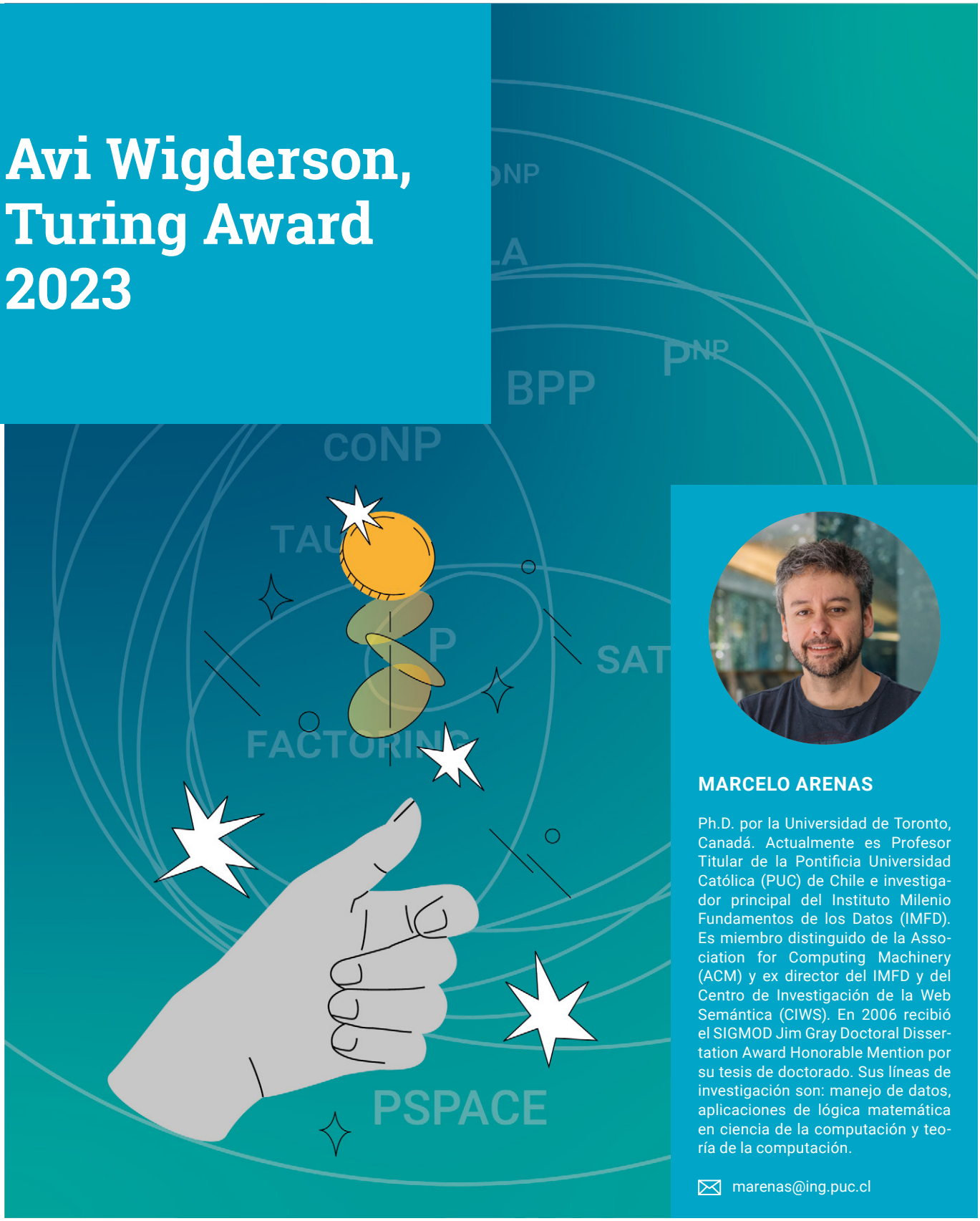
Avi Wigderson, Turing Award 2023



MARCELO ARENAS

Ph.D. por la Universidad de Toronto, Canadá. Actualmente es Profesor Titular de la Pontificia Universidad Católica (PUC) de Chile e investigador principal del Instituto Milenio Fundamentos de los Datos (IMFD). Es miembro distinguido de la Association for Computing Machinery (ACM) y ex director del IMFD y del Centro de Investigación de la Web Semántica (CIWS). En 2006 recibió el SIGMOD Jim Gray Doctoral Dissertation Award Honorable Mention por su tesis de doctorado. Sus líneas de investigación son: manejo de datos, aplicaciones de lógica matemática en ciencia de la computación y teoría de la computación.

marenas@ing.puc.cl

RESUMEN. La influencia de Avi Wigderson en el área de complejidad computacional es innegable; cualquier persona que trabaje en este campo o utilice sus herramientas en otras áreas, inevitablemente encontrará alguno de sus resultados o las consecuencias de su trabajo. A lo largo de su impresionante carrera académica, Avi Wigderson ha moldeado este campo con una serie de resultados variados, originales y profundos. Este artículo no pretende detallar exhaustivamente todos los logros de Avi Wigderson, sino más bien destacar dos de sus contribuciones que sobresalen por su originalidad e impacto.

¿Qué es la complejidad computacional?

El problema fundamental de la ciencia de la computación es diseñar procedimientos, o algoritmos, eficientes que permitan resolver diversos tipos de problemas. Esta noción de eficiencia generalmente se mide en términos del tiempo que necesita un algoritmo para resolver un problema, aunque también puede medirse en términos de otros recursos relevantes, como el espacio.

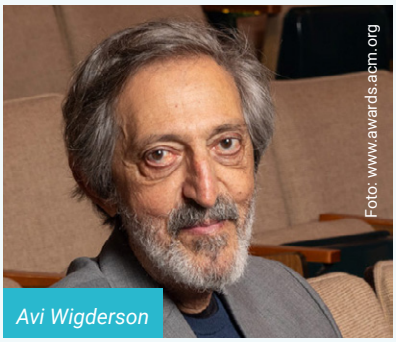
Acompañando esta necesidad de desarrollar técnicas para la creación de algoritmos eficientes, surge también el problema de desarrollar herramientas que permitan demostrar que un problema *no puede* ser resuelto con ciertos recursos. Estas técnicas son fundamentales, ya que evitan esfuerzos infructuosos en la búsqueda de soluciones imposibles.

Como ejemplo, consideremos el problema de ordenar de menor a mayor una lista de n números naturales. Al reflexionar sobre este problema, se puede notar que es sencillo desarrollar un algoritmo que ordene la lista realizando alrededor de n^2 comparaciones de números en la lista. Además, existen algoritmos más eficientes que pueden resolver este problema realizando alrededor de $n \times log_2(n)$ comparaciones. ¿Pero es posible resolver este problema con un algoritmo aún más eficiente, que sólo realice alrededor de n comparaciones? Algunas herramientas desarrolladas en la ciencia de la computación permiten demostrar que un algoritmo para ordenar una lista con n elementos no puede realizar menos de $n \times log_2(n)$ comparaciones. Esto indica inmediatamente que no es posible construir el algoritmo lineal planteado en la pregunta.

El área de complejidad computacional estudia la complejidad inherente de un problema computacional, lo cual se traduce en entender la cantidad de recursos computacionales, como tiempo y espacio, necesarios para resolverlo. En particular, en esta área se desarrollan las técnicas mencionadas anteriormente para demostrar que un problema no puede ser resuelto con ciertos recursos. Es aquí donde surge una de las tareas más fundamentales de la complejidad computacional: demostrar que existen problemas que no pueden ser resueltos de manera eficiente.

Es importante detenernos aquí en la noción de eficiencia. Un algoritmo se considera eficiente si la cantidad de pasos que necesita para resolver un problema está acotada superiormente por un polinomio fijo. Por ejemplo, todos los algoritmos de ordenación mencionados en el párrafo anterior son considerados eficientes ya que realizan n^2 comparaciones. Por otro lado, un algoritmo que realiza 2^n operaciones para resolver un problema, es decir, tiempo exponencial, no se considera eficiente ya que la función 2^n no está acotada superiormente por ningún polinomio fijo. Entonces, la pregunta fundamental de la complejidad computacional se traduce en demostrar que existen problemas que no pueden ser resueltos por un algoritmo de tiempo polinomial.

Ganador del Premio Turing 2023



Avi Wigderson

Por supuesto, el lector podría estar pensando que un algoritmo de tiempo polinomial no es necesariamente eficiente en la práctica; por ejemplo, un algoritmo que realiza n^{100} operaciones no puede ser utilizado ni siquiera para una entrada con 10 elementos ($n = 10$). Sin embargo, lo que debe considerarse aquí es que el objetivo es demostrar que ciertos problemas no pueden ser resueltos por tales algoritmos. Es decir, son problemas tan difíciles que ni siquiera permitiendo algoritmos con tiempos de n^{100}, n^{1000} o n^{10000} podrán ser resueltos por ellos.

¿Qué tipos de problemas computacionales son tan difíciles? Existen muchas posibilidades, pero nos concentraremos aquí en la famosa clase de problemas NP, de la cual el lector puede haber oído en algún momento. Los problemas de esta clase se caracterizan porque es difícil construir una solución para ellos, pero, dada una posible solución, es fácil verificar que efectivamente lo es.

Como ejemplo de un problema en esta clase, consideremos el problema de pintar un mapa con tres colores, de manera tal que dos países limítrofes tengan colores distintos. Llamaremos a este problema 3-COL-MAPA. Seguramente el lector ha escuchado que un mapa siempre puede ser pintado con cuatro colores, lo cual es conocido como el Teorema de los Cuatro Colores. Por el contrario, no todos los mapas pueden ser pintados con tres colores. Como

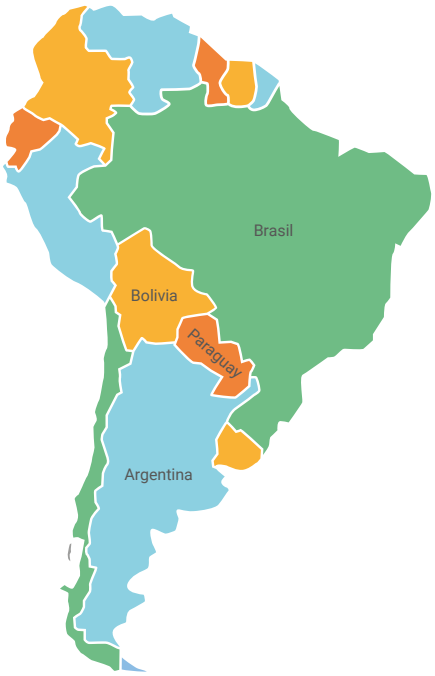


Figura 1. Mapa de Sudamérica pintado con cuatro colores.

ejemplo de esto, considere el mapa de Sudamérica, el cual, como se muestra en la Figura 1, puede ser pintado con cuatro colores. Sin embargo, este mapa no puede ser pintado con tres colores, ya que Argentina, Brasil y Bolivia necesitan colores distintos por ser mutuamente limítrofes, y todos ellos tienen a Paraguay como país limítrofe.

¿Cuál es la complejidad de 3-COL-MAPA? Este es uno de los problemas de la clase NP que son considerados difíciles. En particular, para este problema sólo se conocen algoritmos que, esencialmente, deben probar todas las coloraciones posibles para verificar si un mapa se puede pintar con tres colores. Si el mapa tiene n países, entonces existen 3^n tales coloraciones, por lo que estos algoritmos funcionan en tiempo exponencial. Por otro lado, es simple verificar si una posible solución es correcta: dada una posible coloración del mapa, sólo debemos comprobar que cada par de países limítrofes tiene colores distintos para asegurarnos de que la coloración es correcta. Esta verificación puede realizarse en tiempo polinomial.

Defina P como la clase de problemas que pueden ser solucionados en tiempo polinomial. El problema abierto más importante en complejidad computacional es demostrar que $P \neq NP$, lo cual consiste en demostrar que no existe un algoritmo de tiempo polinomial para el problema 3-COL-MAPA. Por supuesto, el lector podría preguntarse por qué pintar un mapa con tres colores es un problema importante. La respuesta a esta pregunta es simple: 3-COL-MAPA es equivalente a un gran número de otros problemas fundamentales que tienen numerosas aplicaciones en la práctica, como la programación lineal entera, la satisfacibilidad de fórmulas de lógica proposicional y el problema del agente viajero [1]. En particular, esta equivalencia implica que si 3-COL-MAPA no puede ser resuelto en tiempo polinomial, entonces ninguno de los problemas anteriores puede ser resuelto en tiempo polinomial.

Demostrar que $P \neq NP$ tiene una connotación negativa, ya que nos indica que no es posible construir algoritmos eficientes para un gran número de problemas que son importantes en la práctica. Sin embargo, y sorprendentemente, Avi Wigderson también nos muestra que este resultado, y el demostrar que ciertos problemas no pueden ser resueltos en tiempo polinomial, tiene consecuencias positivas para el diseño de algoritmos eficientes. A continuación, presentamos dos de sus resultados más fundamentales que van precisamente en esa dirección.

Estas técnicas [para demostrar que un problema no puede ser resuelto con ciertos recursos] son fundamentales, ya que evitan esfuerzos infructuosos en la búsqueda de soluciones imposibles.

Eliminando el uso de aleatorización en un algoritmo

La búsqueda de algoritmos eficientes ha llevado al desarrollo de una variedad de técnicas. En particular, el uso de soluciones aproximadas o con una probabilidad de error ha permitido el desarrollo de algoritmos en casos para los cuales no se conocen algoritmos eficientes tradicionales (exactos y no aleatorizados). En esta sección, nos concentraremos en el uso de la aleatorización para el desarrollo de algoritmos eficientes, y en la posibilidad de que el uso de la aleatorización sea eliminado, como una forma de construir algoritmos eficientes sin probabilidad de error.

Recuerde que un número natural p es primo si sus únicos divisores son 1 y p. Por ejemplo, 7 es primo, y 8 no lo es (ya que es divisible, por ejemplo, por 2). El problema de verificar si un número es primo ha sido estudiado por más de dos mil años; de hecho, el método de la Criba de Eratóstenes para construir todos los números primos hasta un número dado fue desarrollado en el siglo III a.C. Sin embargo, los primeros algoritmos eficientes para resolver este problema fueron desarrollados en los años setenta, siendo la clave para esto el uso de números aleatorios y el permitir que la respuesta tenga una probabilidad de error. En particular, el algoritmo de Solovay-Strassen [2], propuesto en 1977, puede verificar si un número p es primo realizando un número polinomial de operaciones, y su respuesta tiene una probabilidad de error tanto en el caso de que p sea

Aquí llegamos a otra pregunta fundamental en complejidad computacional: ¿es posible eliminar el uso de aleatorización de un algoritmo?

primo como en el caso de que sea un número compuesto (no primo). Vale decir, si p es un número primo, existe una probabilidad de que el algoritmo diga que es compuesto, y si p es un número compuesto, existe una probabilidad de que el algoritmo diga que es primo. En el algoritmo de Solovay-Strassen se puede controlar la probabilidad de error utilizando un parámetro k; en particular, la probabilidad de error está acotada superiormente por 2^{-k}. Por ejemplo, si $k = 100$, entonces la probabilidad de que el algoritmo se equivoque es a lo más $2^{-100} \approx 7.8 \times 10^{-31}$, un número pequeñísimo.

El algoritmo de Solovay-Strassen, y otros tests de primalidad como el algoritmo de Miller-Rabin [3,4], cumplen todas las condiciones para ser usados en la práctica: funcionan en tiempo polinomial y tienen una probabilidad de error bajísima. Sin embargo, como estos algoritmos son la base de la criptografía moderna y la comunicación segura en Internet, y son ampliamente utilizados cada día, nos gustaría poder eliminar la probabilidad de error en sus respuestas. Y aquí llegamos a otra pregunta fundamental en complejidad computacional: ¿es posible eliminar el uso de aleatorización de un algoritmo? En el caso del problema de verificar si un número es primo, el uso de aleatoriedad fue finalmente eliminado en el algoritmo AKS [5] propuesto en 2002, el cual no tiene una probabilidad de error asociada. La pregunta entonces es si es siempre posible hacer esto.

Defina BPP como la clase de problemas que pueden ser solucionados por un algoritmo aleatorizado que tiene una probabilidad de error acotada por 1/4.[1] La posibilidad de eliminar la aleatoriedad de un algoritmo se traduce en demostrar que P = BPP, donde, como definimos antes, P es la clase de problemas para los cuales existen algoritmos polinomiales usuales (no aleatorizados) que los solucionan. Y aún más, la pregunta parece ser: ¿qué tipo de técnicas y resultados podríamos usar para lograr eliminar la probabilidad de error de un algoritmo aleatorizado?

De manera sorprendente, Avi Wigderson nos muestra que podemos dar una respuesta positiva a esta pregunta demostrando que existen problemas difíciles [6]. La idea de esta solución es la siguiente. Un generador pseudoaleatorio es un algoritmo que, a partir de un número inicial, llamado la semilla, genera una secuencia de números que no pueden ser distinguidos de manera eficiente de una secuencia de números verdaderamente aleatorios. Si tenemos tal generador, podemos eliminar la aleatoriedad de un algoritmo A simplemente tomando todos los valores posibles para la semilla, generando a partir de cada una de ellas una secuencia de números pseudoaleatorios y ejecutando el algoritmo A sobre cada una de las secuencias. En cada una de ellas vamos a obtener una respuesta de A, digamos sí o no, y a partir de ellas construimos la respuesta final del algoritmo B sin aleatoriedad, considerando la mayoría. Por ejemplo, si fue necesario ejecutar A sobre 128 secuencias de números pseudoaleatorios y en 50 casos la respuesta de A fue sí, entonces la respuesta del algoritmo B será no, ya que en la mayor parte de las secuencias la respuesta de A fue no. El correcto funcionamiento del algoritmo B está asegurado por la probabilidad de error de a lo más 1/4 del algoritmo A, la cual nos asegura que al tomar mayoría vamos a obtener la respuesta correcta. Lo que nos queda por responder aquí es cómo se puede construir un generador pseudoaleatorio con las propiedades mencionadas, y la respuesta de Avi Wigderson es que la existencia de un problema que puede ser solucionado en tiempo exponencial y que cumple ciertas condiciones de dificultad asegura la existencia de tal generador pseudoaleatorio [6].

La creación de pruebas de conocimiento cero

Suponga que está utilizando un sistema que requiere de una clave para poder identificarlo. ¿Es posible convencer a este sistema de que usted conoce la clave sin entregar información sobre ella? Esto es lo que se conoce como una prueba de conocimiento cero [7], una demostración de que usted conoce un secreto sin revelar información sobre el secreto.

Las demostraciones de conocimiento cero son muy útiles en la práctica, y por eso han sido ampliamente desarrolladas en el área de criptografía y seguridad computacional. Sólo piense en su interacción con un cajero automático. Cada vez que saca dinero de uno de ellos, debe ingresar su clave, lo cual es riesgoso, ya que alguien podría verla mientras la ingresa o usar un dispositivo para robar claves instalado en el cajero. En cambio, si se utiliza un protocolo de conocimiento cero, el cajero simplemente le pedirá que demuestre que conoce la clave (ver Figura 2). Usted debe entonces proporcionar esta demostración, que no necesita ocultar ya que no revela información sobre la clave. De hecho, si un ladrón tiene acceso a esta demostración, no podrá usarla para hacerse pasar por usted, ya que en este protocolo se incluirán la ubicación, fecha y hora (hasta los

1 El valor 1/4 en la definición de BPP es arbitrario y, de hecho, se puede hacer arbitrariamente más pequeño ejecutando el algoritmo varias veces y tomando como respuesta la mayoría entre los resultados de estas ejecuciones.

Figura 2. Utilizando una demostración de conocimiento cero en un cajero automático (imagen generada por ChatGPT).

segundos) de la transacción. Es decir, usted entrega una demostración de que conoce la clave que además incluye la ubicación, fecha y hora. Si el ladrón intenta reutilizar esta demostración, no le servirá, ya que no podrá replicar la misma ubicación, fecha y hora.

¿Cómo podemos demostrar que conocemos un secreto sin revelar información sobre él? Aquí vamos a ver otro de los resultados fundamentales de Avi Wigderson, quien en [8] demuestra que cada problema en NP admite una prueba de conocimiento cero, utilizando nuevamente para conseguir este resultado positivo la existencia de problemas difíciles. En términos de los problemas considerados en este artículo, esto significa que es posible convencer a alguien de que conozco una forma de pintar un mapa con tres colores, pero sin revelar información sobre esta coloración. A continuación, presentamos la demostración de conocimiento cero para este problema propuesta por Avi Wigderson en [8].[2]

Suponga que tiene una función $f(u, v)$ que recibe como primer parámetro un número $u \in \{1,2,3\}$ y como segundo parámetro una palabra v de símbolos 0 y 1, lo cual denotamos como $v \in \{0,1\}^*$. Además esta función satisface que $f(u, v) \neq f(x, y)$ si $u \neq x$, lo cual implica que dado un valor z de esta función, existe un único $u \in \{1,2,3\}$ tal que $f(u, v) = z$ para alguna palabra $v \in \{0,1\}^*$. Finalmente, suponemos que existe un algoritmo de tiempo polinomial que calcula f, y no existe un algoritmo de tiempo polinomial que dado un valor z de esta función, puede calcular $u \in \{1,2,3\}$ tal que $f(u, v) = z$ para alguna palabra $v \in \{0,1\}^*$. Estas propiedades pueden parecer un poco extrañas, pero finalmente nos dicen que podemos usar f como una función de cifrado, donde el primer parámetro es el mensaje a cifrar y el segundo parámetro es la clave usada para cifrar. Además, imponemos la restricción $u \in \{1,2,3\}$ al primer parámetro, puesto que u va a almacenar el color que se asigna a un país en el mapa, y esto es lo que vamos a cifrar.

Considere un mapa M con n países. Una 3-coloración de este mapa es una función $\phi : \{1,\ldots,n\} \to \{1,2,3\}$ que asigna un color $c \in \{1,2,3\}$ a cada país $k \in \{1,\ldots,n\}$, de manera tal que si dos países k y l son limítrofes, entonces $\phi(k) \neq \phi(l)$. Suponga entonces que usted conoce una 3-coloración ϕ de M, y suponga que existe un verificador a quien usted quiere convencer que conoce ϕ sin revelar información sobre esta coloración. Es decir, después de que usted termine con la demostración de conocimiento cero, el verificador va a estar convencido de que usted conoce ϕ, pero no va a saber cómo pintar el mapa M con tres colores. La demostración de conocimiento cero propuesta por Avi Wigderson en [8] realiza los siguientes pasos:

1. Usted primero elige al azar una permutación $\pi : \{1,2,3\} \to \{1,2,3\}$, vale decir cambia cada uno de los colores por otro color. Además, elige al azar palabras $r_1, r_2, \ldots, r_n \in \{0,1\}^*$, y para cada país $k \in \{1,\ldots,n\}$, usted calcula $S_k = f(\pi(\phi(k)), r_k)$, lo cual corresponde al cifrado con la clave r_k del color $\phi(k)$ asignado al país k, el cual también ha sido escondido por la permutación π. Finalmente envía las palabras S_1, S_2, ..., S_n al verificador.

2 Presentamos aquí una versión simplificada de la demostración original, pero que mantiene sus ingredientes fundamentales.

2. El verificador elige al azar un par de países (k, l) que son limítrofes en el mapa M, y se los envía a usted.

3. Usted demuestra al verificador que conoce los colores de los países k y l, y para esto le envía al verificador los pares $(\pi(\phi(k)), r_k)$ y $(\pi(\phi(l)), r_l)$.

4. El verificador revisa que $(\pi(\phi(k)), r_k)$ y $(\pi(\phi(l)), r_l)$ son asignaciones correctas de colores que usted no modificó desde el primer paso en que usted le envío la 3-coloración de manera cifrada. Vale decir, el verificador revisa que $S_k = f(\pi(\phi(k)), r_k)$, $S_l = f(\pi(\phi(l)), r_l)$ y $\pi(\phi(k)) \neq \pi(\phi(l))$. Si alguna de estas condiciones no se cumple entonces el verificador rechaza la demostración, y se termina el protocolo.

5. Los cuatro pasos anteriores se repiten m^2 veces, donde m es el número de pares de países (k, l) tales que k y l son limítrofes. Si nunca se rechaza en el paso 4, entonces el verificador acepta la demostración.

Si usted efectivamente conoce la función de coloración ϕ, entonces el protocolo la va a aceptar. Nótese que el verificador no va a tener información sobre la coloración por el uso de la función de permutación π, que es elegida al azar en cada una de las m^2 iteraciones. En la otra dirección, si usted no conoce la función de coloración ϕ y trata de engañar al verificador, entonces con una probabilidad altísima el verificador va a rechazar su demostración. Para ver por qué esto es así, suponga que logra engañar al verificador en las m^2 iteraciones del protocolo. En cada una de estas iteraciones usted tuvo que elegir de antemano colores para los países, los cuales no pueden ser cambiados porque van cifrados con la función f. Al recibir un par de países (k, l) elegido al azar, la probabilidad de que su coloración falle con este par es al menos $1/m$, puesto que en al menos un par usted cometió un error (ya que no conocía una 3-coloración del mapa). Entonces la probabilidad de que no sea rechazado en este paso es menor o igual a $(1\text{-}1/m)$, por lo que la probabilidad de que no sea rechazado en las m^2 iteraciones es a lo más $(1\text{-}1/m)^{m^2}$. Y es posible demostrar que esta cantidad está acotada superiormente por 2^{-m}, lo cual es una probabilidad bajísima si, por ejemplo, el mapa tiene al menos 100 países, puesto que $2^{-100} \approx 7.8 \times 10^{-31}$. Vale decir, la probabilidad de que usted engañe al verificador es a lo más 7.8×10^{-31} para un mapa con 100 países.

¿Cómo podemos demostrar que conocemos un secreto sin revelar información sobre él? [Éste es] otro de los resultados fundamentales de Avi Wigderson, quien demuestra que cada problema en NP admite una prueba de conocimiento cero.

Epílogo

No es necesario tener artículos describiendo las contribuciones de Avi Wigderson para saber que es un verdadero gigante de la computación. Mi intención aquí fue mostrar la belleza y profundidad de sus ideas, que han tenido impacto en todos los aspectos fundamentales de la complejidad computacional. En particular, mi objetivo fue mostrar cómo esta mente brillante ha sido capaz de utilizar resultados de dificultad computacional aparentemente negativos para hacer que nuestro mundo funcione de manera más eficiente. ▪

REFERENCIAS

[1] M. R. Garey, David S. Johnson: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman 1979.

[2] Robert Solovay, Volker Strassen: A Fast Monte-Carlo Test for Primality. SIAM Journal on Computing 6(1):84–85, 1977.

[3] Gary L. Miller: Riemann's Hypothesis and Tests for Primality. Journal of Computer and System Sciences 13(3):300–317, 1976.

[4] Michael Rabin: Probabilistic algorithm for testing primality. Journal of Number Theory 12(1):128–138, 1980.

[5] Manindra Agrawal, Neeraj Kayal, Nitin Saxena: PRIMES is in P. Annals of Mathematics 160(2):781–793, 2004.

[6] Noam Nisan, Avi Wigderson: Hardness vs Randomness. Journal of Computer and System Sciences 49(2):149–167, 1994.

[7] Shafi Goldwasser, Silvio Micali, Charles Rackoff: The Knowledge Complexity of Interactive Proof Systems. SIAM Journal on Computing 18(1):186–208, 1989.

[8] Oded Goldreich, Silvio Micali, Avi Wigderson: Proofs that Yield Nothing But Their Validity for All Languages in NP Have Zero-Knowledge Proof Systems. Journal of the ACM 38(3):691–729, 1991.