; 0)

Table 3.1: Range of values for the RD between the masking and non-masking variants of each algorithm (last column); we highlight the maximum (red) and minimum (blue) values taken by the RD. The results are aggregated by dataset. The second column indicates the characteristic of each dataset, in terms of the amount of gaps. The third column shows the number of cases in which the masking variant outperforms the non-masking variant of a coding algorithm, and its percentage among the total pairs of CAIs compared for a dataset.

Consider, for example, the results for the dataset Wind, in the last row. The second column shows that there are no gaps in any of the data types of the dataset (recall the dataset description from Table 1.1). Since the dataset has three data types, we compare a total of $3 \times 48 = 144$ pairs of CAIs. The third column reveals that in none of these comparisons the masking variant a_M outperforms the non-masking variant a_{NM}, i.e. the RD is always negative. The last column shows the range for the values attained by the RD for those tested CAIs.

Observing the last column of Table 3.1, we notice that in every case in which the non-masking variant performs best, the RD is close to zero. The minimum value it takes is -0.29%, which is obtained for the data type "Longitude" of the dataset Tornado, with algorithm APCA, and error parameter $e = 30$. In Figure 3.2 we highlight the marker associated to this minimum with a blue circle. On the other hand, we also notice that for the datasets in which the masking variant performs best, the RD reaches high absolute values. The maximum (50.60%) is obtained for the data type "VWC" of the dataset SST, with algorithm PCA, and error parameter $e = 30$, which is highlighted in Figure 3.1 with a red circle.

The experimental results presented in this section suggest that if we were interested in compressing a dataset with many gaps, we would benefit from using the masking variant of an algorithm, a_M. However, even if the dataset didn't have any gaps, the performance would not be significantly worse than that obtained by using the non-masking variant of the algorithm, a_{NM}. Therefore, since masking variants are, in general, more robust in this sense, in the sequel we focus on the set of variants V^* that we define next.

Definition 3.2.2. We denote by V^* the set of all the masking algorithm variants a_M for $a \in A$.

Notice that V^* includes a single variant for each algorithm. Therefore, in what follows we sometimes refer to the elements of V^* simply as algorithms.

3.3 Window Size Parameter

In this section, we analyze the extent to which the window size parameter impacts on the performance of the coding algorithms. For these experiments we consider the set of algorithm variants V^*_W, which is obtained from V^* by discarding algorithm SF, which doesn't have a window size parameter (recall this information from Table 2.1). Also, we only consider the four datasets that consist of multiple files, i.e. IRKIS, SST, ADCP and Solar (recall this information from Table 1.1). For each file, we compare the compression performance when using the OWS for the dataset, as defined in (3.6), and the LOWS for the file, defined next.

Definition 3.3.1. The *local optimal window size (LOWS)* of a coding algorithm variant $a_v \in V^*_W$, and a threshold parameter $e \in E$, for the data type z of a certain file f is given by

$$LOWS(a_v, e, z, f) = \underset{w\ \in\ W}{\arg\min} \left\{ CR(c_{<a_v,w,e>}, z, f) \right\}, \tag{3.7}$$

where we break ties in favor of the smallest window size.

For each data type z of each dataset d, and each file $f \in F(d, z)$, coding algorithm variant $a_v \in V^*_W$, and threshold parameter $e \in E$, we calculate the RD between $c_{<a_v,w^*_{global},e>}$ and $c_{<a_v,w^*_{local},e>}$, as defined in (3.2), where $w^*_{global} = \text{OWS}(a_v, e, z, d)$ and $w^*_{local} = \text{LOWS}(a_v, e, z, f)$. In what follows, we denote the OWS and the LOWS as w^*_{global} and w^*_{local}, respectively.

As an example, in figures 3.3 and 3.4 we show w^*_{global}, w^*_{local}, and the RD between $c_{<a_v,w^*_{global},e>}$ and $c_{<a_v,w^*_{local},e>}$, as a function of the threshold parameter e, obtained for the data type z = "VWC", for two different files of the dataset d = IRKIS. Figure 3.3 shows the results for the file f = "vwc_1202.dat.csv", and Figure 3.4 shows the results for f = "vwc_1203.dat.csv". Observe that the values of w^*_{global} are the same for both figures, which is expected, since both are obtained from the same data type of the same dataset.

In Figure 3.3 we notice, for instance, that for algorithm APCA the OWS and LOWS values match for every threshold parameter e, except 3 and 10. The OWS is larger than the LOWS when $e = 3$, but it is smaller when $e = 10$. In these two cases, the RD values are 1.52% and 1.76%, respectively. In Figure 3.4 we observe that in every case the OWS is larger than or equal to the LOWS . We highlight the marker for the maximum RD value (10.68%) obtained for all the tested CAIs, and we further comment on this point in the remaining of the section. Notice that in both figures the RD is non-negative in every plot, which makes sense, since the CR obtained with the OWS can never be lower than the CR obtained with the LOWS .

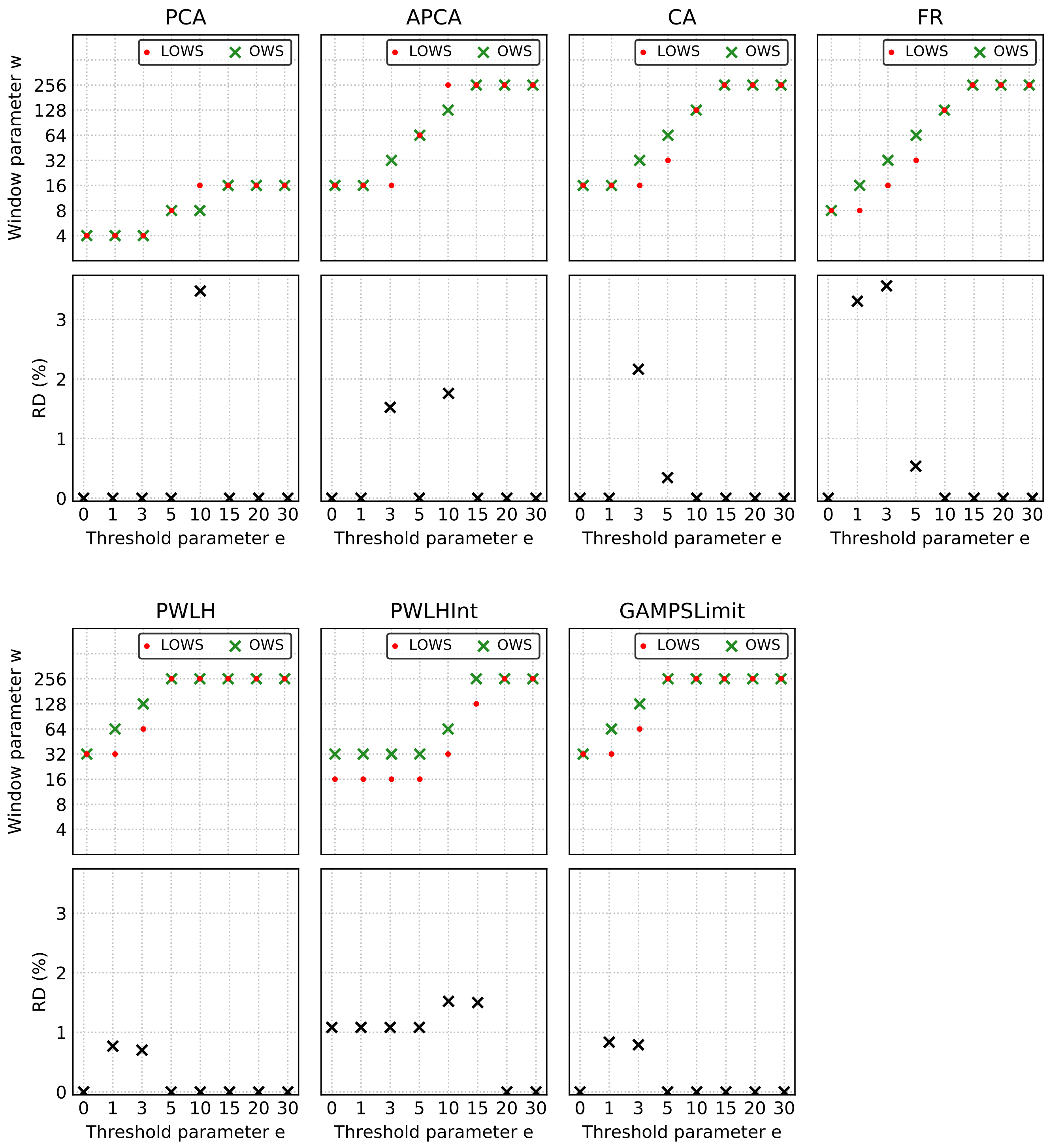

FIGURE 3.3: Plots of w^*_{global}, w^*_{local}, and the RD between $c_{<a_v,w^*_{global},e>}$ and $c_{<a_v,w^*_{local},e>}$, as a function of the threshold parameter e, obtained for the data type "VWC" of the file "vwc_1202.dat.csv" of the dataset IRKIS.

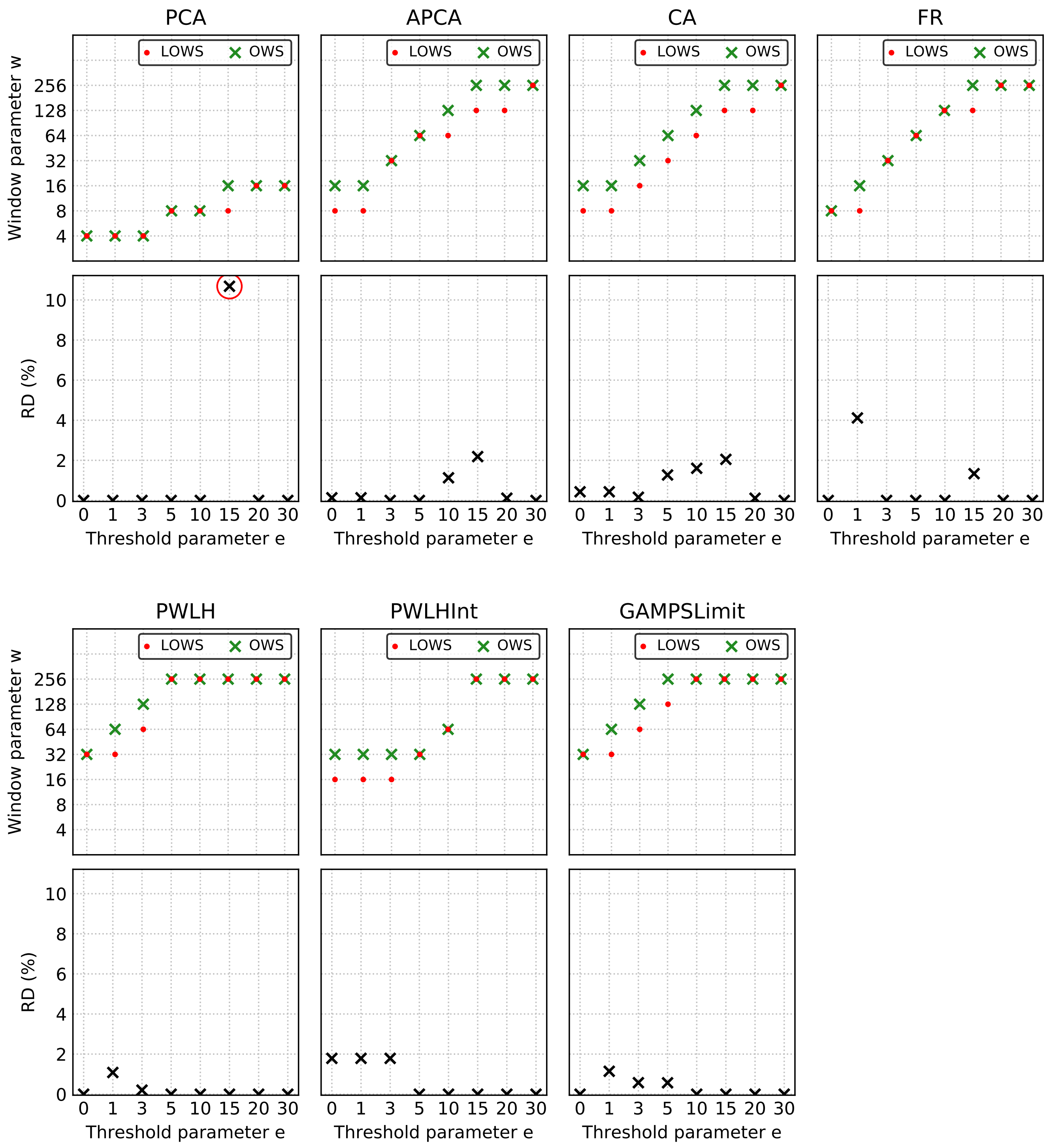

FIGURE 3.4: Plots of w^*_{global}, w^*_{local}, and the RD between $c_{<a_v, w^*_{global}, e>}$ and $c_{<a_v, w^*_{local}, e>}$, as a function of the threshold parameter e, obtained for the data type "VWC" of the file "vwc_1203.dat.csv" of the dataset IRKIS. In the RD plot for algorithm PCA we highlight with a red circle the marker for the maximum value (10.68%) obtained for all the tested CAIs.

We analyze the experimental results to evaluate the impact of using the OWS instead of the LOWS on the compression performance of the tested coding algorithms. For each algorithm, we iterate through each threshold parameter, and each data type of each file, and we calculate the RD between the CAI with the OWS and the CAI with the LOWS . Since we consider 8 threshold parameters and there are 13 files with a single data type and 4 files with 3 different data types each, for each algorithm we compare a total of $8 \times (13 + 4 \times 3) = 200$ pairs of CAIs. Table 3.2 summarizes the results of these comparisons, aggregated by algorithm and the range to which the RD belongs.

	RD (%) Range				
Algorithm	**0**	**(0,1]**	**(1,2]**	**(2,5]**	**(5,11]**
PCA	186 (93%)	4 (2%)	3 (1.5%)	2 (1%)	5 (2.5%)
APCA	174 (87%)	13 (6.5%)	7 (3.5%)	6 (3%)	0
CA	172 (86%)	16 (8%)	6 (3%)	6 (3%)	0
FR	171 (85.5%)	14 (7%)	8 (4%)	7 (3.5%)	0
PWLH	184 (92%)	13 (6.5%)	3 (1.5%)	0	0
PWLHInt	173 (86.5%)	9 (4.5%)	13 (6.5%)	4 (2%)	1 (0.5%)
GAMPSLimit	182 (91%)	16 (8%)	2 (1%)	0	0
Total	1,242 (88.7%)	85 (6.1%)	42 (3%)	25 (1.8%)	6 (0.4%)

Table 3.2: RD between the OWS and LOWS variants of each CAI.
The results are aggregated by algorithm and the range to which the RD belongs.

For example, consider the results for algorithm CA, in the third row. The first column indicates that the RD is equal to 0 for exactly 172 (86%) of the 200 evaluated pairs of CAIs for that algorithm. The second column reveals that for 16 pairs of CAIs (8%), the RD takes values greater than 0 and less than or equal to 1%. The remaining three columns cover other ranges of RD values. Notice that for every row (except the last one), the values add up to a total of 200, since we compare exactly 200 pairs of CAIs for each algorithm.

The last row of Table 3.2 is obtained by adding the values of the previous rows, which combines the results for all algorithms. We notice that in 88.7% of the total number of evaluated pairs of CAIs, the RD is equal to 0. In these cases, in fact, the OWS and the LOWS coincide. In 97.8% of the cases, the RD is less than or equal to 2%. This means that, for the vast majority of CAI pairs, either the OWS and the LOWS match or they yield roughly the same compression performance. This result suggests that we could fix the window size parameter in advance, for example by optimizing over a training set, without compromising the performance of the coding algorithm. This is relevant, since calculating the LOWS for a file is, in general, computationally expensive.

We notice that there are only 6 cases (0.4%) in which the RD falls in the range (5, 11], most of which (5 cases) involve the algorithm PCA. The maximum value taken by RD (10.68%) is obtained for the data type "VWC" of the file "vwc_1203.dat.csv" of the dataset SST, with algorithm PCA, and error parameter $e = 15$. In Figure 3.4 we highlight this maximum value with a red circle. In this case, the OWS is 16 and the LOWS is 8. According to these results, the performance of algorithm PCA seems to be more sensible to the window size parameter than the rest of the algorithms. Except for these few cases, we observe that, in general, the impact of using the OWS instead of the LOWS on the compression performance of coding algorithms is rather small. Therefore, in the following section, in which we compare the algorithms performance, we always use the OWS.

3.4 Algorithms Performance

In this section, we compare the compression performance of the coding algorithms presented in Chapter 2, by encoding the various datasets introduced in Chapter 1. We begin by comparing the algorithms among each other and later we compare them with gzip, a popular lossless compression algorithm. We analyze the performance of the algorithms on complete datasets (not individual files), so we always apply definitions 3.1.6–3.1.9. Following the results obtained in sections 3.2 and 3.3, we only consider the masking variants of the evaluated algorithms (i.e. set V^*), and we always set the window size parameter to the OWS (recall Definition 3.2.1).

For each data type z of each dataset d, and each coding algorithm variant $a_v \in V^*$ and threshold parameter $e \in E$, we calculate the CR of $c_{<a_v, w^*_{global}, e>}$, as defined in (3.4), where $w^*_{global} = \text{OWS}(a_v, e, z, d)$. The following definition is useful for analyzing which CAI obtains the best compression result for a specific data type.

Definition 3.4.1. Let z be a data type of a certain dataset d, and let $e \in E$ be a threshold parameter. We denote by $c^b(z, d, e)$ the *best CAI* for z, d, e, and define it as the CAI that minimizes the CR among all the CAIs in CI, i.e.,

$$c^b(z, d, e) = \underset{c\ \in CI}{\arg\min} \left\{ CR(c_{<a_v, w^*_{global}, e>}, z, d) \right\}. \tag{3.8}$$

When $c^b(z, d, e) = c_{<a^b_v, w^{b*}_{global}, e>}$, we refer to a^b and w^{b*}_{global} as the *best coding algorithm* and the *best window size* for z, d, e, respectively.

Our experiments include a total of 21 data types, in 8 datasets. As an example, in Figure 3.5 we show the CR and the window size parameter w^*_{global}, as a function of the threshold parameter, obtained for each algorithm, for the data type “SST” of the dataset ElNino. For each threshold parameter $e \in E$, we use blue circles to highlight the markers for the minimum CR value and the best window parameter (in the respective plots corresponding to the best algorithm). For instance, for $e = 0$, the best CAI achieves a CR equal to 0.33 using algorithm PCA with a window size of 256. So in this case, algorithm PCA is the best coding algorithm, and 256 is the best window size. For the remaining seven values for the threshold parameter, the blue circles indicate that in every case the best algorithm is APCA, and the best window size ranges from 4 up to 32.

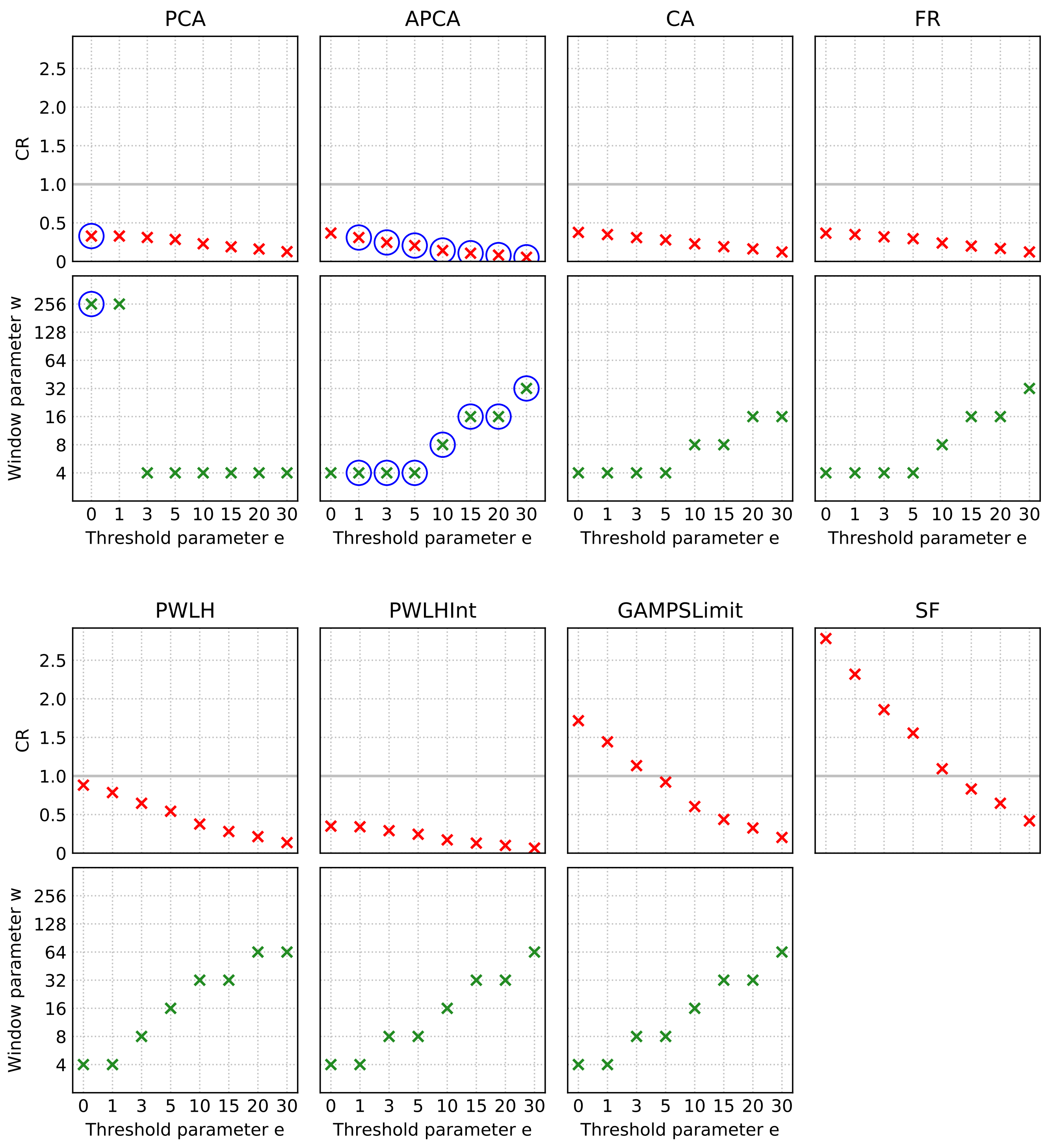

FIGURE 3.5: CR and window size parameter plots for every algorithm, for the data type "SST" of the dataset ElNino. For each threshold parameter $e \in E$, we use blue circles to highlight the markers for the minimum CR value and the best window size parameter (in the respective plots corresponding to the best algorithm)

Table 3.3 summarizes the compression performance results obtained by the evaluated coding algorithms, for each data type of each dataset. Each row contains information relative to certain data type. For example, the 13th row shows summarized results for the data type "SST" of the dataset ElNino, which are presented in more detail in Figure 3.5. For each threshold, the first column shows the CR obtained by the best CAI, the second column shows the base-2 logarithm of its window size parameter, and the cell color identifies the best algorithm.

🟦 PCA	🟩 APCA	🟨 FR

		e = 0		e = 1		e = 3		e = 5		e = 10		e = 15		e = 20		e = 30	
Dataset	Data Type	CR	w	CR	w	CR	w	CR	w	CR	w	CR	w	CR	w	CR	w
IRKIS	VWC	🟩 0.20	4	🟩 0.18	4	🟩 0.12	5	🟩 0.07	6	🟩 0.03	7	🟩 0.02	8	🟩 0.02	8	🟩 0.01	8
SST	SST	🟦 0.61	8	🟩 0.28	3	🟩 0.14	5	🟩 0.09	6	🟩 0.05	7	🟩 0.03	8	🟩 0.02	8	🟩 0.02	8
ADCP	Vel	🟦 0.68	8	🟦 0.68	8	🟩 0.67	2	🟩 0.61	2	🟩 0.48	2	🟩 0.41	2	🟩 0.35	3	🟩 0.26	3
Solar	GHI	🟦 0.78	2	🟩 0.76	3	🟩 0.71	4	🟩 0.67	4	🟩 0.59	4	🟩 0.52	4	🟩 0.47	4	🟩 0.38	4
	DNI	🟦 0.76	2	🟩 0.72	4	🟩 0.66	4	🟩 0.61	4	🟩 0.54	4	🟩 0.49	4	🟩 0.43	4	🟩 0.36	4
	DHI	🟦 0.78	2	🟩 0.77	2	🟩 0.72	4	🟩 0.68	4	🟩 0.60	4	🟩 0.54	4	🟩 0.48	4	🟩 0.39	4
ElNino	Lat	🟩 0.16	4	🟩 0.16	4	🟩 0.16	4	🟩 0.15	4	🟩 0.12	4	🟩 0.10	5	🟩 0.09	5	🟩 0.06	6
	Long	🟩 0.17	3	🟩 0.17	4	🟩 0.13	4	🟩 0.12	5	🟩 0.09	6	🟩 0.07	6	🟩 0.05	7	🟩 0.02	8
	Z. Wind	🟦 0.31	8	🟦 0.31	8	🟦 0.31	8	🟦 0.31	8	🟩 0.27	2	🟩 0.24	2	🟩 0.21	2	🟩 0.16	3
	M. Wind	🟦 0.31	8	🟦 0.31	8	🟦 0.31	8	🟦 0.31	8	🟩 0.29	2	🟩 0.26	2	🟩 0.23	2	🟩 0.19	2
	Humidity	🟦 0.23	8	🟦 0.23	8	🟦 0.23	8	🟦 0.23	8	🟩 0.21	2	🟩 0.18	2	🟩 0.16	2	🟩 0.13	2
	AirTemp	🟦 0.33	8	🟦 0.33	8	🟩 0.30	2	🟩 0.27	2	🟩 0.22	2	🟩 0.19	3	🟩 0.17	3	🟩 0.13	4
	SST	🟦 0.33	8	🟩 0.31	2	🟩 0.25	2	🟩 0.21	2	🟩 0.14	3	🟩 0.11	4	🟩 0.08	4	🟩 0.05	5
Hail	Lat	🟦 1.00	8	🟦 1.00	8	🟩 0.90	2	🟩 0.83	2	🟩 0.71	2	🟩 0.65	3	🟩 0.57	3	🟩 0.47	3
	Long	🟦 1.00	8	🟦 1.00	8	🟩 0.86	2	🟩 0.78	2	🟩 0.65	2	🟩 0.55	3	🟩 0.49	3	🟩 0.39	4
	Size	🟩 0.81	2	🟩 0.81	2	🟩 0.81	2	🟩 0.81	2	🟩 0.81	2	🟩 0.81	2	🟩 0.81	2	🟩 0.64	3
Tornado	Lat	🟦 1.00	8	🟩 0.85	2	🟩 0.71	2	🟩 0.65	2	🟩 0.54	3	🟩 0.47	3	🟩 0.42	4	🟩 0.33	4
	Long	🟦 1.00	8	🟩 0.82	2	🟩 0.65	2	🟩 0.58	3	🟩 0.46	3	🟩 0.40	4	🟩 0.35	4	🟩 0.28	4
Wind	Lat	🟦 1.00	8	🟦 1.00	8	🟩 0.89	2	🟩 0.81	2	🟩 0.70	2	🟩 0.62	3	🟩 0.56	3	🟩 0.47	3
	Long	🟦 1.00	8	🟩 0.95	2	🟩 0.80	2	🟩 0.73	2	🟩 0.62	3	🟩 0.54	3	🟩 0.49	3	🟩 0.40	4
	Speed	🟨 0.65	4	🟩 0.44	3	🟨 0.26	6	🟨 0.17	7	🟩 0.16	5	🟩 0.12	6	🟩 0.10	6	🟩 0.08	6

TABLE 3.3: Compression performance of the best evaluated coding algorithm, for various error thresholds on each data type of each dataset. Each row contains information relative to certain data type. For each threshold, the first column shows the minimum CR, and the second column shows the base-2 logarithm of the window size parameter for the best algorithm (the one that achieves the minimum CR), which is identified by a certain cell color described in the legend above the table.

We observe that there are only three algorithms (PCA, APCA, and FR) which are used by the best CAI for at least one of the 168 possible data type and threshold parameter combinations. Algorithm APCA is used in exactly 134 combinations (80%), including every case in which $e \geq 10$, and most of the cases in which $e \in [1, 3, 5]$. PCA is used in 31 combinations (18%), including most of the lossless cases, while FR is the best algorithm in only 3 combinations (2%), all of them for data type "Speed" of the dataset Wind.

Since there is not a single algorithm that obtains the best compression performance for every data type, it is useful to analyze how much is the RD between the best algorithm and the rest, for every experimental combination. With that in mind, next we define a pair of metrics.

Definition 3.4.2. The *maximum RD* (*maxRD*) of a coding algorithm $a \in A$ for certain threshold parameter $e \in E$ is given by

$$maxRD(a, e) = \max_{z,d} \left\{ RD(c^b(z, d, e), c_{<a_v, w^*_{global}, e>}) \right\}, \tag{3.9}$$

where the maximum is taken over all the combinations of data type z and dataset d, and we recall that $c^b(z, d, e)$ is the best CAI for z, d, e.

The maxRD metric is useful for assessing the compression performance of a coding algorithm a on the set of data types as a whole. Notice that maxRD is always non-negative. A satisfactory result (i.e. close to zero) can only be obtained when a achieves a good compression performance *for every data type*. In other words, bad compression performance *on a single data type* yields a poor result for the maxRD metric altogether. When maxRD is equal to zero, a achieves the best compression performance for every combination. Analyzing the results in Table 3.3, we observe that maxRD(APCA, e) = 0 for every $e \geq 10$. Since the best algorithm is unique for every combination (i.e. exactly one algorithm obtains the minimum CR in every case), it is also true that, when $a \neq$ APCA, maxRD(a, e) > 0 for every $e \geq 10$.

Definition 3.4.3. The *minmax RD* (*minmaxRD*) for certain threshold parameter $e \in E$ is given by

$$minmaxRD(e) = \min_{a \in A} \left\{ maxRD(a, e) \right\}, \tag{3.10}$$

and we refer to $\arg\min_{a \in A}$ as the *minmax coding algorithm* for e.

Again, minmaxRD is always always non-negative. Notice that minmaxRD(e) = 0 for certain e, if and only if there exists a minmax coding algorithm a such that maxRD(a, e) = 0. Continuing the analysis from the previous paragraph, it should be clear that APCA is the minmax coding algorithm for every $e \geq 10$, since maxRD(APCA, e) = 0 for every $e \geq 10$.

Table 3.4 shows the maxRD(a, e) obtained for every pair of coding algorithm variant $a_v \in V^*$ and threshold parameter $e \in E$. For each e, the cell corresponding to the minmaxRD(e) value (i.e. the minimum value in the column) is highlighted.

	maxRD (%)							
Algorithm	e = 0	e = 1	e = 3	e = 5	e = 10	e = 15	e = 20	e = 30
PCA	40.52	42.28	53.11	62.01	71.73	75.33	77.21	80.28
APCA	33.25	15.64	9.00	29.96	0	0	0	0
CA	38.28	38.28	54.68	63.12	65.44	72.94	77.21	81.84
PWLH	73.46	72.93	72.52	82.14	83.24	86.86	88.94	91.19
PWLHInt	29.72	34.00	49.94	68.95	76.68	69.96	74.72	79.89
FR	48.75	49.85	52.21	52.70	54.82	55.35	54.48	64.72
SF	92.46	92.23	92.16	92.11	91.68	91.24	90.95	91.33
GAMPSLimit	85.84	85.73	84.37	83.92	83.02	83.00	82.88	82.22

TABLE 3.4: maxRD(a, e) obtained for every pair of coding algorithm variant $a_v \in V^*$ and threshold parameter $e \in E$. For each e, the cell corresponding to the minmaxRD(a) value is highlighted.

In the lossless case, PWLHInt is the minmax coding algorithm, with minmaxRD being equal to 29.72%. This value is rather high, which means that none of the considered algorithms achieves a CR that is close to the minimum simultaneously *for every data type*. Recalling the results from Table 3.3 we notice that $e = 0$ is the only threshold parameter value for which the minmax coding algorithm doesn't obtain the minimum CR in any combination. In other words, when

$e = 0$, PWLHInt is the algorithm that minimizes the RD with the best algorithm among every data type, even though it itself is not the best algorithm for any data type.

When $e \in [1, 3, 5]$, the minmax coding algorithm is always APCA, and the minmaxRD values are 15.64%, 9.00% and 29.96%, respectively. Again, these values are fairly high, so we would select the most convenient algorithm depending on the data type we want to compress. Notice that in the closest case (algorithm FR for $e = 5$), the second best maxRD (52.70%) is about 75% larger than the minmaxRD, which is a much bigger difference than in the lossless case.

When $e \geq 10$, the minmax coding algorithm is also always APCA, but in these cases the minmaxRD values are always 0. In the closest case (algorithm FR for $e = 20$) the second best maxRD is 54.48%. If we wanted to compress any data type with any of these threshold parameter values, we would pick algorithm APCA, since according to our experimental results, it always obtains the best compression results with a significant difference over the remaining algorithms.

3.4.1 Comparison with algorithm gzip

In this subsection we consider the results obtained by the general purpose compression algorithm gzip [10]. This algorithm only operates in lossless mode (i.e. the threshold parameter can only be $e = 0$), and it doesn't have a window size parameter w. Therefore, for each data type z of each dataset d, we have a unique CAI (and obtain a unique CR value) for gzip.

In all our experiments with gzip we perform a column-wise compression of the dataset files, which, in general, yields a much better performance than a row-wise compression. This is due to the fact that in most of our datasets, there is a greater degree of temporal than spatial correlation between the signals. All the reported results are obtained with the "`--best`" option of gzip, which targets compression performance optimization [11].

Table 3.5 summarizes the compression performance results obtained by gzip and the other evaluated coding algorithms, for each data type of each dataset. Similarly to Table 3.3, each row contains information relative to a certain data type, and for each threshold, the first column shows the CR obtained by the best CAI, the second column shows the base-2 logarithm of its window size parameter (when applicable), and the cell color identifies the best algorithm. Notice that, for $e > 0$ we compare the gzip lossless result with the results obtained by lossy algorithms.

We observe that algorithm gzip obtains the best compression results in 36 (21%) of the 168 possible data type and threshold parameter combinations. Algorithms APCA, PCA, and FR now obtain the best results in exactly 106 (63%), 23 (14%), and 3 (2%) of the total combinations, respectively. Algorithm APCA is still the best algorithm for most of the cases in which $e \geq 3$. However, now there is no value of e for which APCA outperforms the rest of the algorithms for every data type, since gzip is the best algorithm for at least one data type in every case. In particular, gzip obtains the best compression results for the data type "Size" of the dataset Hail for every e. We also observe that gzip obtains the best relative results against the other algorithms for smaller values of e, which is expected, since lossy algorithms performance improves for larger values of e. However, even for $e = 0$, gzip only outperforms the rest of the algorithms in about a half (10 out of 21) of the data types.

🟧 GZIP	🟦 PCA	🟩 APCA	🟨 FR

		e = 0		e = 1		e = 3		e = 5		e = 10		e = 15		e = 20		e = 30	
Dataset	Data Type	CR	w	CR	w	CR	w	CR	w	CR	w	CR	w	CR	w	CR	w
IRKIS	VWC	🟧 0.13		🟧 0.13		🟩 0.12	5	🟩 0.07	6	🟩 0.03	7	🟩 0.02	8	🟩 0.02	8	🟩 0.01	8
SST	SST	🟧 0.52		🟩 0.28	3	🟩 0.14	5	🟩 0.09	6	🟩 0.05	7	🟩 0.03	8	🟩 0.02	8	🟩 0.02	8
ADCP	Vel	🟧 0.61		🟧 0.61		🟧 0.61		🟩 0.61	2	🟩 0.48	2	🟩 0.41	2	🟩 0.35	3	🟩 0.26	3
Solar	GHI	🟧 0.69		🟧 0.69		🟧 0.69		🟩 0.67	4	🟩 0.59	4	🟩 0.52	4	🟩 0.47	4	🟩 0.38	4
	DNI	🟧 0.67		🟧 0.67		🟩 0.66	4	🟩 0.61	4	🟩 0.54	4	🟩 0.49	4	🟩 0.43	4	🟩 0.36	4
	DHI	🟧 0.61		🟧 0.61		🟧 0.61		🟧 0.61		🟩 0.60	4	🟩 0.54	4	🟩 0.48	4	🟩 0.39	4
ElNino	Lat	🟧 0.08		🟧 0.08		🟧 0.08		🟧 0.08		🟧 0.08		🟧 0.08		🟧 0.08		🟩 0.06	6
	Long	🟧 0.07		🟧 0.07		🟧 0.07		🟧 0.07		🟧 0.07		🟩 0.07	6	🟩 0.05	7	🟩 0.02	8
	Z. Wind	🟦 0.31	8	🟦 0.31	8	🟦 0.31	8	🟦 0.31	8	🟩 0.27	2	🟩 0.24	2	🟩 0.21	2	🟩 0.16	3
	M. Wind	🟦 0.31	8	🟦 0.31	8	🟦 0.31	8	🟦 0.31	8	🟩 0.29	2	🟩 0.26	2	🟩 0.23	2	🟩 0.19	2
	Humidity	🟦 0.23	8	🟦 0.23	8	🟦 0.23	8	🟦 0.23	8	🟩 0.21	2	🟩 0.18	2	🟩 0.16	2	🟩 0.13	2
	AirTemp	🟦 0.33	8	🟦 0.33	8	🟩 0.30	2	🟩 0.27	2	🟩 0.22	2	🟩 0.19	3	🟩 0.17	3	🟩 0.13	4
	SST	🟧 0.32		🟩 0.31	2	🟩 0.25	2	🟩 0.21	2	🟩 0.14	3	🟩 0.11	4	🟩 0.08	4	🟩 0.05	5
Hail	Lat	🟦 1.00	8	🟦 1.00	8	🟩 0.90	2	🟩 0.83	2	🟩 0.71	2	🟩 0.65	3	🟩 0.57	3	🟩 0.47	3
	Long	🟦 1.00	8	🟦 1.00	8	🟩 0.86	2	🟩 0.78	2	🟩 0.65	2	🟩 0.55	3	🟩 0.49	3	🟩 0.39	4
	Size	🟧 0.37		🟧 0.37		🟧 0.37		🟧 0.37		🟧 0.37		🟧 0.37		🟧 0.37		🟧 0.37	
Tornado	Lat	🟦 1.00	8	🟩 0.85	2	🟩 0.71	2	🟩 0.65	2	🟩 0.54	3	🟩 0.47	3	🟩 0.42	4	🟩 0.33	4
	Long	🟦 1.00	8	🟩 0.82	2	🟩 0.65	2	🟩 0.58	3	🟩 0.46	3	🟩 0.40	4	🟩 0.35	4	🟩 0.28	4
Wind	Lat	🟦 1.00	8	🟦 1.00	8	🟩 0.89	2	🟩 0.81	2	🟩 0.70	2	🟩 0.62	3	🟩 0.56	3	🟩 0.47	3
	Long	🟦 1.00	8	🟩 0.95	2	🟩 0.80	2	🟩 0.73	2	🟩 0.62	3	🟩 0.54	3	🟩 0.49	3	🟩 0.40	4
	Speed	🟨 0.65	4	🟩 0.44	3	🟨 0.26	6	🟨 0.17	7	🟩 0.16	5	🟩 0.12	6	🟩 0.10	6	🟩 0.08	6

TABLE 3.5: Compression performance of the best evaluated coding algorithm, for various error thresholds on each data type of each dataset, including the results obtained by gzip. Each row contains information relative to certain data type. For each threshold, the first column shows the minimum CR, and the second column shows the base-2 logarithm of the window size parameter for the best algorithm (the one that achieves the minimum CR), which is identified by a certain cell color described in the legend above the table. Algorithm gzip doesn't have a window size parameter, so the cell is left blank in these cases.

Similarly to Table 3.4, Table 3.6 shows the maxRD(a, e) obtained for every pair of coding algorithm variant $a_v \in V^* \cup \{\text{gzip}\}$ and threshold parameter $e \in E$. For each e, the cell correspondent to the minmaxRD(e) value is highlighted.

We observe that, for every e, the minmaxRD values are rather high, the minimum being 26.58% (algorithm gzip for $e = 0$). We conclude that none of the considered algorithms achieves a competitive CR *for every data type*, and the selection of the most convenient algorithm depends on the specific data type we are interested in compressing.

gzip is the minmax coding algorithm when $e \in [0, 1]$, and in both cases the minmaxRD values are rather high, i.e. 26.58% and 47.98%, respectively. APCA remains the minmax coding algorithm for every $e \geq 3$, but its minmaxRD values are now not only always greater than zero, but also quite high, ranging from 42.92% ($e = 30$) up to 54.42% ($e = 3$). This implies that there exist some data types for which the RD between the APCA and gzip CAIs is considerable, which means that, if we had the possibility of selecting gzip as a compression algorithm, APCA would no longer be the obvious choice for compressing any data type when $e \geq 10$, as we had concluded in the previous section.

	maxRD (%)							
Algorithm	e = 0	e = 1	e = 3	e = 5	e = 10	e = 15	e = 20	e = 30
GZIP	26.58	47.98	73.79	82.94	91.10	93.94	95.40	96.69
PCA	73.94	73.55	68.28	67.23	71.73	75.33	77.21	80.28
APCA	59.11	58.39	54.42	54.41	54.40	54.38	54.38	42.92
CA	73.83	73.46	69.31	68.30	65.44	72.94	77.21	81.84
PWLH	87.53	87.46	87.37	87.27	87.02	86.86	88.94	91.19
PWLHInt	71.26	71.01	70.71	68.95	76.68	69.96	74.72	79.89
FR	76.46	76.12	72.78	71.97	67.37	64.35	64.05	64.72
SF	96.31	96.13	96.09	95.96	95.87	95.74	95.64	95.05
GAMPSLimit	91.51	91.51	91.51	91.51	91.50	91.50	91.50	88.85

TABLE 3.6: maxRD(a, e) obtained for every pair of coding algorithm variant $a_v \in V^* \cup \{\text{gzip}\}$ and threshold parameter $e \in E$. For each e, the cell corresponding to the minmaxRD(a) value is highlighted.

3.5 Conclusions

In conclusion, our experimental results indicate that none of the implemented coding algorithms obtains a satisfactory compression performance in every scenario. This means that selection of the best algorithm is heavily dependent on the data type to be compressed and the error threshold that is allowed. In addition, we have shown that, in some cases, even a general compression algorithm such as gzip can outperform our implemented algorithms. In general, according to our results, algorithms APCA and gzip achieve better compression results for larger error thresholds, while PCA, APCA, FR and gzip are preferred for lower thresholds. Therefore, if one wishes to compress certain data type, our recommended way for choosing the appropriate algorithm is to select the best algorithm for said data type according to Table 3.6.

In our research we have also compared the compression performance of the coding algorithms' masking and non-masking variants. The experimental results show that on datasets with few or no gaps both variants have a similar performance, while on datasets with many gaps the masking variant always performs better, sometimes achieving a significative difference. We concluded that the masking variant of a coding algorithm is preferred, since it is more robust and performs better in general.

In addition, we have studied the extent to which the window size parameter **impacts/affects** the compression performance of the coding algorithms. We analyzed the compression results obtained when using optimal global and local window sizes. The experimental results reveal that the **impact/effect** of using the optimal global window size instead of the optimal local window size for each file is rather small.

3.6 Future Work (TODO)

Some ideas:

- Consider non-linear models (e.g. Chebyshev Approximation)
- Consider new datasets
- Investigate why certain algorithms perform better on certain data types
- Create universal coder, with every algorithm as a subroutine

Bibliography

[1] EnviDat - IRKIS Soil moisture measurements Davos. https://www.envidat.ch/dataset/soil-moisture-measurements-davos, 2019. [Accessed 27 July 2020].

[2] NOAA - TAO Data Download. https://tao.ndbc.noaa.gov/tao/data_download/search_map.shtml, 2016. [Accessed 27 July 2020].

[3] SolarAnywhere - Data. https://data.solaranywhere.com/Data, 2020. [Accessed 27 July 2020].

[4] El Nino Data Set. https://archive.ics.uci.edu/ml/datasets/El+Nino, 1999. [Accessed 27 July 2020].

[5] Kaggle - Storm Prediction Center. https://www.kaggle.com/jtennis/spctornado, 2016. [Accessed 27 July 2020].

[6] N.Q.V. Hung, H. Jeung, and K. Aberer. An Evaluation of Model-Based Approaches to Sensor Data Compression. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2434–2447, 2013.

[7] T. Bose, S. Bandyopadhyay, S. Kumar, A. Bhattacharyya, and A. Pal. Signal Characteristics on Sensor Data Compression in IoT - An Investigation. *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–6, 2016.

[8] Data Compression With Arithmetic Coding. https://marknelson.us/posts/2014/10/19/data-compression-with-arithmetic-coding.html, 2014. [Accessed 27 July 2020].

[9] Arithmetic Coding + Statistical Modeling = Data Compression. https://marknelson.us/posts/1991/02/01/arithmetic-coding-statistical-modeling-data-compression.html, 1991. [Accessed 27 July 2020].

[10] GNU Gzip. https://www.gnu.org/software/gzip/, 2018. [Accessed 27 July 2020].

[11] GNU Gzip Manual - Invoking gzip. https://www.gnu.org/software/gzip/manual/html_node/Invoking-gzip.html, 2018. [Accessed 27 July 2020].