Universidad de la República

Tesis de Maestría

Codificación de señales multicanal con muestreo irregular

Autor:
Pablo Cerveñansky

Supervisores:
Álvaro Martín
Gadiel Seroussi

Núcleo de Teoría de la Información
Facultad de Ingeniería

2 de septiembre de 2019

Capítulo 1

Datasets

1.1 Introducción

Explicar que todos los datasets se pasaron a un formato común.
Mostrar ejemplos de los csv.

Dataset	**Notación**	**#Archivos**	**#Tipos**
IRKIS	IRKIS	7	1
NOAA-SST	SST	3	1
NOAA-ADCP	ADCP	3	1
ElNiño	ElNiño	1	7
SolarAnywhere	Solar	4	3
NOAA-SPC-hail	Hail	1	3
NOAA-SPC-tornado	Tornado	1	2
NOAA-SPC-wind	Wind	1	3

Tabla 1.1: Resumen del conjunto de datasets.

1.2 IRKIS

1.3 SST

1.4 ADCP

1.5 ElNiño

1.6 Solar

1.7 Hail

1.8 Tornado

1.9 Wind

Capítulo 2

Codificadores

2.1 Introducción

2.2 CoderBase

```
entrada: archivo ∈ Datasets*: archivo csv a codificar
salida : CoderBase(archivo): archivo binario codificado con CoderBase
 1 out = crear_archivo_binario()
 2 out.codificar_entero(CODER_BASE, 8)
 3 out.codificar_header(archivo)
 4 out.codificar_entero(archivo.total_filas_de_datos(), 24)
 5 foreach columna in archivo.columnas do
 6 |   foreach entrada in columna.entradas do
 7 |   |   if entrada = NO_DATA then
 8 |   |   |   valor = columna.entero_no_data
 9 |   |   else
10 |   |   |   valor = entrada + columna.offset
11 |   |   end
12 |   |   out.codificar_entero(valor, columna.total_bits)
13 |   end
14 end
15 out.cerrar_archivo()
```

FIGURA 2.1: Pseudocódigo de *CoderBase*.

```
entrada: archivo: archivo binario codificado con CoderBase
salida : DecoderBase(archivo): archivo csv decodificado con DecoderBase
 1 out = crear_archivo_csv()
 2 codigo_codificador = archivo.decodificar_entero(8)
 3 out.decodificar_header(archivo)
 4 total_filas_de_datos = archivo.decodificar_entero(24)
 5 if codigo_codificador = CODER_BASE then
 6 |   foreach columna in out.columnas do
 7 |   |   foreach entrada in columna.entradas do
 8 |   |   |   valor = archivo.decodificar_entero(columna.total_bits)
 9 |   |   |   if valor = columna.entero_no_data then
10 |   |   |   |   out.escribir_string(NO_DATA)
11 |   |   |   else
12 |   |   |   |   out.escribir_string(valor − columna.offset)
13 |   |   |   end
14 |   |   end
15 |   end
16 else
17 |   . . . // si archivo fue codificado con otro codificador
18 end
19 out.cerrar_archivo()
```

FIGURA 2.2: Pseudocódigo de *DecoderBase*.

2.3 CoderPCA

2.4 CoderAPCA

2.5 CoderCA

2.6 CoderPWLH y CoderPWLHInt

2.7 CoderGAMPS y CoderGAMPSLimit

2.8 CoderFR

2.9 CoderSF

Capítulo 3

Resultados experimentales

En este capítulo presentamos los resultados experimentales del proyecto. El objetivo principal de nuestros experimentos era estudiar qué tan bien funcionaba cada uno de los codificadores implementados en el capítulo 2. Para ello, analizamos su rendimiento al codificar los distintos tipos de datos incluidos en los datasets presentados en el capítulo 1. En la sección 3.1 detallamos en qué consistieron los experimentos realizados. En la sección 3.2 comparamos el rendimiento relativo de los codificadores con y sin máscara. TODO: agregar secciones.

3.1 Experimentos

Los experimentos realizados consistieron en codificar cada uno de los tipos de dato del conjunto de datasets, tomando diferentes combinaciones de parámetros. En particular, se consideraron los siguientes tres parámetros:

- 15 codificadores:
  - *CoderBase*
  - *CoderPCA-NM* y *CoderPCA-M*
  - *CoderAPCA-NM* y *CoderAPCA-M*
  - *CoderCA-NM* y *CoderCA-M*
  - *CoderPWLH-NM* y *CoderPWLH-M*
  - *CoderPWLHInt-NM* y *CoderPWLHInt-M*
  - *CoderGampsLimit-NM* y *CoderGampsLimit-M*
  - *CoderFR-M*
  - *CoderSF-M*
- 7 tamaños de ventana: 4, 8, 16, 32, 64, 128 y 256.
- 8 umbrales de error:
  - 0 (codificación sin pérdida)
  - 1, 3, 5, 10, 15, 20 y 30 (codificación con pérdida)

Para referirnos a estos conjuntos utilizamos las notaciones C, T y U, respectivamente. Los experimentos consistieron en codificar cada tipo de dato iterando sobre todas las combinaciones posibles de parámetros $<c \in C,\ t \in T, u \in U>$.

Como explicamos en la sección 2.2, *CoderBase* es un codificador de baja complejidad, que implementamos con el fin de simplificar la comparación del rendimiento del resto de codificadores entre sí. Con ese objetivo, para cada combinación de parámetros se calculó la tasa de compresión del codificador para cada archivo de datos. A continuación definimos la tasa de compresión.

Definición 3.1.1. La *tasa de compresión* de un codificador $c \in C$ para un archivo a está dada por la ecuación

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

donde $|c(a)|$ y $|CoderBase(a)|$ son los tamaños de los archivos obtenidos al codificar a con c y con *CoderBase*, respectivamente.

Al disminuir $|c(a)|$, disminuye la tasa de compresión y mejora el rendimiento del codificador c. Por lo tanto, uno de nuestros objetivos era estudiar qué codificadores minimizan (3.1) al tomar distintas combinaciones de parámetros.

Nos parece importante hacer algunas aclaraciones adicionales acerca los experimentos realizados:

- El codificador *CoderBase* no tiene parámetro tamaño de ventana y es el único codificador que solamente puede codificar sin pérdida.
- El codificador *CoderSF-M* no tiene parámetro tamaño de ventana.
- No se implementaron versiones sin máscara de los codificadores *CoderFR-M* y *CoderSF-M*. Más detalles sobre este punto en las secciones 2.8 y 2.9.
- Para los codificadores *CoderPCA-NM* y *CoderPCA-M* el tamaño de ventana es fijo, mientras que en el resto de los algoritmos (salvo *CoderBase* y *CoderSF-M*) el tamaño de ventana es variable y el parámetro indica su tamaño máximo.
- Con el parámetro umbral de error hacemos un abuso de notación, ya que en realidad, cuando tomamos $u \in U$, el umbral de error que consideramos es el $u\%$ de la desviación estándar del tipo de datos que se quiere codificar.

3.2 Rendimiento relativo de los codificadores

En esta sección analizamos el rendimiento relativo de los codificadores con (*M*) y sin (*NM*) máscara al codificar los archivos de los datasets. Solamente nos interesa comparar un par de codificadores entre sí cuando sus implementaciones parten del mismo algoritmo original pero una utiliza máscara y la otra no. Por lo tanto, del conjunto de codificadores definidos en la sección 3.1, vamos a comparar:

- *CoderPCA-NM* y *CoderPCA-M*
- *CoderAPCA-NM* y *CoderAPCA-M*
- *CoderCA-NM* y *CoderCA-M*
- *CoderPWLH-NM* y *CoderPWLH-M*
- *CoderPWLHInt-NM* y *CoderPWLHInt-M*
- *CoderGampsLimit-NM* y *CoderGampsLimit-M*

Para comparar el rendimiento de dos codificadores al codificar cierto archivo, se calcula la diferencia relativa entre ambos, la cual definimos a continuación.

Definición 3.2.1. La *diferencia relativa* entre un codificador $c_M \in C_M$ y un codificador $c_{NM} \in C_{NM}$ para un archivo a está dada por la ecuación

$$\mathit{DiferenciaRelativa}(c_M, c_{NM}, a) = \begin{cases} 100 \times \frac{|c_{NM}(a)| - |c_M(a)|}{|c_{NM}(a)|}, & \text{si } |c_{NM}(a)| \neq |c_M(a)|, \\ 0, & \text{si } |c_{NM}(a)| = |c_M(a)|, \end{cases} \tag{3.2}$$

donde $|c_M(a)|$ y $|c_{NM}(a)|$ son los tamaños de los archivos obtenidos al codificar a con c_M y con c_{NM}, respectivamente.

El codificador c_M logra una mejor tasa de compresión que el codificador c_{NM} cuando el resultado de la ecuación (3.2) es positivo. Al aumentar el resultado, mejora el rendimiento relativo del codificador c_M respecto al codificador c_{NM}.

Dado cierto $u \in U$, al comparar dos codificadores c_M y c_{NM}, consideramos únicamente el resultado obtenido con los tamaños de ventana $t_M, t_{NM} \in T$ que minimizan la tasa de compresión de cada codificador. O sea que para cada codificador solamente consideramos la ventana con la que se logra el mejor rendimiento.

En la tabla 3.1 se muestra un resumen de los resultados obtenidos al comparar el rendimiento relativo de los codificadores c_{NM} y c_M para cada cada dataset. En la tercera y cuarta columnas aparece el porcentaje de las combinaciones $<c \in C, u \in U>$ con las que se obtiene la mejor tasa con cada codificador. En la última columna se muestra el rango en el que varía el resultado de la ecuación *DiferenciaRelativa* para dichas combinaciones.

En los datasets que tienen muchos gaps, *DiferenciaRelativa* siempre es positiva, o sea que en todos los casos se obtienen mejores tasas de compresión al utilizar los algoritmos c_M. En cambio, en los datasets sin gaps, *DiferenciaRelativa* es negativa, por lo que siempre se tiene mejor rendimiento con los algoritmos c_{NM}. En el dataset con pocos gaps, en cada mitad de las combinaciones se obtienen mejores tasas con algoritmos distintos.

Dataset	**Información**	**Mejor c_{NM}**	**Mejor c_M**	**Rango *DiferenciaRelativa***
IRKIS	Muchos gaps	-	100%	(0; 36.88]
SST	Muchos gaps	-	100%	(0; 50.60]
ADCP	Muchos gaps	-	100%	(0; 17.35]
ElNiño	Muchos gaps	-	100%	(0; 50.52]
Solar	Algunos gaps	49%	51%	[-0.25; 1.77]
Hail	Sin gaps	100%	-	[-0.04; 0)
Tornado	Sin gaps	100%	-	[-0.29; 0)
Wind	Sin gaps	100%	-	[-0.12; 0)

TABLA 3.1: Rendimiento relativo de los codificadores c_{NM} y c_M. Están marcadas en azul y rojo las máximas diferencias relativas a favor de los algoritmos c_M y c_{NM}, respectivamente

Observamos que, siempre que se obtienen mejores tasas con el codificador c_{NM}, la diferencia relativa está cerca de 0. En las gráficas de la figura 3.1 se muestra el caso en el que se obtiene la mejor diferencia relativa a favor de c_{NM}. Esto ocurre para el tipo de dato "Longitude" del dataset Tornado, al comparar los codificadores *CoderAPCA-NM* y *CoderAPCA-M*, con el umbral de error igual a 30. En la tabla 3.1 verificamos que en dicho caso la diferencia relativa vale -0.29.

Por otro lado, cuando se logran mejores tasas con el codificador c_M, las diferencias relativas alcanzan mayores valores absolutos, con un máximo de 50.60 para el tipo de dato "VWC" del dataset SST. En las gráficas de la figura 3.2 vemos que dicho resultado se obtiene al comparar los codificadores *CoderPCA-NM* y *CoderPCA-M*, con el umbral de error igual a 30.

Teniendo en cuenta los resultados presentados, si quisiéramos codificar un dataset que a priori supiéramos tiene muchos gaps, obviamente nos convendría utilizar un algortimo c_M. Sin embargo, incluso si el dataset no tuviera gaps, la diferencia de rendimiento a favor del algoritmo c_{NM} sería despreciable. Como el algoritmo c_M es más robusto y funciona mejor en general, en las próximas secciones nos vamos a enfocar en su estudio.

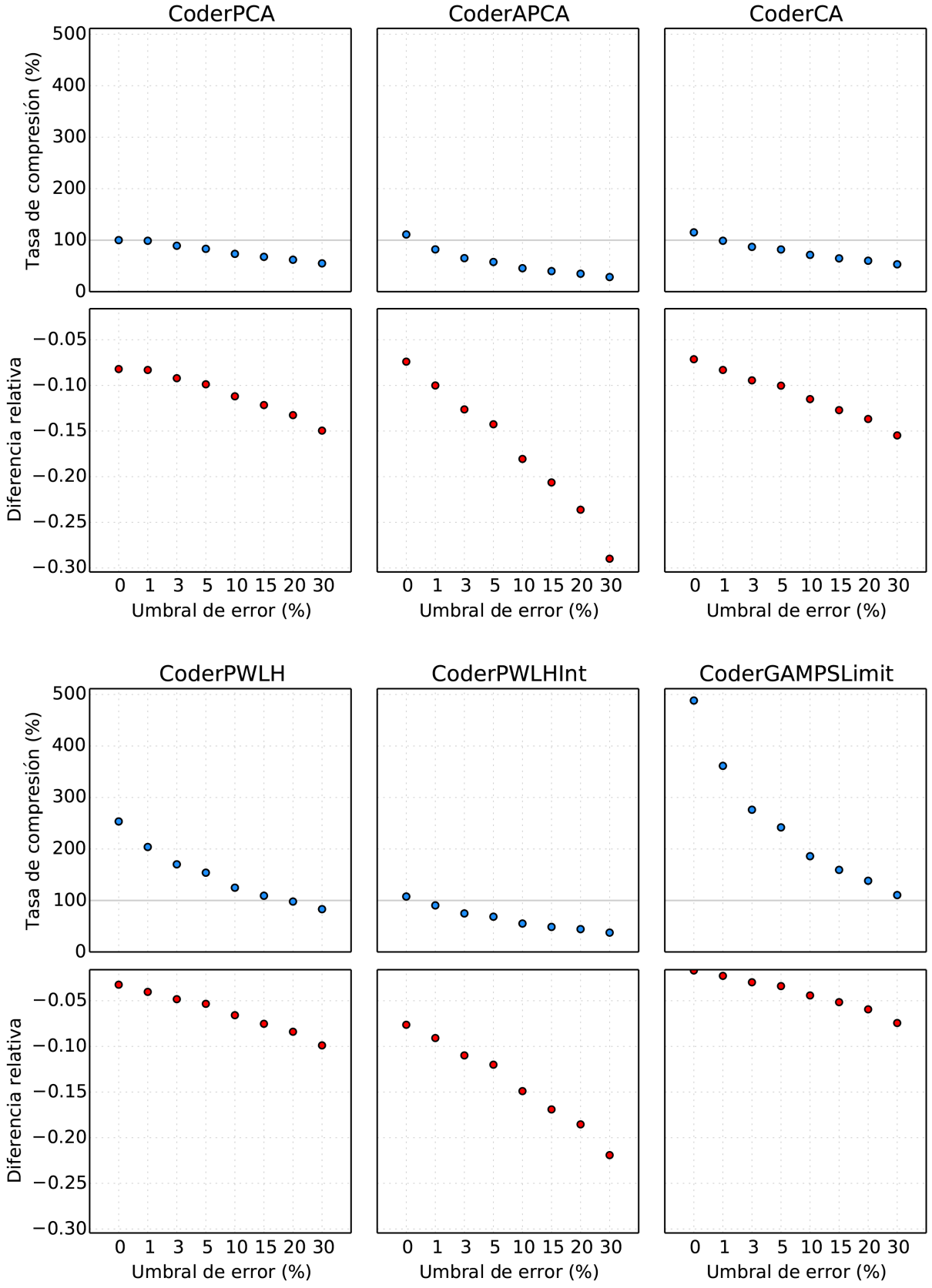

FIGURA 3.1: Tasa de compresión y Diferencia relativa para las distintas combinaciones $<c \in C,\ u \in U>$ para el tipo de dato "Longitude" del dataset Tornado.

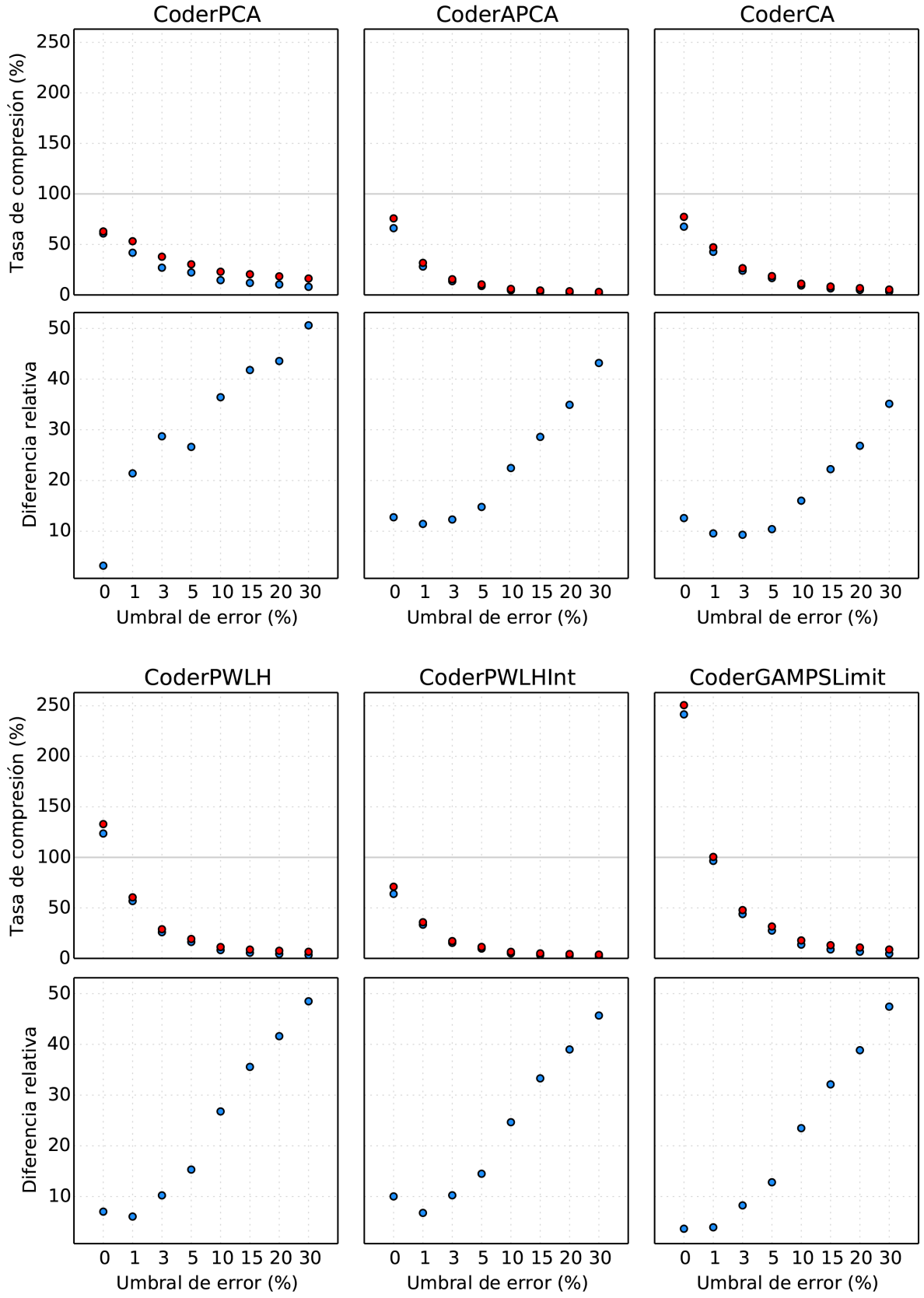

FIGURA 3.2: Tasa de compresión y Diferencia relativa para las distintas combinaciones $<c \in C, u \in U>$ para el dataset SST.

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.