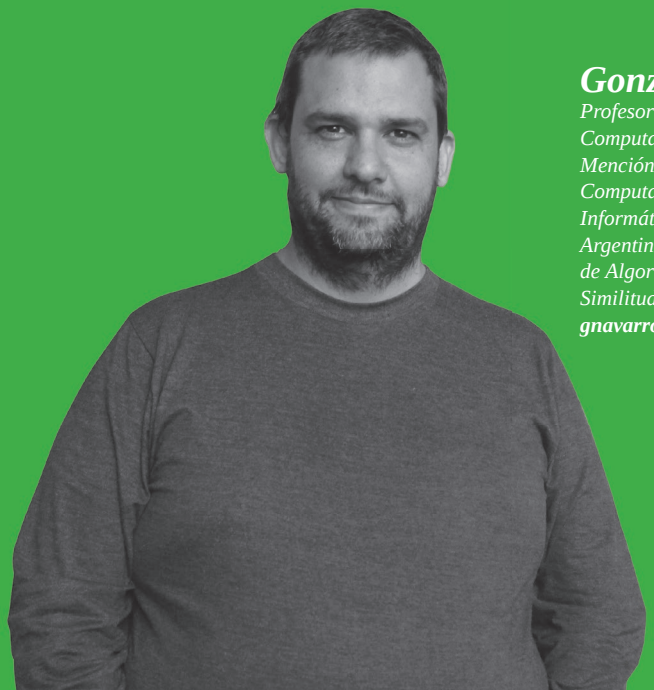
Teoría de la Información



Gonzalo Navarro

Profesor Titular Departamento de Ciencias de la Computación, Universidad de Chile. Doctor en Ciencias Mención Computación, Magíster en Ciencias mención Computación, Universidad de Chile; Licenciatura en Informática, Universidad Nacional de La Plata y ESLAI, Argentina. Líneas de investigación: Diseño y Análisis de Algoritmos, Bases de Datos Textuales, Búsqueda por Similitud, Compresión.

gnavarro@dcc.uchile.cl

La Teoría de la Información puede definirse como el estudio de la cantidad de información que contiene un mensaje, o que puede codificarse en un mensaje. De la mera definición se hace evidente su relevancia en áreas tan amplias de la Computación como el almacenamiento eficiente de la información, la transmisión eficiente de mensajes, la compresión, e incluso la resistencia a errores en el medio de transmisión o almacenamiento.

Nuestra experiencia en el mundo moderno sería bastante distinta si la compresión no existiera. Sería impensable almacenar cientos de fotografías, vídeos y música en nuestro computador. Una película de una hora a un modesto 800x600 a 20 cuadros por segundo, que es bastante baja calidad, necesitaría casi 100 GB de almacenamiento. Con una conexión de 20 Mbits por segundo sólo podría ver vídeo a 2 cuadros por segundo. Una memoria de 1 GB en su cámara de 5 megapíxeles se llenaría con menos de 70 fotos. La transmisión de TV y vídeo en línea, analógica o digitalmente, sería probablemente un sueño imposible. Incluso en el caso menos dramático de la información textual, toda su navegación en Internet sería unas cuatro veces más lenta, y sus discos y USBs rendirían a un cuarto de lo que pueden rendir.

Pero, ¿cómo medir cuánta información contiene un mensaje? Ni siquiera es algo sencillo de definir. Consideremos secuencias binarias para no enredarnos con otros problemas. Tomemos como ejemplo la siguiente secuencia:
00000000000000000000000000000000...

Esta secuencia contiene muy poca información. Una descripción corta es “puros ceros”. La siguiente es igualmente simple:
01010101010101010101010101010101...

y la siguiente algo más compleja, pero igualmente fácil de describir:
00101101110111101111101111110...

¿Se le ocurre cómo describirla? Podríamos explicar cómo generarla: poner un 0 y ningún 1, luego un 0 y un 1, luego un 0 y dos 1s, luego un 0 y tres 1s, etc. ¿Y la siguiente?
10110111111000010101000101100001...

Ésta parece completamente aleatoria. Pero realmente no lo es. Son los primeros bits de la expansión fraccionaria del número e = 2.718..., la base de los logaritmos naturales. Basta esta descripción para reproducir la secuencia, no importa cuán larga sea.

Pero no todas las descripciones son igualmente útiles. Para dar un ejemplo un poco esotérico, imagine que listáramos todas las sentencias en lógica de predicados (la que se usa para describir teoremas matemáticos), ordenadas por largo creciente, y alfabéticamente dentro de cada largo. Entonces podríamos definir una secuencia donde el i-ésimo dígito es 1 si la afirmación es un teorema y 0 si no lo es. Como existen conjeturas matemáticas que aún no se sabe si son verdaderas o falsas, este tipo de descripción, si bien es clara y formal, no nos permite saber cuál es la secuencia (¡más bien, si tuviéramos la secuencia, tendríamos resueltos todos los problemas matemáticos!).

¿Cómo podemos definir un concepto de *descripción útil*? ¡Aquí entra la computación en juego! Andrey Kolmogorov (1903-1987) fue un matemático ruso que consideró este problema y propuso una medida de “complejidad” de las secuencias que se considera hoy en día la definición más clara, correcta y universal en término de descripciones útiles. *La Complejidad de Kolmogorov de una secuencia es la cantidad de bits que se necesita para representar un programa de computador que genere la secuencia*[1].

Esta definición restringe las descripciones a aspectos *computables*, es decir, que la secuencia se pueda producir mecánicamente a partir de la descripción. Esto

[1] Si bien la definición parece depender del lenguaje en que escribimos el programa y cómo lo codificamos en bits, este hecho se hace irrelevante cuando consideramos secuencias suficientemente largas, pues el programa puede incluir un decodificador y un intérprete de otro lenguaje, y éstos son de largo fijo.

incluye nuestros primeros ejemplos pero excluye el último, donde la descripción no permitía obtener la secuencia. De hecho, todos nuestros primeros ejemplos tienen una complejidad de Kolmogorov muy baja: se pueden hacer programas muy cortos para generar secuencias muy largas. Más aún, considere un texto de largo n que se pueda comprimir a m bits usando un compresor cualquiera. Entonces su complejidad de Kolmogorov no es mayor a $c+m$, donde c es el largo del programa descompresor. Es decir, esta complejidad abarca cualquier técnica de compresión conocida o por conocer.

Por otro lado, si no existe ningún programa de largo menor a n que genere una cierta secuencia de largo n, entonces su complejidad de Kolmogorov será a lo menos n y diremos que la secuencia es completamente aleatoria. ¿Existen estas secuencias completamente aleatorias? Sí, siempre existe al menos una, pues existen 2^n secuencias binarias de largo n, pero el conjunto de todos los programas posibles de largo menor a n (codificados en binario) suma 2^n-1 . En realidad, por cada programa de largo m que emite una secuencia de largo n, hay otros 2^{n-m} programas invalidados, por lo que en realidad la gran mayoría de las secuencias son *completamente aleatorias*. ¿Y cuán larga puede ser la complejidad de Kolmogorov de una secuencia? No tan mala: basta un programa ligeramente más largo que n, que diga "Imprimir 01001100111...."

Seguramente ha escuchado, para explicar la Teoría de la Evolución, que si sentara a un mono frente a un teclado por suficiente tiempo, éste terminaría escribiendo las obras completas de Shakespeare. Digamos que éstas tienen un largo n en bits. Entonces el mono requeriría en promedio 2^n intentos para dar con el texto correcto. Como existen compresores capaces de comprimir estas obras a un cuarto de su tamaño original, la complejidad de Kolmogorov de las obras de Shakespeare es a lo más $n/4$. Por ello, si sentáramos al mono frente a un computador a escribir programas, daría mucho antes que el que usa la máquina de escribir, en unos $2^{n/4}$ intentos, con un *programa* que generara estas obras como output. Es decir, un computador funciona como un dispositivo que aumenta las probabilidades de producir secuencias interesantes. Volveré pronto sobre este importante punto.

La complejidad de Kolmogorov ha sido sumamente útil en muchos aspectos, por ejemplo para calcular cuánto se tienen que parecer dos secuencias genéticas para determinar si existe entre ellas alguna relación o no. Y debería conducirnos al *compresor universal* óptimo, que nos entregara la descripción más corta posible para cualquier secuencia. Sin embargo, esta medida tiene una limitante muy grave: no es *computable*. Es decir, dada una secuencia, ningún programa de computador puede determinar cuál es su complejidad de Kolmogorov[2]. Lamentablemente necesitamos utilizar un modelo menos general para poder medir la cantidad de información y diseñar un compresor acorde a ella.

Claude Shannon (1916-2001) fue un matemático estadounidense, considerado el padre de la Teoría de la Información. Shannon atacó el problema desde otro punto de vista: consideremos una fuente infinita de símbolos sobre un alfabeto de tamaño s. ¿Cuántos bits necesitamos para codificar cada símbolo? Bien, si la fuente emite los símbolos en forma equiprobable, necesitaremos lg(s) bits, donde lg denota el logaritmo en base 2. Es decir, si la fuente tiene cuatro símbolos, digamos A, B, C y D, necesitaremos dos bits para codificar cada símbolo (por ejemplo podríamos codificar A como 00, B como 01, C como 10, y D como 11).

Pero ¿qué ocurre si los símbolos no son equiprobables? Digamos que la mitad de las veces se emite A, la cuarta parte de las veces se emite B, un octavo de las veces se emite C y otro octavo se emite D. Entonces podríamos usar el siguiente código: 0 denota A, 10 denota B, 110 denota C y 111 denota D. Como ninguno de los cuatro códigos es prefijo de otro, podemos concatenar los códigos en una secuencia y no tendremos problemas para decodificarlos. Haciendo las cuentas, resulta que un mensaje de largo n con estas frecuencias requerirá $1.75n$ bits, en vez de los $2n$ que necesitaríamos si los codificáramos con 2 bits por símbolo.

Shannon demostró que lo óptimo es utilizar $\lg(\frac{1}{p})$ bits para codificar un símbolo que aparece con frecuencia relativa p, lo que da lugar a su definición de *entropía de una fuente*. La entropía de una fuente que emite símbolos con probabilidades $p_1, p_2, \ldots, p_s$ es $H=\sum_{i=1}^{s} p_i \lg(\frac{1}{p_i})$.

Dicho de otro modo, Shannon demostró que, si una fuente infinita genera símbolos con esas frecuencias (y no tiene otras regularidades aparte de ésa), entonces es imposible codificar un mensaje de esa fuente utilizando menos de H bits por símbolo. Con el tiempo aparecieron codificadores (como la codificación aritmética) que se acercan mucho a ese mínimo teóri-

[2] Tal vez se le ocurra que podría considerar todos los programas posibles, de largos crecientes, hasta dar con el primero que produzca la secuencia que le interesa. Desengáñese: un programa puede ejecutar durante un tiempo arbitrariamente largo antes de emitir su output, y tampoco es computable saber si alguna vez emitirá algo.

co: nH+2 bits para un mensaje de largo n con entropía H.

La medida de entropía de Shannon no es tan universal como la de Kolmogorov. Sólo se aplica cuando la frecuencia es la única regularidad que tiene una fuente. Por ejemplo, la secuencia de la expansión del número e se ve como completamente aleatoria en este sentido, si bien su complejidad de Kolmogorov es muy baja. En cambio, la complejidad de Kolmogorov de una secuencia que tiene entropía de Shannon H es muy cercana a Hn: como ya mencionamos, basta comprimir la secuencia con un codificador aritmético, cuyo programa mida c bits, y el programa sería el descompresor más la secuencia comprimida, que ocupa a lo más c+Hn+2 bits. A medida que n crece, el extra de c+2 bits se hace menos relevante.

Es posible extender la entropía de Shannon a casos más complejos, como aquellos en los que la fuente tiene una cierta cantidad de memoria finita. Es decir, la probabilidad de los símbolos no es fija sino que depende de los últimos símbolos emitidos. Esto modela muy bien los lenguajes humanos, donde por ejemplo la probabilidad de ver una consonante en español es muy baja si las dos letras previas ya fueron consonantes. Los compresores que se basan en modelar los textos como si vinieran de este tipo de fuente se llaman compresores estadísticos. Un ejemplo son los compresores tipo PPM. Instale, por ejemplo, ppmdi en su computador, y verá tasas de compresión de 4-5 veces en sus textos.

Otro tipo de medida de entropía, menos conocida por ser menos elegante, pero muy utilizada en los compresores, fue definida implícitamente por Abraham Lempel y Jacob Ziv en 1976. Se basa en el hecho de que un texto donde muchos pasajes son copias de pasajes previos es fácilmente compresible. Lempel y Ziv dan un algoritmo que va procesando el texto, y en cada paso lee la mayor secuencia posible que ya se haya visto antes. El número de "frases" así formadas es una medida de entropía. Mientras que se puede probar que, en las secuencias que consideró Shannon, la entropía de Lempel-Ziv converge (si bien algo lentamente) a la de Shannon, existen casos en que la entropía de Lempel-Ziv es mucho más baja, por ejemplo cuando la secuencia es muy repetitiva (imagínese el conjunto de versiones de un artículo, por ejemplo, o de un sistema de software). Los compresores tipo "zip", ampliamente disponibles en Linux y Windows, se basan en variantes de este tipo de compresión.

¿Existe un compresor que funcione siempre? ¿Todas las secuencias son compresibles? Tal como ocurría con la complejidad de Kolmogorov, y con más razón en estos modelos más restrictivos de compresibilidad, la respuesta es un rotundo no. De hecho, la gran mayoría de las secuencias son incompresibles. Por cada secuencia compresible que hay, muchas otras no lo son.

¿Por qué entonces en la vida práctica los compresores parecen funcionar siempre? Porque los aplicamos siempre sobre cierto tipo de secuencias. Más aún, a falta de un compresor universal de Kolmogorov, utilizamos compresores adecuados al tipo de compresibilidad que esperamos ver. Los compresores estadísticos y tipo Lempel-Ziv son adecuados a secuencias tipeadas por humanos. Las imágenes, audio y vídeo necesitan otro tipo de compresor[3]. Los textos procesables automáticamente, como XML o programas, pueden comprimirse utilizando gramáticas de lenguajes formales. Las secuencias biológicas necesitan compresores especializados para detectar copias de bloques complementadas e invertidas, y así.

Sin embargo, todo esto es posible solamente porque los *textos, audios, imágenes y vídeos que nos interesan tienen una complejidad de Kolmogorov muy baja*. Si los humanos encontráramos que el ruido blanco de la radio es apasionante, las estaciones de radio tendrían serios problemas para distribuir esa "música" en forma comprimida. Es decir, *lo que es interesante para nuestro cerebro resulta ser "interesante" algorítmicamente, pues es compresible*. Esto es una indicación muy fuerte de que el procesamiento de nuestro cerebro es esencialmente algorítmico, en el sentido de lo que entendemos por algoritmo. O dicho de otra manera, nuestra noción de algoritmo parece capturar correctamente lo que hace nuestro cerebro.

Si tomamos otra fuente de secuencias, como el ADN, por ejemplo, encontramos que la situación es muy distinta. El ADN bacteriano, en particular, es muy difícil de comprimir. Y por una buena razón: en una bacteria no sobra espacio para moléculas muy largas, de modo que la funcionalidad de la bacteria debe ser expresada en una secuencia lo más corta posible. La evolución ha hecho el resto, favoreciendo a las bacterias capaces de codificar

[3] Estos compresores en general admiten una cierta diferencia entre lo que comprimieron y lo que reproducen, que se busca que sea mínima para el receptor. Esto introduce otra dimensión que no abordo en este artículo, que es el compromiso entre tasa de compresión y distorsión permitida al recuperar el mensaje. La entropía de Shannon también regula hasta cuánta compresión se puede obtener por el precio de cuánta distorsión. En mensajes textuales normalmente no se admite ninguna distorsión.

comportamiento más sofisticado en el mismo espacio. El resultado es un "programa" muy corto que hace mucho; algo cercano al ideal de Kolmogorov.

ESTRUCTURAS DE DATOS COMPRIMIDAS

Antes de terminar este artículo quisiera hablar sobre una rama reciente derivada de la compresión, que se llama *estructuras de datos comprimidas*. Hablamos de la utilidad de la compresión para ahorrar espacio y tiempo de comunicación. Esta nueva área, en cambio, se relaciona con el hecho de que las memorias más rápidas son mucho más pequeñas que las memorias más lentas. Las memorias cachés son decenas de veces más rápidas que la RAM, y ésta es miles a millones de veces más rápida que el disco. Un programa que pueda funcionar en una memoria menor gozará de accesos a memoria mucho más rápidos, y esto influenciará su tiempo de ejecución. Y si, en cambio, estamos dispuestos a costearnos un sistema distribuido con tantas memorias RAM como necesitemos (como hacen los grandes buscadores de Internet), el utilizar menos espacio redundará en comprar menos máquinas, reducir el tiempo de comunicación entre ellas, y ahorrar en la energía que consumen. Hoy en día la energía es una parte muy alta de la factura que pagan los grandes centros de datos.

Para lograr esto necesitamos más que compresión. No basta comprimir los datos si es que para procesarlos vamos a necesitar descomprimirlos primero. Necesitamos *poder operar los datos en forma comprimida*. Y no sólo los datos, sino las estructuras de datos que usamos para organizarlos. Si tenemos números en un árbol binario, necesitamos poder representar el árbol binario en forma comprimida de modo que nos permita buscar, y tal vez insertar y borrar elementos, *sin descomprimirlo*.

Éste es el campo de estudio de las estructuras de datos comprimidas, que combina la Teoría de la Información con los Algoritmos y Estructuras de Datos, y ofrece desafíos apasionantes. Daré algunos ejemplos para ilustrarlo. Considere un árbol general de n nodos. Una implementación normal utilizará, como mínimo, n punteros para representarlo en memoria. Como un puntero tiene que por lo menos distinguir entre los n nodos del árbol, estaremos utilizando $n \lg n$ bits de memoria. Pero ¿es ésta una representación eficiente? La Teoría de la Información nos dice que no. El total de árboles distintos de n nodos es cercano a $\frac{4^n}{n^{3/2}}$.[4]

Esto significa que, incluso considerando a todos los árboles igualmente probables, bastarán $\lg(\frac{4^n}{n^{3/2}}) \leq 2n$ bits para representar cualquier árbol general. ¡Esto es mucho, mucho menos, que los $n \lg n$ bits que se suelen usar! Para un árbol de un millón de nodos, es ¡diez veces menos!

¿Se atreve a diseñar una representación comprimida de árboles que use $2n$ bits? No es tan difícil en realidad. Recorra el árbol en profundidad, recursivamente. Cada vez que llegue a un nodo por primera vez, abra un paréntesis. Cada vez que lo abandone, cierre un paréntesis. La secuencia resultante es de largo $2n$, y como se compone de paréntesis abiertos y cerrados, se puede representar usando $2n$ bits. Viendo la secuencia, puede fácilmente reconstruir el árbol. ¡Felicitaciones, ha diseñado un excelente compresor especializado en árboles! Pero aún no tiene una estructura de datos comprimida para árboles. Para esto, debería ser capaz de navegar en esta secuencia de paréntesis, por ejemplo encontrando el padre de un nodo (podemos identificar un nodo, por ejemplo, con la posición de su paréntesis que abre), bajando al i-ésimo hijo, calculando el tamaño de un subárbol, la profundidad o la altura de un nodo, el ancestro común más bajo entre dos nodos, y decenas de otras operaciones útiles. Hoy en día se sabe cómo realizar todas estas operaciones en tiempo constante, y utilizando esencialmente los $2n$ bits. Existen implementaciones que usan unos $2.3n$ bits y resuelven todas esas consultas en microsegundos.

Un segundo ejemplo son los grafos Web, es decir subconjuntos de la Web, donde los nodos son páginas y las aristas son links. Estos se utilizan con múltiples propósitos, como calcular relevancia de páginas, encontrar autoridades, detectar fábricas de spam, distinguir comunidades, etc. Para representar un grafo dirigido de n nodos y e aristas en forma clásica, por ejemplo como listas de adyacencia, se necesitan unos $n \lg e + e \lg n$ bits. Esta vez la Teoría de la Información no nos ayuda mucho: la cantidad de grafos distintos es tan grande que el número de bits que se necesita para representarlos no es muy distinto del que se usa con una lista de adyacencia. Pero tal como ocurre con las imágenes, los grafos Web no son grafos cualquiera. Tienen algunas características que los hacen "interesantes" y algorítmicamente compresibles. Existen hoy en día estructuras de datos comprimidas para grafos Web que usan poco más de un bit por arista (es decir, veinte a treinta veces menos que una representa-

[4] Para los interesados, el valor exacto está dado por los llamados Números de Catalán.

ción clásica, para grafos grandes) y permiten navegarlos eficientemente. Esto permite analizar subgrafos de la Web mucho mayores en memoria principal.

Un último ejemplo es el genoma humano, que contiene unas tres mil millones de bases (A, C, G, T). Si bien éste se puede representar en unos modestos 700 MB, en bioinformática se necesita mucho más que representarlos. Se necesita realizar complejos análisis para detectar repeticiones, similitudes, y muchas otras regularidades. Para ello se usa frecuentemente una estructura de datos conocida como árbol de sufijos. Si bien esta estructura permite responder eficientemente muchas preguntas complejas, una representación clásica requiere unos $20n$ bytes, ¡con lo cual necesitamos una memoria de 60 GB para representar un genoma! Sin embargo, en términos de Teoría de la Información, el árbol de sufijos es completamente redundante, pues no contiene nada que no se pueda derivar del texto mismo, y por ello debería ser posible, en principio, representarlo en tan poco espacio como el texto mismo, unos $2n$ bits. Hoy en día existen implementaciones de árboles de sufijos comprimidos que, si bien no se acercan tanto a este ideal, sí pueden almacenar un genoma en menos de 2 GB y simular todas las operaciones del árbol de sufijos en microsegundos.

Pero esto es un solo genoma. Hoy en día las grandes compañías están secuenciando miles de genomas por día. Pronto tendremos bases de datos bioinformáticas con miles y hasta millones de genomas. Esto, multiplicado por tres mil millones de bases, representa un desafío más allá de todas nuestras capacidades. Incluso pensando en 700 MB por genoma nos lleva a números imposibles. ¿Cómo haremos frente a estos desafíos? La Teoría de la Información nos da una pista. Los genomas de una misma especie se parecen mucho entre sí, en más de 99.9% en muchos casos. Es decir, una colección de genomas humanos se puede ver como una colección altamente repetitiva, donde las técnicas de Lempel y Ziv deberían tener mucho que ofrecernos. Y es así: estos compresores pueden obtener tasas de unas pocas centésimas de bit por símbolo cuando los aplicamos a este tipo de colecciones. Pero otra cosa son las estructuras de datos comprimidas. El estudio de estructuras como árboles de sufijos para colecciones altamente repetitivas está en sus comienzos, pero creo que es la clave para los desafíos que nos esperan en este campo.

Y no sólo la bioinformática promete inundarnos de datos en el siglo que comienza. Los datos astronómicos, los meteorológicos y ambientales, los de comportamiento social (clicks, tweets, compras online), los científicos, los geográficos, los multimediales, los de realidad aumentada, etc., se han convertido en esenciales para la ciencia, la tecnología, y los aspectos más básicos de la vida diaria, y no han hecho más que empezar. Se incrementarán rápidamente con la computación ubicua y con las interfaces directas con los órganos sensoriales o el cerebro. Esta avalancha de datos representará muy en breve desafíos gigantescos de almacenamiento y procesamiento. Estoy convencido de que la Teoría de la Información y la Algorítmica serán herramientas esenciales para enfrentar esos desafíos y ser capaces de dar el salto a una sociedad mucho más compleja que la de hoy. BITS