LZCOMPLEXITY: BIBLIOTECA PARA EL CÁLCULO DE MEDIDAS ENTRÓPICAS

LZCOMPLEXITY: ENTROPY MEASURING LIBRARY

E. Aragón-Pérez[a], E. Estévez-Rams[b,c†]

a) Universidad Central "Marta Abreu" de Las Villas, 54830 Santa Clara, Villa Clara, Cuba
b) Facultad de Física, Universidad de La Habana, 10400 La Habana, Cuba; estevez@fisica.uh.cu†
c) Instituto de Ciencias y Tecnología de Materiales (IMRE), Universidad de La Habana, 10400 La Habana, Cuba
†para la correspondencia

Recibido 3/10/2024; Aceptado 26/5/2025

Se presenta la biblioteca *lzcomplexity*, desarrollada para el análisis de complejidad utilizando métricas entrópicas. Se describen los algoritmos utilizados para estimar las métricas empleando la factorización de Lempel-Ziv. Para enfrentar los retos computacionales asociados al análisis de grandes volúmenes de datos, *lzcomplexity* implementa algoritmos que permiten un procesamiento paralelo de las secuencias. Los experimentos realizados y su uso prácticos demuestran que *lzcomplexity* puede analizar secuencias extensas en tiempos razonables y que las medidas entrópicas incluidas son un indicador confiable para describir los sistemas complejos.

The library *lzcomplexity* is presented, developed for complexity analysis using entropic metrics. The algorithms used to estimate the metrics by means of Lempel–Ziv factorization are described. To face the computational challenges associated with the analysis of large volumes of data, *lzcomplexity* implements algorithms that enable parallel processing of the sequences. The experiments carried out and its practical use demonstrate that *lzcomplexity* can analyze long sequences in reasonable times and that the included entropic measures are reliable indicators for describing complex systems.

Keywords: complexity (complejidad), entropy (entropía), Lempel-Ziv.

I. INTRODUCCIÓN

Capturar de manera cuantitativa el elusivo término de complejidad en una pocas magnitudes es un reto sin solución general hasta el momento. La tarea bien puede ser imposible, por definición complejidad implica, en su generalidad, comportamientos que no pueden ser reducidos a la suma de sus partes y en ellos, la emergencia de dinámicas nuevas es una característica ubicua. Teniendo eso en cuenta, no obstante, caracterizar tales sistemas con herramientas robustas pero generales ha sido y sigue siendo un objetivo de la investigación y para ello se busca en cuerpos matemáticos de aplicabilidad amplia bajo pocos axiomas como estacionaridad y ergodicidad. Teoría de la información [1, 2] es uno de esos cuerpos matemáticos que mejor cumplen con tales condiciones y por ello, han sido utilizados extensivamente en el estudio de sistemas complejos [3–8].

Aunque definido de manera clara y matemáticamente preciso, la estimación de la entropía de Shannon, la magnitud esencial en la teoría de la información, está lejos de ser trivial. Su definición matemática se hace sobre ensembles infinitos o secuencias bi-infinitas, cuando en todos los casos de importancia práctica, la finitud de la data es inevitable.

Otro cuerpo matemático cuya generalidad lo hace, al menos teóricamente, adecuado para la misma tarea, es la complejidad de Kolmogorov o complejidad algorítmica que tiene una fuente importante en los estudios de computabilidad de Turing y la definición de la Máquina Universal de Turing [9]. El decir que la complejidad algorítmica es de utilidad teórica descansa en el hecho de que su uso práctico está impedido, en toda su generalidad, por el teorema de la parada de Turing. Eso implica la necesidad de buscar realizaciones prácticas de contexto más limitado.

Los comentarios anteriores implican que la estimación de las magnitudes descritas es un tema de investigación tanto desde el punto de vista teórico como práctico en la búsqueda de herramientas utilizables para estudiar sistemas complejos concretos, escapando de la frontera de los desarrollos puramente formales o teóricos. En esa dirección, en 1976, Lempel y Ziv, introdujeron una factorización, llamada exhaustiva, de cadenas finitas que conduce de manera natural a definir como una magnitud de interés el número de factores como medida de complejidad de una secuencia [10]. Introducida en el contexto de la compresión de la información, un teorema posterior [11] demuestra que la hoy conocida como complejidad de Lempel-Ziv, es, en el límite superior, igual a la tasa de entropía de Shannon, a la entropía de Shannon por símbolo. El teorema de Ziv, por tanto, establece el puente entre el uso teórico y práctico de las magnitudes entrópicas y de ahí, permite también definir una variante restringida pero útil de distancia informacional que descansa en la complejidad de Kolmogorov [12].

La implementación eficiente de un algoritmo que realice la factorización de Lempel-Ziv es un reto abordado por un número de autores y se reportan sistemáticamente mejoras en las heurísticas. En la actualidad existen distintas bibliotecas de programación orientadas al análisis de complejidad

utilizando medidas entrópicas como [13–16]. En la práctica, muchos sistemas de interés pueden implicar una cantidad masiva de datos que incluso pueden llegar a 10^9 símbolos como en el análisis de material genético o el monitoreo en tiempo real de sensores o los flujos finacieros. Esto implica un desafío para el cómputo y la memoria utilizada, lo que se requiere de algoritmos eficientes que puedan aprovechar la arquitectura de hardware moderno para reducir el tiempo de procesamiento y manejar la complejidad computacional.

En esta contribución reportamos una biblioteca que llamaremos *lzcomplexity* como una solución a estas demandas extremas. En la biblioteca se integran algoritmos optimizados para el análisis de secuencias mediante el uso de procesamiento paralelo. Optimización necesaria debido a que el procesamiento secuencial de grandes volúmenes de datos es insostenible en términos de tiempo y recursos. En la solución propuesta se paraleliza el algoritmo descrito en [17]. A diferencia de otras bibliotecas, que se limitan al cálculo de la densidad de entropía, *lzcomplexity* incluye otras métricas entrópicas como la medida efectiva de complejidad [18] para analizar la transferencia de información en los sistemas y la distancia informacional [12] para realizar comparaciones entre sistemas.

II. FUNDAMENTOS TEÓRICOS

II.1. Medidas entrópicas

La entropía de Shannon [1,2], magnitud central de la teoría de la información, permite determinar el nivel de incertidumbre de un sistema. La ocurrencia de los eventos en el sistema se consideran una variable aleatoria X y cada evento se emite con una probabilidad p_i [19]:

$$H(X) = -\sum_i p_i \log p_i. \tag{1}$$

La información mutua, por otro lado, mide cuanto de común, en términos de información, tiene la variable X, respecto a la variable Y:

$$I(X:Y) = H(X) + H(Y) - H(XY). \tag{2}$$

Cuando el sistema es descrito por una secuencia de símbolos, las magnitudes anteriores se adaptan para medirse sobre la ocurrencia de subcadenas en la secuencia que se asume bi-infinita. Una condición necesaria para que estas definiciones tengan sentido es considerar que el proceso es ergódico, lo que implica que para una longitud suficientemente larga, toda subcadena mayor o igual a dicha longitud es representativa del sistema como un todo, permitiendo pasar del promedio sobre ensembles al promedio sobre cadenas.

La entropía de bloque de Shannon se define como

$$H_n = -\sum_{\{X_n\}} p_i^n \log p_i^n, \tag{3}$$

donde $\{X_n\}$ es el conjunto de cadenas de longitud n extraibles de la cadena bi-infinita que describe al sistema, p_i^n es la probabilidad de ocurrencia de una cadena particular de longitud n.

Utilizando la entropía de bloque es posible obtener la densidad de entropía del sistema o el nivel de desorden irreducible del mismo

$$h = \lim_{n\to\infty} \frac{H_n}{n} \tag{4}$$

La densidad de entropía está estrechamente relacionada con la compresión máxima que es posible alcanzar con una codificación optima de la secuencia [11].

Otra medida entrópica de utilidad es la medida efectiva de complejidad (exceso de entropía) o E, que se puede definir como la información mutua entre las dos mitades infinitas de la secuencia [18]:

$$E(X) = \sum_{n=1}^{\infty} [h_n(X) - h(X)], \tag{5}$$

con $h_n(X) = H_n - H_{n-1}$ como la ganancia de entropía al pasar de considerar bloques de longitud $n-1$ a bloques de longitud n. Teniendo h y E es posible construir mapas de complejidad-entropía [20,21]. Los mapas o diagramas de complejidad-entropía muestran el balance alcanzado por el sistema entre las medidas de desorden y almacenamiento de la información.

La distancia informacional normalizada [12] compara dos sistemas desde un punto de vista algorítmico utilizando la complejidad de Kolmogorov [19]. La complejidad de Kolmogorov $K(X)$ se define como el algoritmo más pequeño capaz de producir una secuencia X en una Maquina Universal de Turing. La complejidad de Kolmogorov es no computable debido al teorema del problema de parada de Turing [9]. Si se tienen dos sistemas, y uno de ellos es posible derivarlo a partir del otro por medio de un algoritmo corto la distancia entre estos dos sistemas sería muy cercana a cero. Lo contrario pasaría si las secuencias no tienen ningún tipo de correlación algorítmica, la distancia en ese caso es cercana a 1, por la necesidad de un algoritmo tan largo como las secuencia misma para transformar una secuencia en la otra. La distancia informacional se define como:

$$d_K(X,Y) = \frac{K(XY) - min(K(X), K(Y))}{max(K(X), K(Y))}, \tag{6}$$

donde XY significa la concatenación de las cadenas X y Y.

Como se ha mencionado, las medidas para el análisis de complejidad están sustentadas teóricamente sobre la premisa de que las secuencias son bi-infinitas, en la práctica tenemos que lidiar con secuencias finitas y formas de estimación se hacen necesarios.

II.2. La factorización de Lempel-Ziv

Una historia de $S = s_1 s_2 \dots s_n$ es una serie de subcadenas o factores $s(i,j)$ con $1 \le i,j \le n$ que concatenados producen como resultado a $S = s(1,l_1)s(l_1+1,l_2)...s(l_{k-1}+1,l_k)$.

Para que una historia sea exhaustiva es necesario que cada factor $s(l_{i-1}+1, l_i)$ no sea una subcadena del prefijo $s(1, l_i - 1)$. Por ejemplo, para la secuencia binaria $S = 0010101110$ la descomposición nos quedaría 0.01.01011.10 donde cada factor está delimitado por un punto. Cada secuencia tiene una historia exhaustiva única.

La complejidad de Lempel-Ziv (C_{lz}) es el número de factores de la historia exhautiva [10]. Para el ejemplo anterior, la $C_{lz}(S) = 4$.

En el límite $n \to \infty$ se cumple [10]

$$C_{lz} \leq \frac{n}{\log n}. \tag{7}$$

Definiendo a $c_{lz}(S)$ como:

$$c_{lz}(S) = \frac{C_{lz}(S)}{n/\log n}. \tag{8}$$

En [11], se demuestra que para todas las secuencias S generadas por una fuente ergódica se cumple

$$\lim_{N\to\infty} \sup c_{lz}(S) = h(S). \tag{9}$$

En la práctica para obtener un valor estimado de la densidad de entropía se emplea la ecuación (8) para una secuencia de longitud n representativa (en el sentido del Teorema de la Propiedad Asimptótico de Equipartición) del sistema.

Para la estimación de E se utiliza la expresión:

$$E_{lz}(S) = \sum_{m=1}^{M} (c_{lz}(S^m) - c_{lz}(S)) \tag{10}$$

donde S^m es el resultado de dividir la secuencia S en bloques disjuntos de tamaño m y realizar un reordenamiento aleatorio de los bloques. De esta forma se rompen las correlaciones entre los conjuntos de valores mayores a m pero se mantiene la frecuencia de los símbolos. En [22] se muestra como a pesar de que E_{lz} no es estrictamente E, se comporta de manera similar.

Para estimar la distancia informacional, se utilizará:

$$d_{lz}(S, P) = \frac{c_{lz}(SP) - min(c_{lz}(S), c_{lz}(P))}{max(c_{lz}(S), c_{lz}(P))} \tag{11}$$

Para obtener buenas estimaciones de h y E es necesario procesar cadenas lo suficientemente largas para lograr captar el comportamiento del sistema. Para esto es necesario tener algoritmos eficientes que permitan obtener C_{lz} en períodos relativamente cortos de tiempo. En *lzcomplexity* se modifica el algoritmo presentado en [17] para calcular C_{lz} de forma paralela, los detalles de la bliblioteca se exponen en la siguiente sección.

III. IMPLEMENTACIÓN Y ARQUITECTURA DEL SOFTWARE

La biblioteca *lzcomplexity* está orientada al análisis entrópico de secuencias temporales. La biblioteca permite calcular las métricas definidas en la sección anterior. La factorización de Lempel-Ziv (*LZ76*) es la base fundamental para estimar la densidad de entropía, la medida efectiva de complejidad y la distancia informacional (figura 1). Una vez calculada la densidad de entropía, es posible el cálculo de las demás magnitudes. Cada método está implementado aprovechando las funcionalidades de la biblioteca *oneTBB* [23] para que puedan ser ejecutados de forma paralela.

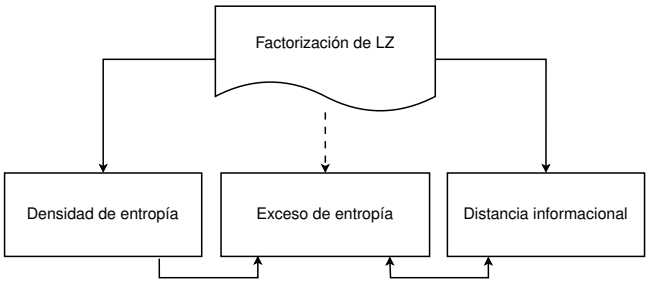

Figura 1. Se muestra la relación entre las distintas métricas calculadas por la biblioteca. La factorización de Lempel-Ziv es la base para obtener la densidad de entropía, la medidad efectiva de complejidad y la distancia informacional.

Para el desarrollo de la biblioteca se utilizó el lenguaje de programación C/C++ [24], aprovechando la eficiencia del código que genera y la madurez de las optimizaciones existentes en los compiladores correspondientes. Además de la implementación base de *lzcomplexity*, se desarrolló un enlance para extender el uso de las funciones incluidas en la biblioteca al lenguaje de programación Python [25]. En la construcción de estos enlaces se utiliza *pybind11* [26] que es una biblioteca ligera implementada sobre C++.

Para obtener el valor C_{lz} de una secuencia, la biblioteca brinda la función *lz76Factorization*. El algoritmo implementado en esta función consta de tres fases [17].

La fase inicial ordena ascendentemente todos los sufijos de la secuencia de entrada S. Un sufijo es la subsecuencia formada por los símbolos a partir del índice i hasta el final de la secuencia ($s(i, n)$). Luego de obtener el arreglo de todos los sufijos ordenados (SA) se hallan las longitudes de los prefijos más largos (LCP) entre sufijos concecutivos.

Existen varios algoritmos secuenciales de complejidad lineal como [27–31] que permiten obtener los SA y LCP. De igual forma se han creado algoritmos paralelos como [32–34]. En la bilbioteca se implementó el algoritmo CaPS-SA que muestra mejor tiempo de ejecución que otros algoritmos paralelos [34] y que utiliza el principio detrás del algoritmo samplesort [35], con el empleo de un número de particiones y un conjunto de pivotes para lograr ordenar paralelamente el conjunto de sufijos. En el peor de los casos el algoritmo se comporta con una complejidad computacional de $O(n^2 \log(\frac{n}{p}))$ y una complejidad espacial de $4\omega N$, con $\omega = \{4, 8\}$ el número de bytes del tipo de dato utilizado para almacenar los valores de la secuencia.

Con los SA y los LCP se calcula el mayor factor anterior (LPF) para cada posición i de S. El vector LPF contiene la longitud del mayor factor comenzando en la posición i-ésima y tiene una ocurrencia anterior en S. Es decir, para cada uno de los valores de i, $LPF[i] = max(l)$ tal que $s(i, i + l - 1)$ es un factor en

$s(1, i + l - 2)$. Para obtener los valores de *LPF* se implementó el algoritmo propuesto en [17] que tiene una complejidad computacional de $O(n)$ independientemente del alfabeto de la secuencia. Finalmente se obtiene la factorización de LZ76 por medio de los *LPF* (Algoritmo 1). El algoritmo genera como resultado un vector lzf que contiene los índices, en relación con la secuencia original, donde comienzan cada uno de los factores de *LZ*76. Por lo tanto $C_{lz}(S) = size(lzf)$

Entrada: *LPF*
Salida : arreglo lzf con la posición de los factores

```
lzf[0] = 0, lzf[1] = 1;
i = 1;
while lzf[i] ≤ size(LPF) do
    lzf[i + 1] = lzf[i] + LPF[lzf[i]] + 1;
    i = i + 1;
end
return lzf;
```

Algoritmo 1. Algoritmo para obtener la factorización de LZ76 a partir de los valores de *LPF*. Se calcula el vector lzf que contiene los índices donde comienzan cada uno de los factores de la factorización

En caso de querer obtener, además de C_{lz}, los factores que conforman la historia exhaustiva de la secuencia, se debe utilizar la función *lz*76*Factors*. Para la denesidad de entropía (h_{lz}) la biblioteca brinda la función *lz*76*EntropyDensity*. Procedimiento que no agrega complejidad al procedimiento anterior debido a que para obtener h_{lz} luego de tener C_{lz} se realiza una operación matemática.

La distancia informacional d_{lz} se puede obtener al realizar una llamada a la función *lz*76*InformationDistance*. El método calcula la complejidad C_{lz} de las dos secuencias a comparar y la concatenación entre estas. Como cada procesamiento es independiente entre sí, es posible efectuarlos de forma paralela utilizando la función *parallel_do* de *oneTBB*. Luego siguiendo la ecuación (11) se combinan los resultados de cada una de las funciones para obtener el valor final.

Por otro lado, para calcular la medida efectiva de complejidad se utiliza la función *lz*76*RandomShuffleComplexity*. El algoritmo implementado necesita determinar las diferencias entre $c_{lz}(S^m)$ y $c_{lz}(S)$ con $m = 1 \ldots M$ (veáse ecuación (12)). Posteriormente, se suman todas estas diferencias para obtener el resultado final.

En la primera etapa de procedimiento, son necesarias $M + 1$ llamadas a la función de factorización *lz*76*Factorization*. Similar al algoritmo de la distancia, las $M + 1$ llamadas al método son posible realizarse en su totalidad de forma paralela, combinando en este caso las funciones *parallel_do* y *parallel_for* de *oneTBB*. Como resultado se obtiene la factorización de la secuencia original y un vector de complejidades, donde cada posición i contiene el valor de $C_{lz}(S^i)$ con $i = 1..M$. Luego sobre este vector se aplica la técnica map-reduce, aprovechando la propiedad asociativa de la suma. Es decir, para cada elemento en el vector de complejidades, se calcula el $c_{lz}(S^i)$ y luego la diferencia con respecto a $c_{lz}(S)$. Finalmente, se reducen los resultados sumando cada diferencia. Este procedimiento es posible hacerlo de forma paralela empleando la función *parallel_reduce* de *oneTBB*.

Una variación del algoritmo podría ser realizar el cálculo de $C_{lz}(S^m)$ dentro del proceso de map-reduce. En este caso, se aprovecha el mapeo de los valores de m para calcular la factorización de cada una de las secuencias generadas a partir de la mezcla, justo antes de realizar la resta con $c_{lz}(S)$. Esta variante reduce el uso de la memoria al no tener que generar un vector auxiliar de complejidades. En relación a la complejidad computacional ambas variantes presentan igual complejidad.

En caso de quere obtener todas las métricas entrópicas que caracterizan una secuencia dada, la biblioteca ofrece la función *lz*76. Función que optimiza las llamadas al método que permite calcular la factorización de una secuencia. De esta forma se utiliza menos memoria y se reduce el tiempo de cálculo para extraer los valores entrópicos de la secuencia.

Todas las funciones permiten ser invocadas de dos formas. Una forma básica donde solo se necesita una cadena de caracteres de entrada. Y una segunda forma, donde además de la secuencia, las funciones permiten recibir un objeto LZ_Args donde el usuario va a poder controlar distintos hiperparámetros de los algoritmos. Con este objeto podemos forzar un número de símbolos en el alfabeto de las secuencias, definir una base para el cálculo de los logaritmos o poder controlar el número de particiones utilizadas por CaPS-SA.

IV. EVALUACIÓN DE RENDIMIENTO

En la figura 2 se muestra el promedio del tiempos de ejecución del algoritmo CaPS-SA y el SA-IS [36] sobre 20 secuencias binarias generadas de forma aleatoria para longitudes de 10^8 y 10^9. Los algoritmos se ejecutaron en el cluster de la Universidad Central "Marta Abreu" de Las Villas en un nodo con 16 CPUs. Se puede apreciar como el algoritmo paralelo, al igual que en los resultado de [34], muestra una ejecución eficiente al procesar secuencia de 10^9 símbolos en menos de 80 segundos.

Con el fin de evaluar el desempeño del método utilizado para obtener *LZ*76, la figura 3 muestra los tiempos de ejecución obtenidos para la factorización de secuencias de distintas longitudes. La comparación se realizó con otras dos implementaciones pertenecientes a las bibliotecas de código abierto, Complexity[1] y antropy[2]. Ambas implementaciones corresponden al algoritmo clásico propuesto por Lempel y Ziv en 1976 [10], algoritmo que computacionalmente se comporta en el orden de $O(n)$. Una de las implementaciones está hecha sobre el lenguaje de programación C/C++ y la otra emplea el lenguaje Python con optimizaciones brindadas por el compilador numba [37].

[1] https://github.com/kyralianaka/Complexity
[2] https://github.com/raphaelvallat/antropy

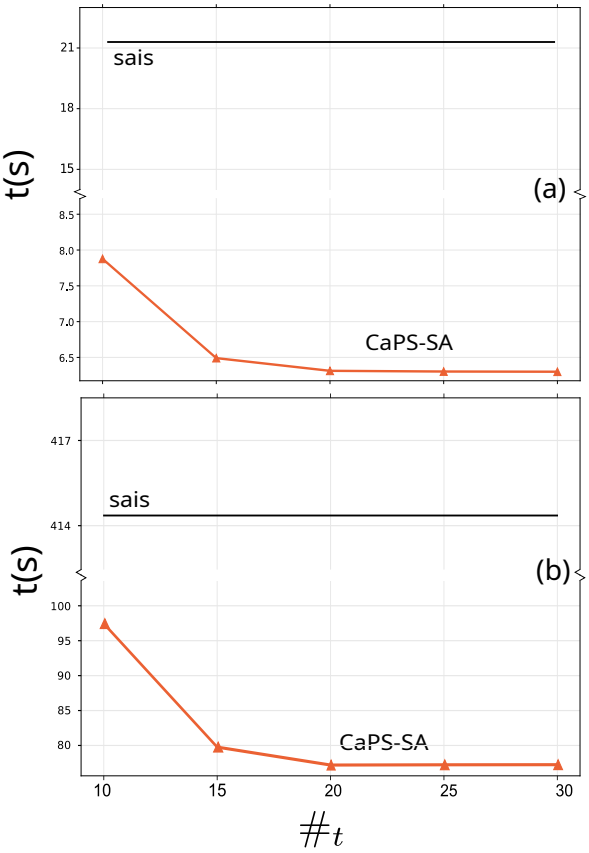

Figura 2. Comparación entre el tiempo de ejecución promedio como función del número de hilos $\#_t$, del algoritmo secuencial *sais* y el algoritmo paralelo *CaPS-SA* para la construcción del SA: (a) secuencias de longitud 10^8 y (b)) longitudes de 10^9. En la medida que el número de hilos aumenta, el algoritmo paralelo mejora su rendimiento respecto al algoritmo secuencial.

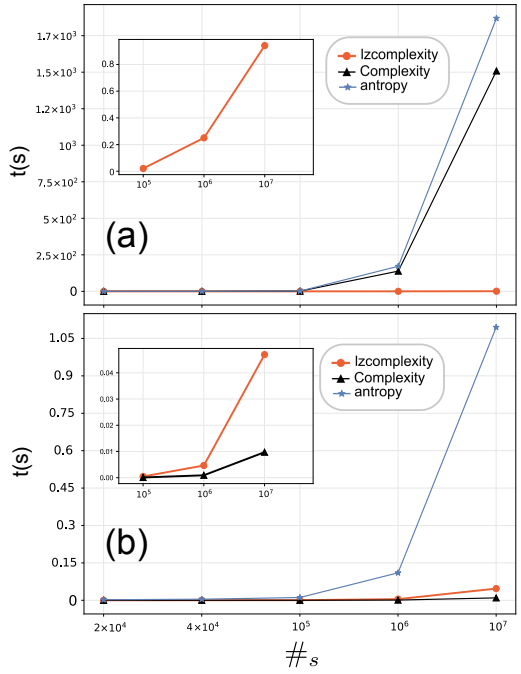

Figura 3. Comparación entre el tiempo de ejecución promedio como función del número de símbolos $\#_s$ para (a) secuencias aleatorias y (b) secuencias periódicas. Los tiempos se corresponden con la función *lz76Facotrization* de *lzcomplexity*, la implementación en C/C++ de *Complexity* y la implementación en Python de *antropy*.

Las pruebas se realizaron sobre secuencias que varían el número de símbolos entre 2×10^4 y 10^7. Se emplearon secuencias periódicas y secuencias totalmente aleatorias para de esta forma comparar el rendimiento de las implementaciones sobre cadenas de complejidad mínima y con entropía máxima. Los resultados alcanzados se corresponden con el promedio de 10 ejecuciones por cada tipo de secuencia, realizadas en una MacBook Pro con el chip M2 Pro de 10 CPU y 16Gb de memoria RAM.

Las pruebas muestran como el procedimiento expuesto en este trabajo se mantiene de forma estable por debajo del segundo de duración para todas las longitudes del experimento. En el caso de secuencias periódicas (figura 3b) la implementación de *Complexity* logra alcanzar mejores tiempos de ejecución. Esto se debe a que el número de comparaciones que se realizan sobre cada símbolo es mínimo debido a la periodicidad de las secuencias. Aún así, la diferencia del tiempo de ejecución con respecto a la implementeción de *lzcomplexity* se puede considerar irrelevante estando en el orden de 10^{-2}.

Muy diferente ocurre con las secuencias de máxima entropía (figura 3a) donde existen un alto número de factores. Los tiempos de ejecución de las otras librerías utilizadas sobrepasan los 20 s cuando las secuencias alcanzan los 10^6 símbolos, llegando a sobrepasar los 1.5×10^3 segundos aproximadamente para cadenas de longitud 10^7. En contraste nuestro procedimiento de queda por debajo de los 0.8 s, lo que atestigua la eficiencia estable del algoritmo implementado a medida que las secuencias crecen en longitud y en complejidad.

V. APLICACIONES

Para ejemplificar el uso de *lzcomplexity* se utilizaron las magnitudes entrópicas para clasificar diferentes idiomas en familias linguísticas. El lenguaje humano ha sido caso de estudio por su diversidad y complejidad [8, 38]. Visto como resultado de un sistema emisor de símbolos, el lenguaje natural es un sistema que tiene un nivel de estructura dado por las reglas gramaticales y semánticas, así como la lógica temporal que le confiere su comprensibilidad. A la vez, un texto en lenguaje natural tiene un nivel de impredictibilidad resultado de la creatividad del autor y su libre albedrío. Se utilizó la biblia por ser el texto traducido a más idiomas. La biblia tiene varias características importantes. Es un texto escrito por varios autores en diferentes momentos, algunas partes diferenciándose por más de un siglo. Esto apunta a una diversidad de estilos dentro del propio texto. Por otro lado, la biblia tiene un número grande de nombres propios que se mantienen, en muchos casos para diferentes idiomas [39].

Antes de la estimación de las magnitudes entrópicas, se tomaron de cada texto traducido una porción de texto común de $7.4{\times}10^4$ caracteres. Las magnitudes entrópicas se calcularon con el texto sin cambiar y con el texto aleatorizado a nivel de palabras para romper las correlaciones existentes fuera del nivel de una palabra. Se procesaron 37 idiomas.

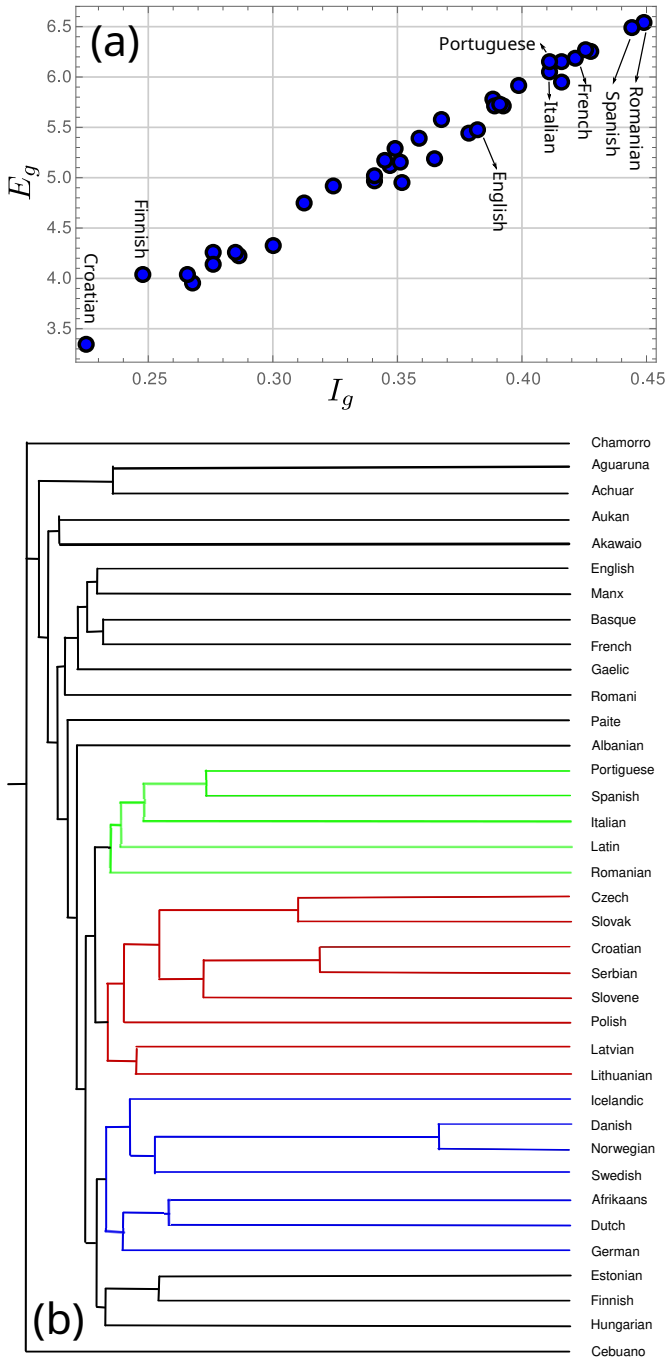

Figura 4. (a) Mapa de complejidad-entropía y (b) dendrograma basado en distancias informacionales entre traducciones bíblicas. Idiomas que pertenecen a una misma familia, aparecen agrupados corespondientemente. En verde los idiomas romances, en rojo los idiomas eslavos y en azul, los idiomas germánicos.

El mapa de complejidad-entropía se contruyó con h_g vs E_g:

$$h_g = h^r_{lz} - h_{lz}$$
$$E_g = E_{lz} - E^r_{lz}, \quad (12)$$

donde h^r_{lz} y E^r_{lz} es la densidad de entropía y el exceso de entropía estimados del texto aleatorizado. h_g es la ganancia de información o la cantidad de información necesaria para reorganizar las palabras hasta obtener nuevamente el texto original, E_g es el exceso de ganancia de entropía que implica reorganizar las palabras.

En la figura 4a el mapa de complejidad-entropía logra diferenciar distintos grupos de lenguajes. Por ejemplo aparecen agrupados los idiomas Español, Rumano, Francés, Portugués e Italiano que son lenguas romances, y en consecuencia tienen características gramaticales comunes. El cálculo nos indica que las lenguas romances tienen un alto valor de h_g, alrededor de 0.43 bit/caracter, y de E_g con un promedio de 6.2 bit/caracter. En el otro extremo del mapa las lenguas eslávas y nórdicas muestran una baja tasa de h_g y E_g indicando lenguas con menor nivel de coyundas gramaticales.

El cálculo de las distancias mutuas entre todos los idiomas, permitió calcular una matriz de distancia de donde un dendrograma fue generado (figura 4b). El dendrograma logra captar la relación entre distintos grupos lingüisticos con mayor claridad. Los idiomas de origen latino aparecen agrupados en una misma rama del árbol hasta el nivel de las hojas. Obsérvese como el español y el portugués, idiomas muy cercanos, aparecen como "hermanos" y a su vez ambos emparentados a un nivel más arriba al italiano. Los tres idiomas aparecen emparentados al latín. El rumano es un idioma romance derivado del latín pero algo más alejado del español, el portugués y el italiano. Este nivel de relación jerárquica entre los idiomas es destacable si consideramos las características señaladas del texto analizado.

Un análisis similar puede hacerse con las lenguas eslavas y las germánicas. En los tres casos el dendrograma indica su pertenencia a una familia común, y determinados detalles de su relación pueden ser señalados entre ellos. Por ejemplo, el afrikaner y el holandés tiene un origen común, siendo el primero una derivación del segundo, creado durante la dominación colonial de Sudáfrica. A su vez estos dos idiomas están emparentados cercanamente al alemán.

VI. CONCLUSIONES

En este trabajo se presentó la biblioteca *lzcomplexity* con las funciones para calcular magnitudes entrópicas sobre secuencias de símbolos. La implementación paralela del algoritmo de factorización permite procesar sistemas con longitudes al menos de 10^9 símbolos en un tiempo razonable. Esta longitud de secuencia es relevante en distintos escenarios, en particular en el análisis del código ADN de organismos vivos. Estos resultados serán publicados en otra contribución.

Para ejemplificar el uso del algoritmo se analizó la biblia traducida a una treintena de idiomas para determinar familias lingüisticas. El análisis entrópico permite clasificar los lenguajes en sus respectivos grupos lingüisticos.

VII. AGRADECIMIENTOS

Agradecemos los intercambios fructíferos con MC Ania Mesa Rodríguez que han permitido detectar errores en la implementación de los algoritmos. La data de la biblia fue curada por Ania Mesa.

Esta contribución se realizó en el marco del proyecto CARDENT (PNCB PN223LH010) para el análisis entrópico de data cardíaca y fue financiado parcialmente por el CITMA

REFERENCIAS

[1] C. Shannon, Bell Syst Tech J 30, 379 (1951).
[2] C. Shannon, Bell Syst Tech J 30, 50 (1951).
[3] R. Nagarayama, IEEE Trans Biomed Eng 49, 1371 (2002).
[4] A. Lesne, J. l. Blanc and L. Pezard, Phys Rev E 79, 046208 (2009).
[5] E. Estevez-Rams, R. Lora-Serrano, C. A. J. Nunes and B. Aragón-Fernández, Chaos 25, 123106 (2015).
[6] L. M. Alonso, Chaos DOI: 10.1063/1.4984800 (2017).
[7] H. Y. D. Sigaki, M. Perc and H. V. Ribeiro, PNAS 115, E8585 (2018).
[8] E. Estevez-Rams, A. Mesa-Rodriguez and D. Estevez-Moya, PLoS ONE 14, e0214863 (2019).
[9] M. Li and P. Vitányi, An Introduction to Kolmogorov Complexity and Its Applications. (Springer Verlag, 1993).
[10] A. Lempel and J. Ziv, IEEE Trans Inf Th IT-22, 75 (1976).
[11] J. Ziv, IEEE Trans Inf Th IT-24, 405 (1978).
[12] M. Li, X. Chen, X. Li, B. Ma and P. M. B. Vitanyi, IEEE Trans Inf Th 50, 3250 (2004).
[13] P. Li, BioMed Eng OnLine 18, 30 (2019).
[14] L. E. V. Silva, R. Fazan and J. A. Marin-Neto, Computer Methods and Programs in Biomedicine 197, 105718 (2020).
[15] M. W. Flood and B. Grimm, PLoS ONE 16, e0259448 (2021).
[16] D. Mayor, D. Panday, H. K. Kandel, T. Steffert and D. Banks, Entropy 23, 321 (2021).
[17] M. Crochemore, L. Ilie and W. F. Smyth, Data Compression Conference (DCC 2008), 482 (IEEE, 2008).
[18] P. Grassberger, Int J Theor Phys 25, 907 (1986).
[19] T. M. Cover and J. A. Thomas, Elements of information theory. Second edition (Wiley Interscience, New Jersey, 2006).
[20] J. P. Crutchfield and K. Young, Phys Rev Lett 63, 105 (1989).
[21] J. P. Crutchfield, Nature 8, 17 (2012).
[22] O. Melchert and A. K. Hartmann, Phys Rev E 91, 023306 (2015).
[23] I. Corporation, Intel® oneapi threading building blocks (onetbb) (2024).
[24] B. Stroustrup, The C++ Programming Language (Addison-Wesley, Reading, Mass, 2000), special ed edition.
[25] Welcome to Python.org (2025). URL `https://www.python.org/`.
[26] Pybind11 documentation (2025). URL `https://pybind11.readthedocs.io/en/stable/index.html`.
[27] D. K. Kim, J. S. Sim, H. Park and K. Park, J of Discrete Algorithms 3, 126 (2005).
[28] J. Kärkkäinen, P. Sanders and S. Burkhardt, J ACM 53, 918 (2006).
[29] M. Salson, T. Lecroq, M. Léonard and L. Mouchard, J of Discrete Algorithms 8, 241 (2010).
[30] G. Nong, S. Zhang and W. H. Chan, IEEE Trans Comput 60, 1471 (2011).
[31] U. Baier, 27th Annual Symposium on Combinatorial Pattern Matching, 12 (Dagstuhl Publishing, 2016).
[32] F. Kulla and P. Sanders, Parallel Computing 33, 605–612 (2007).
[33] J. Labeit, J. Shun and G. E. Blelloch, Journal of Discrete Algorithms 43, 2 (2017).
[34] J. Khan, T. Rubel, E. Molloy, L. Dhulipala and R. Patro, Algorithms for Molecular Biology 19, 16 (2024).
[35] W. D. Frazer and A. C. McKellar, J ACM 17, 496 (1970), ISSN 0004-5411, 1557-735X.
[36] G. Nong, S. Zhang and W. H. Chan, 2009 data compression conference, 193–202 (IEEE, 2009).
[37] Numba: A high performance python compiler (2025). URL `https://numba.pydata.org/`.
[38] M. A. Montemurro and D. H. Zanette, PLoS One 6, e19875 (2011).
[39] C. Christodouloupoulos and M. Steedman, Lang Resources and Evaluation 49, 375 (2015).

This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0, `https://creativecommons.org/licenses/by-nc/4.0`) license.