Notes on Computational Complexity
Notas sobre complejidad computacional

» Ricardo Rosenfeld
Centro de Altos Estudios en Tecnología Informática (CAETI), Universidad Abierta Interamericana (UAI)

Recibido: septiembre 2023./ Aceptado: noviembre 2024 / Publicado: diciembre 2024
Cómo citar Rosenfeld R. Notas sobre complejidad computacional. Revista Abierta de Informática Aplicada [Internet]. [cited 2025 Jan. 27];8(1):109-3. https://doi.org/10.59471/raia2024210

Abstract

Two central aspects of the ongoing CAETI project, to create a software development environment based on advanced concepts of modularization and behavioral synthesis, are correctness and efficiency. Regarding the first aspect, in previous articles we described the axiomatic verification of programas. In this article we focus on efficiency, presenting a series of notes that succinctly cover relevant topics in computational complexity, an area of computational theory that studies the inherent difficulty of problems. The deterministic, probabilistic and quantum paradigms are discussed, including elements related to cryptography, proofs and the derandomization of algorithms. The material integrates the contents of the course Formal Methods in Software Engineering, which es taught in the Doctorate in Computer Science of the Faculty of Computer Technology of the UAI.

KEYWORDS: computational complexity, P VS NP, algorithm.

Resumen

Dos aspectos centrales del proyecto en curso del CAETI, de creacion de un ambiente de desarrollo de software basado en conceptos avanzados de modularizacion y sintesis de comportamiento, son la correctitud y la eficiencia. En relación al primer aspecto, en artículos anteriores describimos la verificación axiomática de programas. En este artículo nos enfocamos en la eficiencia, presentando una serie de notas que cubren sucintamente temas relevantes de la complejidad computacional, área de la teoría de la computación que estudia la dificultad inherente de los problemas. Se tratan los paradigmas determinístico, probabilístico y cuántico, incluyendo elementos vinculados con la criptografía, las pruebas y la desaleatorización de los algoritmos. El material integra los contenidos de la asignatura Métodos Formales en la Ingeniería de Software, que se dicta en el Doctorado en Informática de la Facultad de Tecnología Informática de la UAI.

PALABRAS clave: complejidad computacional, P VS NP, algoritmo.

1. Introducción

Los objetos de estudio de la *complejidad computacional* son los *problemas*, con foco en los recursos que se necesitan para resolverlos, independientemente del modelo computacional empleado. Por lo tanto, su objetivo último es determinar la dificultad inherente de los problemas.

Si bien los algoritmos existen desde hace más de 2000 años, y ya antes del siglo XX se diseñaron dispositivos computacionales, se considera que los trabajos fundacionales de las ciencias de la computación son los de A. Church y A. Turing de 1936. Un poco después, pasada la segunda guerra mundial, comenzaron a construirse las primeras computadoras, y como consecuencia casi inmediata, la eficiencia de los algoritmos cobró protagonismo. En 1964, A. Cobham definió la clase P de los problemas con resoluciones algorítmicas de tiempo polinomial, y sugirió que dicha clase era una buena formalización de las computaciones eficientes. Un año más tarde, algo similar planteó J. Edmonds, y simultáneamente, J. Hartmanis, R. Stearns y P. Lewis presentaron los primeros análisis sistemáticos de complejidad temporal y espacial. Estos hitos forman parte esencial de los orígenes de la complejidad computacional, que desde entonces se ha desarrollado de manera ininterrumpida a través de notables creaciones y descubrimientos, desde el concepto de NP-completitud a comienzos de los años 1970, pasando por los algoritmos probabilísticos unos pocos años más tarde, y llegando en décadas más recientes, sólo por nombrar algunos ejemplos, a los algoritmos cuánticos, las pruebas chequeables probabilísticamente y los generadores pseudoaleatorios.

En las siguientes secciones presentamos una serie de temas, a nuestro criterio de los más relevantes de la complejidad computacional. Por razones de espacio, las descripciones son sucintas. El material forma parte de los contenidos de la asignatura Métodos Formales en la Ingeniería de Software, dictado actualmente en el Doctorado en Informática de la Facultad de Tecnología Informática de la UAI, y al igual que los artículos anteriores sobre la verificación axiomática de programas, complementa las actividades del proyecto en curso del CAETI, de creacion de un ambiente de desarrollo de software basado en conceptos avanzados de modularizacion y sintesis de comportamiento, en este caso teniendo en cuenta el aspecto de la eficiencia de los algoritmos. En las secciones 2 a 5 presentamos la clásica jerarquía espacio-temporal de la complejidad computacional. La sección 6 trata las aproximaciones polinomiales de los problemas de optimización. La sección 7 se enfoca en los problemas con resoluciones paralelas eficientes. En las secciones 8 a 10 ampliamos la perspectiva de análisis; en las secciones 8 a 9 consideramos los algoritmos probabilísticos, dedicando el capítulo 9 a las pruebas, y la sección 10 trata la complejidad computacional de los algoritmos cuánticos. Finalmente, la sección 11 completa el artículo con observaciones finales. Se asume un lector con conocimientos básicos de matemáticas (teoría de conjuntos, funciones, lógica, grafos, lenguajes, etc).

2. Escenario de estudio

Para estudiar la dificultad inherente de los problemas se necesita un escenario robusto, sobre todo que los resultados obtenidos en su marco no dependan del modelo computacional que se

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

emplee. En esta sección introducimos el modelo computacional elegido, además del universo de los problemas a considerar, las métricas para evaluar la complejidad computacional de los problemas, y otras convenciones.

La máquina de Turing

El modelo computacional que vamos a utilizar es el habitual para desarrollar los contenidos de la complejidad computacional, la *máquina de Turing* (abreviada con MT), la modelización más sencilla e intuitiva de una computadora, creada por A. Turing (Turing, 1936), la cual tuvo aceptación universal inmediatamente después de su presentación. En efecto, a pesar de su simplicidad, a casi un siglo de su creación se mantiene firme la tesis, conocida como Tesis de Church-Turing, de que una MT puede simular cualquier dispositivo computacional físicamente realizable. En lo que hace a su eficiencia, la situación es menos clara: según la Tesis Fuerte de Church-Turing, los retardos en las simulaciones de las MT en relación a los tiempos de ejecución de los dispositivos originales no son significativos, pero esto podría no ser así incluyendo entre los dispositivos a las computadoras cuánticas.

En su visión más general, una MT resuelve un *problema de búsqueda*, devolviendo a partir de toda *instancia* del problema una *solución*, si existe. Las instancias y las soluciones de los problemas se representan con *cadenas de símbolos*, o directamente *cadenas*. Por ejemplo, una MT que resuelve el *problema de accesibilidad*, dado un grafo G y dos vértices i y j de G, devuelve un camino en G de i a j, si el camino existe. Otro ejemplo es el de una MT que resuelve el *problema de satisfactibilidad*: dada una fórmula booleana φ (sin cuantificadores – las fórmulas booleanas con cuantificadores las consideraremos recién en la sección 4 –), devuelve una asignación de valores de verdad que satisface φ, si la asignación existe.

Una visión más restringida de MT, que es la que se utiliza comúnmente en la complejidad computacional y consideraremos de ahora en más (salvo en la sección 6), es la de una máquina que resuelve un *problema de decisión*, aceptando o rechazando instancias según tengan o no solución, respectivamente. Con esta visión, una MT reconoce un *lenguaje*, el lenguaje de las cadenas que representan las *instancias positivas* de un problema, es decir las instancias con solución, lo que simplifica la exposición de los temas pero sin perder generalidad con respecto a las características a describir. Por ejemplo, en el caso del problema de decisión de accesibilidad, una MT M que lo resuelve reconoce el lenguaje de las ternas (G, i, j), tales que G es un grafo y tiene un camino de su vértice i a su vértice j, y en el caso del problema de decisión de satisfactibilidad (de ahora en más también problema SAT), una MT M que lo resuelve reconoce el lenguaje de las fórmulas booleanas φ satisfactibles. En lo sucesivo, salvo mención expresa, *lenguaje* y *problema* se entenderán como sinónimos.

Una MT tiene: una *función de transición*, que es la descripción del algoritmo que ejecuta; una *cinta* infinita en ambos extremos, con celdas de memoria en las que almacena símbolos de un *alfabeto*; un *cabezal*, que recorre la cinta; y una *unidad de control*, que aloja a lo largo de las computaciones *estados* que describen la situación de las mismas (incluyendo los estados finales de aceptación y rechazo). Al comienzo, la MT está en el estado inicial, tiene en la cinta una *cadena de entrada*, y el cabezal apunta al primer símbolo de la cadena. Iniciada su ejecución, la MT avanza paso a paso de acuerdo a la función de transición, leyendo el estado guardado en la unidad de control y el símbolo apuntado por el cabezal, a partir de lo cual, eventualmente,

modifica uno o ámbos y mueve el cabezal un lugar a la derecha o a la izquierda. Cuando y si alcanza un estado final, la MT se detiene, aceptando o rechazando según cuál sea el estado. Estas son todas las capacidades de una MT, que sin embargo alcanzan para simular cualquier dispositivo computacional físicamente realizable, y corresponden a su variante estándar. Existen otras variantes de MT con poder computacional equivalente, como la MT con varias cintas; la MT *no determinística*, que a partir de un estado y un símbolo puede continuar de distintas maneras, aceptando sii alguna computación acepta (la consideraremos más adelante); la MT con cintas semi-infinitas; la MT con un solo estado además de los estados finales; etc.

Los problemas decidibles

Por la *computabilidad* sabemos que existen problemas *indecidibles*, es decir, problemas que no cuentan con MT que se detengan a partir de todas sus instancias. Un ejemplo clásico de problema indecidible es el *problema de la detención*, que consiste en determinar, dadas una MT M, codificada de alguna manera, y una cadena w, si M se detiene a partir de w, lo que extrapolado a la programación significa que en general no se puede decidir si un programa termina. Otros problemas indecidibles clásicos son el *problema de decisión en la lógica de predicados*, también conocido como Entscheidungsproblem, que plantea establecer si una fórmula ψ es un teorema de la lógica de predicados, es decir si los axiomas y las reglas de inferencia de la lógica de predicados permiten derivar la fórmula ψ, y el *problema de decisión en la aritmética*, problema definido como el Entscheidungsproblem pero en la axiomática de los números naturales.

Los problemas mencionados están íntimamente relacionados con el origen de las ciencias de la computación. En el Congreso Internacional de Matemáticos de París de 1900, D. Hilbert formuló 23 problemas que influirían notoriamente en la evolución de las matemáticas (entre otros, la Hipótesis del Continuo, el problema de la consistencia de la aritmética, la Hipótesis de Riemann, la Conjetura de Goldbach, y el problema de la resolución de las ecuaciones diofánticas). Casi tres décadas más tarde, el mismo Hilbert volvió a desafiar a la comunidad matemática con el Entscheidungsproblem, en el marco de un ambicioso proyecto que lideraba para mecanizar las pruebas matemáticas, con el objeto de asegurar la consistencia de los sistemas axiomáticos en uso sin depender de la intuición. Los Principia Mathematica de A. Whitehead y B. Russell de 1913, y la prueba de la completitud de la lógica de predicados de K. Gödel de 1929, eran indicios alentadores para una respuesta positiva al problema. Pero la prueba, en 1931, del propio Gödel, de la incompletitud de la aritmética (Gödel, 1931), acabó abruptamente con el proyecto: una de las teorías más sencillas de las matemáticas resultaba incompleta. Peor aún, cinco años más tarde, A. Church en (Church, 1936a) y (Church, 1936b), y de manera independiente A. Turing en su artículo de 1936 ya referido, probaban la indecibilidad del Entscheidungsproblem. En particular, la prueba de Turing se basaba en la indecibilidad del problema de la detención que él mismo había definido y demostrado (los resultados de Church y Turing clarificaron el concepto de lo computable y se consideran fundacionales en las ciencias de la computación).

La computabilidad recorre muchos de los dilemas planteados por los matemáticos y los filósofos durante siglos, y sus resultados, conceptos y métodos de prueba nutren a la complejidad computacional. Teniendo en cuenta, en particular, la clasificación de los problemas en decidibles e indecidibles, como la complejidad computacional se enfoca en los recursos que se necesitan para resolverlos, su ámbito de aplicación lo constituyen exclusivamente los problemas *decidibles*, es decir, los problemas que cuentan con MT que se detienen a partir de todas sus instancias.

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

Las métricas a utilizar y otras convenciones

Para establecer la complejidad computacional de un problema utilizaremos las métricas de *tiempo* y de *espacio*, las cuales consideran los pasos que ejecuta y las celdas de memoria que ocupa una MT que lo resuelve, respectivamente. De esta manera, nos referiremos a la *complejidad temporal* y la *complejidad espacial* de un problema. Existen muchas otras métricas, de distinta naturaleza, pero las mencionadas son las más intuitivas y de uso más extendido.

Además, nos basaremos en las siguientes convenciones (relacionadas con la complejidad temporal – las correspondientes a la complejidad espacial las postergamos hasta la sección 4 –):

Tiempo de ejecución con respecto al tamaño de las cadenas de entrada. Las MT tardan más (hacen más pasos) a medida que procesan cadenas de entrada w más grandes. Por eso se utilizan *funciones temporales* T(n), con n = |w| (|w| denota el tamaño de w), que especifican su crecimiento en relación al crecimiento de las cadenas, crecimiento que puede ser lineal como n, cuadrático como n^2, polinomial como n^k, cuasipolinomial como $n^{\log_2 n}$, subexponencial como $2^{\sqrt{n}}$, exponencial como 2^{n^k}, doble exponencial como 2 elevado a la 2^{n^k}, etc., con k constante. Formalmente, se define que una MT M tarda o se ejecuta en tiempo T(n) sii para toda entrada w, con |w| = n, M a partir de w hace a lo sumo T(n) pasos. En lo que sigue utilizaremos la expresión *poly(n)* para denotar una función polinomial, y la expresión *exp(n)* para denotar una función no polinomial (que identificaremos directamente como exponencial para simplificar).

Tiempo de ejecución en el peor caso. De acuerdo a la definición anterior, el tiempo de ejecución de una MT queda determinado por el *tiempo máximo* que puede tardar, considerando la población completa de las cadenas de entrada (*peor caso*). Sería lógico utilizar también el *tiempo promedio*, considerando la distribución de las entradas, y el *tiempo mínimo* insumido, cualquiera sea la entrada, pero en general estos valores son difíciles de obtener, y a pesar de importances avances, hoy día son pocos los problemas que cuentan con ellos.

Tiempo de ejecución como orden de magnitud. En lugar de utilizar tiempos exactos T(n), en la complejidad temporal se manejan *órdenes de magnitud*, que se expresan con la notación asintótica O(T(n)). Dicha expresión denota el conjunto de todas las funciones temporales acotadas superiormente por c.T(n), con c constante positiva. Formalmente, $T_1(n) = O(T_2(n))$, y se dice que la función $T_1(n)$ es *del orden* de la función $T_2(n)$ sii $T_1(n) \leq c.T_2(n)$ para todo n mayor o igual que un determinado n_0, con c > 0. Por ejemplo, $5n^2 + 3n + 8 = O(n^2)$, $n^2 = O(n^3)$, y $n^3 = O(2^n)$. El uso de órdenes de magnitud permite manejar distintos niveles de abstracción. Particularmente, los factores constantes de las funciones T(n) se pueden eliminar (Teorema de Aceleración Lineal).

Problemas tratables e intratables. Los problemas que demandan para resolverse un tiempo de ejecución muy grande, no pueden ser considerados *tratables*. Concretamente, en la complejidad temporal se toman como tratables los problemas con resolución de *tiempo polinomial*. La fundamentación matemática de esta convención es que la diferencia entre el tiempo polinomial y el tiempo exponencial se amplía bruscamente a medida que crece el tamaño de las cadenas. No menos importante, la fundamentación empírica es que al cabo de varias décadas de actividad algorítmica, el criterio adoptado se ha mantenido robusto. Podría aducirse cierta arbitrariedad en la convención: ¿acaso es tratable un problema que se resuelve en tiempo $O(n^{1000})$ e intratable uno con resolución de tiempo $O(1{,}0000001^n)$? Pero en la práctica estos valores no se observan. Diremos que los problemas tratables cuentan con MT o algoritmos *eficientes*. Obviamente,

que un problema no pueda resolverse en tiempo polinomial no implica que no sea de interés computacional. Muchísimos problemas que no cuentan al día de hoy con algoritmos eficientes son muy tenidos en cuenta. Lo que se hace en estos casos, para lograr su tratabilidad, es recurrir a algún método de remediación, como restringir el dominio de las cadenas de entrada, aplicar algún algoritmo de aproximación polinomial (en el caso de los problemas de búsqueda de un óptimo, máximo o mínimo), utilizar un modelo computacional alternativo, etc.

Modelo de máquina de Turing y representación de las cadenas. El modelo computacional adoptado es la MT *determinística con cualquier cantidad de cintas*, que según la Tesis Fuerte de Church-Turing tiene la propiedad de simular cualquier otro modelo computacional *razonable*, es decir físicamente realizable, a lo sumo con un retardo polinomial (al menos mientras las computadoras cuánticas no sean una realidad). De esta manera, un problema tratable o intratable con dicho modelo computacional será tratable o intratable, respectivamente, con cualquier otro modelo computacional razonable. Un ejemplo de modelo computacional no razonable es la MT no determinística; no es físicamente realizable porque requiere en general una cantidad exponencial de procesadores. El concepto de razonabilidad también se aplica sobre la representación de las cadenas. En este caso, se excluye la *codificación unaria* de las cadenas (cadenas de 1). Esta codificación no sólo es prácticamente irrealizable, sino que además contrasta con la robustez del resto de las codificaciones: un algoritmo eficiente o ineficiente con una codificación no unaria sigue siendo eficiente o ineficiente, respectivamente, con otra codificación no unaria, lo que no sucede cuando una de ellas es la unaria. Para uniformar, usaremos siempre la *codificación binaria* de las cadenas (cadenas de 1 y 0).

3. La jerarquía temporal

Para estudiar la complejidad temporal de los problemas se define una *jerarquía temporal*, la cual se estructura en *clases temporales* TIME(T(n)). Una clase temporal TIME(T(n)) agrupa a todos los problemas que se pueden resolver en tiempo O(T(n)). La jerarquía, que describimos a continuación, es *densa*: dada una clase TIME(T_1(n)), siempre existe otra clase TIME(T_2(n)) tal que TIME(T_1(n)) $\subset$ TIME(T_2(n)) (TIME(T_1(n)) está incluida estrictamente en TIME(T_2(n))).

La clase P

La clase TIME(T(n)) de la jerarquía temporal tal que $T(n) = n^k$ con k constante, es decir la clase de los problemas que se resuelven en tiempo polinomial, se denomina P (por polinomial), y por lo tanto es la clase de los problemas tratables.

Por la densidad de la jerarquía temporal, es obvio que deben existir problemas decidibles fuera de la clase P. La prueba de que P está incluida estrictamente en el conjunto de los problemas decidibles puede hacerse mediante el método de *diagonalización*, método muy eficaz de la complejidad computacional, y también de la computabilidad, que consiste en encontrar un problema *separador* entre dos clases de problemas (es decir, un problema que pertenezca a una clase y no pertenezca a la otra). El método proviene de las matemáticas; lo ideó G. Cantor, el creador de la teoría de conjuntos, para demostrar que la cardinalidad del conjunto de los números reales es mayor que la cardinalidad del conjunto de los números naturales.

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

Se prueba además que P es *cerrada* con respecto a las operaciones de complemento, intersección y unión, lo que significa que si dos problemas A y B pertenecen a P, entonces también pertenecen a P el complemento de A, que es el problema que tiene todas las cadenas de 1 y 0 que no tiene A más la cadena vacía sólo si no la tiene A; la intersección de A y B, que tiene las cadenas que están en A y B; y la unión de A y B, que tiene las cadenas que están en A o B.

En la clase P existen miles de problemas. El problema de accesibilidad mencionado previamente es uno de ellos. Otro ejemplo es el problema SAT también referido antes, pero restringido a las fórmulas booleanas en *forma normal conjuntiva* (conjunción de disyunciones) con cláusulas de dos literales (disyunciones de dos variables o variables negadas), problema denominado 2-SAT.

El denominador común de todos los algoritmos que resuelven los problemas de P es que son *analíticos*, en contraposición a los que consisten en la *búsqueda exhaustiva* de una solución a lo largo del espacio de todas las soluciones posibles, en general de tamaño exponencial (método conocido como de *fuerza bruta*). Contar con un algoritmo analítico, posible si se posee un conocimiento profundo de la estructura del problema que se pretende resolver, es precisamente lo que distingue a un problema de la clase P del resto de los problemas decidibles (dicho sea de paso, lo mismo que distingue, en la computabilidad, a un problema decidible de uno indecidible).

Desafortunadamente, al día de hoy contamos sólo con el método de fuerza bruta para resolver miles de problemas de interés computacional, como los que integran la segunda clase de problemas que pasamos a describir.

La clase NP

La clase NP es la clase de problemas de la jerarquía temporal que tienen la siguiente propiedad: la pertenencia de sus cadenas se puede *verificar en tiempo polinomial* con la ayuda de cadenas adicionales conocidas como *certificados*, que son comúnmente soluciones. Dicho de otra manera: podamos o no *decidir* en tiempo polinomial si una cadena pertenece a un problema de NP, lo que seguro podemos hacer es *verificar* en tiempo polinomial que la cadena pertenece al problema, si efectivamente pertenece, con la ayuda de una cadena adicional, que hace las veces de certificado o prueba. Formalmente, un problema A pertenece a NP si existe una MT M de tiempo polinomial, conocida como *verificador eficiente*, que acepta todas y sólo las cadenas w de A, contando para ello con la ayuda, en cada caso de w, de un certificado x. Naturalmente, se cumple $|x| \leq \text{poly}(|w|)$, y por eso a la cadena x se la llama *certificado sucinto*.

Una segunda caracterización de la clase NP, basada en la anterior y que retomaremos más adelante, es que todo problema A de NP cuenta con un *verificador* V (MT de tiempo polinomial), tal que para toda cadena w, si w pertenece a A entonces existe un probador P (MT de poder ilimitado) que puede convencer a V, con un certificado sucinto x, de que w pertenece a A, y si w no pertenece a A entonces ningún probador P tiene chance alguna de convencer a V de lo contrario, cualquiera sea el certificado sucinto x con que lo intente.

Hay todavía una tercera caracterización de la clase NP, de la que proviene su nombre (NP es por *no determinística polinomial*): un problema pertenece a NP si existe una MT no determinística de tiempo polinomial que lo resuelve, donde una MT no determinística tarda tiempo polinomial si todas sus computaciones tardan tiempo polinomial.

El problema SAT es un ejemplo de problema de NP. Un certificado sucinto de una fórmula booleana φ de SAT es una asignación de valores de verdad que la satisface: mide como φ, y la evaluación correspondiente tarda tiempo polinomial.

Por definición, $P \subseteq NP$ (P está incluida en NP), porque una MT que resuelve un problema de P es un verificador eficiente (que no hace uso de ningún certificado). La conjetura ampliamente aceptada es que $P \neq NP$. Intuitivamente, obtener una solución es más difícil que verificarla (por ejemplo, se supone que demostrar un teorema es más difícil que verificar una prueba del teorema). Y en la práctica, la única opción con que se cuenta actualmente para resolver miles de problemas de NP es la fuerza bruta, lo que provoca ejecuciones de tiempo exponencial, dado que una cadena de tamaño n tiene $2^{poly(n)}$ certificados sucintos posibles. El problema SAT, por ejemplo, no estaría en P: no se conoce otro método para resolverlo que no sea el de fuerza bruta, y como una fórmula booleana de m variables tiene 2^m asignaciones de valores de verdad posibles que la satisfagan, la búsqueda exhaustiva de una solución puede alcanzar un tiempo exponencial. La conjetura $P \neq NP$ tiene más de 50 años, y no parece que pueda comprobarse ni refutarse en el corto plazo. Se la ha identificado como el problema abierto más importante de las ciencias de la computación y las matemáticas. Más aún, en el año 2000, el prestigioso Instituto Clay de Matemáticas de los Estados Unidos incluyó al problema entre los *siete problemas matemáticos del milenio*, y ofreció un premio de un millón de dólares a quien logre resolverlo.

Por lo observado recién, todo problema A de NP se puede resolver con un algoritmo que, para toda cadena w, con $|w| = n$, decide su pertenencia a A procesando sus $2^{poly(n)}$ certificados posibles, cada vez en tiempo polinomial. Así, siendo EXP una tercera clase de la jerarquía temporal, la de los problemas con resolución de tiempo $O(2^{poly(n)})$, se cumple $NP \subseteq EXP$. También en este caso la conjetura ampliamente aceptada es que las clases son distintas. Por ejemplo, el complemento del problema SAT pertenece a EXP, ya que en tiempo $O(2^m)$ se puede decidir si una fórmula booleana de m variables es insatisfactible, y no estaría en NP, porque no parece posible que exista un certificado sucinto de una fórmula booleana insatisfactible (hay que considerar las 2^m asignaciones de valores de verdad posibles). Notar que esto sugiere que NP no sería cerrada con respecto a la operación de complemento, lo que refuerza la conjetura $P \neq NP$.

En definitiva, se cumple $P \subseteq NP \subseteq EXP$, pareciera que ambas inclusiones son estrictas, y al menos una inclusión lo es porque $P \subset EXP$.

El problema P vs NP

Numerosas publicaciones reflejan la opinión generalizada de que se cumple $P \neq NP$, como por ejemplo (Gasarch, 2002), (Gasarch, 2012) y (Gasarch, 2019), tres encuestas en las que participaron los referentes más importantes de la complejidad computacional de las últimas dos décadas, quienes además coincidieron mayoritariamente en que el problema P vs NP se resolverá con un método matemático hoy inexistente y no en el corto plazo.

Resulta utópica la idea de que los miles de algoritmos de fuerza bruta, de tiempo exponencial, que hoy día caracterizan a la clase NP, puedan sustituirse alguna vez por otros que sean eficientes. Entre otras cosas, podríamos ejecutar eficientemente las búsquedas exhaustivas de los inmensos árboles de posibilidades propios de la inteligencia artificial, y contaríamos con demostradores automáticos de teoremas de tiempo polinomial para muchos sistemas axiomáticos de interés.

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

Algunos problemas de NP pueden estar en P, la cuestión es encontrarlos, lo que requiere insoslayablemente conocer profundamente su estructura, además de ingenio, experiencia y metodología (estrategias, métodos, técnicas, como *dividir y conquistar*, *programación dinámica*, *algoritmos codiciosos*, *problema dual*, *reducciones*, etc). Y es evidente que en la algorítmica, con un derrotero histórico aún breve, queda mucho por aprender, sobre todo entender por qué algunos problemas son más difíciles que otros. Podemos verificarlo revisando algunos de sus numerosos resultados, cuanto menos curiosos:

- La dificultad de muchos problemas, en general difíciles, se reduce ampliamente cuando se *especializan*. Lo vimos en el problema de satisfactibilidad: mientras que con fórmulas booleanas de forma arbitraria no estaría en P (problema SAT), sí lo está con fórmulas booleanas en forma normal conjuntiva con dos literales por cláusula (problema 2-SAT). Ya con cláusulas de tres literales (problema 3-SAT), la tratabilidad se volvería a perder.
- Algo similar con los números 2 y 3 ocurre en el *problema de coloración de grafos*, que consiste en determinar si con K colores se pueden colorear los vértices de un grafo, de modo tal que nunca dos vértices adyacentes queden con el mismo color: cuando K = 2, el problema se resuelve en tiempo polinomial, y en cambio si K = 3, no se le conoce algoritmo eficiente alguno. Pero a diferencia de SAT, a partir de K = 4, si los grafos son *planares*, es decir si pueden dibujarse en el plano sin que sus arcos se crucen, el problema siempre tiene solución (de acuerdo al Teorema de los Cuatro Colores, probado por computadora, motivo por el cual generó no pocas controversias en el ambiente matemático).
- Por el contrario, algunos problemas se tornan más difíciles cuando se especializan. Un ejemplo es la *programación lineal*: se puede decidir eficientemente si un sistema de inecuaciones lineales tiene solución en el dominio de los números reales, lo que no sucedería si se requiere que las soluciones sean exclusivamente de números enteros (variante conocida como *programación lineal entera*).
- Otra curiosidad para destacar es la existencia de problemas con especificaciones muy similares pero que se resuelven con algoritmos de eficiencia muy diferente. Es el caso, por ejemplo, del *problema del circuito hamiltoniano* y el *problema del circuito euleriano*. El primero, que no pertenecería a P, plantea establecer, dado un grafo, si arrancando y terminando en uno de sus vértices se pueden recorrer sin repetir todos sus vértices restantes. El segundo problema plantea lo mismo, pero considerando el recorrido de los arcos en lugar del recorrido de los vértices, y se prueba que se puede resolver en tiempo polinomial, teniendo en cuenta una propiedad que deben cumplir los vértices del grafo.

Frente al panorama actual de abundancia de problemas abiertos en la complejidad temporal (en realidad, en la complejidad computacional en general), que contrasta con las certezas de la computabilidad, se ha configurado un escenario de estudio con distintas herramientas complementarias, como el concepto de *completitud* y una variedad de aproximaciones más de naturaleza *estructural* que con foco en problemas particulares, definidas considerando distintas jerarquías y distintos modelos computacionales. Dichas herramientas permiten formular enunciados *condicionales*, con *conjeturas*. La idea, que se puede observar en algunos pasajes de las próximas secciones, es relacionar resultados, apuntando a esclarecer algunos a partir del esclarecimiento de otros.

Los problemas NP-completos

En ciertos problemas de la clase NP, en comparación con el resto, la evidencia de que no pertenecerían a la clase P es más marcada. Nos referimos a los problemas *NP-completos*, que constituyen la clase NPC.

Cada uno de estos problemas tiene la propiedad de que si perteneciera a P, entonces todos los problemas de NP pertenecerían a P, es decir que se cumpliría la igualdad P = NP. La demostración de este enunciado se basa en el método de prueba conocido como *reducción polinomial*, método de la complejidad computacional tan eficaz como el método de diagonalización (según el caso conviene recurrir a uno u otro). Reducir polinomialmente un problema A a un problema B consiste en construir una MT de tiempo polinomial que transforme toda cadena de A en una cadena de B y toda cadena fuera de A en una cadena fuera de B, lo que permite resolver eficientemente A si se cuenta con una MT de tiempo polinomial que resuelve B, dado que componiendo secuencialmente dos MT de tiempo polinomial se obtiene otra MT de tiempo polinomial. La notación $A \leq_p B$ expresa que existe una reducción polinomial del problema A al problema B. Las reducciones, sin considerar en qué tiempo se llevan a cabo, son muy útiles también en la computabilidad, para poblar las clases de los problemas decidibles y los problemas indecidibles.

Formalmente, se define que un problema es NP-completo si pertenece a NP y es *NP-difícil*, lo que significa que todos los problemas de NP se reducen polinomialmente a él. De la definición y lo dicho antes se desprende que resolver eficientemente un problema NP-completo permite resolver eficientemente cualquier problema de NP. En otras palabras, los problemas NP-completos son los más difíciles de NP, y así identifican la dificultad de toda la clase. En la práctica, como la conjetura ampliamente aceptada es que $P \neq NP$, un problema NP-completo queda *condenado* a no pertenecer a P.

Existen miles de problemas NP-completos, relacionados con grafos, fórmulas booleanas, números, conjuntos, sistemas de ecuaciones, autómatas, etc., todos ellos de mucho interés computacional. El primer problema que se probó que era NP-completo fue SAT, a comienzos de la década de 1970, de manera independiente por parte de S. Cook en los Estados Unidos (Cook, 1971) y L. Levin en la Unión Soviética (Levin, 1973). Más allá de las fechas de las publicaciones, las demostraciones se hicieron prácticamente al mismo tiempo, curiosamente como A. Church y A. Turing demostraron en 1936 la indecibilidad del Entscheidungsproblem. La simultaneidad de los trabajos de Cook y Levin no fue casual: a uno y otro lado de la Cortina de Hierro se estaban analizando desde hacía tiempo varios problemas, de la investigación operativa, la lógica, la aritmética, la combinatoria, etc., que se distinguían por su grado de dificultad y su mutuo relacionamiento. De todos modos, observando los trabajos de esa época, pareciera que en general la comunidad científica de entonces no captó en su real dimensión la relación entre la NP-completitud y el problema P vs NP.

Probada la NP-completitud de SAT, la clase NPC se pobló rápidamente, por medio de reducciones polinomiales primero desde SAT y después desde otros problemas NP-completos, dada la transitividad de las mismas (se cumple que si $A \leq_p B$ y $B \leq_p C$, entonces $A \leq_p C$). Algunos de los primeros problemas NP-completos encontrados después de SAT fueron el *problema del cubrimiento de vértices*, consistente en determinar si un grafo G tiene un cubrimiento de vértices de tamaño K, es decir un subconjunto de K vértices con al menos un extremo de todos los arcos de G; el *problema del clique*, que plantea establecer si un grafo incluye un subgrafo completo de K vértices; los problemas del circuito hamiltoniano y de coloración de grafos mencionados

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

anteriormente; y el *problema del viajante de comercio*, problema en el que se pretende determinar, a partir de un grafo ponderado G y un número K, si G tiene un circuito hamiltoniano tal que los valores de sus arcos suman a lo sumo K.

Una idea fascinante que emerge a partir de las pruebas de NP-completitud con reducciones polinomiales, es que tras la enorme cantidad de problemas de la clase NPC en realidad se esconden muchos menos, o expresado de otro modo, que en el ámbito de la NP-completitud existen pocas familias de problemas, cada una con muchos problemas que son en verdad un *único problema* pero especificado de múltiples maneras y en múltiples disciplinas. Distintos hechos refuerzan esta interpretación, como por ejemplo:

- Tomando dos problemas cualesquiera de gran parte de los problemas NP-completos conocidos, se cumple que los certificados de las cadenas de uno se pueden transformar eficientemente en los certificados de las cadenas del otro.
- Entre todo par de problemas NP-completos conocidos existe una biyección, tal que tanto ella como su inversa son computables en tiempo polinomial (se prueba que si esto se cumpliera para todos los problemas NP-completos, valdría $P \neq NP$).
- Todos los problemas NP-completos conocidos son *densos*, tienen muchas cadenas (se prueba que si se encontrara algún problema NP-completo *disperso*, es decir que no sea denso, valdría $P = NP$).

En la práctica, cuando se prueba que un problema es NP-completo se encara alguna alternativa de *remediación*, para poder procesarlo en un tiempo razonable (se asume $P \neq NP$, y por lo tanto que el problema no puede resolverse en tiempo polinomial). Entre las remediaciones se encuentran la *restricción a cadenas de cierto tipo*, el *método de retroceso* (*backtracking*), y las *aproximaciones polinomiales* (la sección 6 la dedicamos a estas últimas).

Pero teóricamente no se descartan otras posibilidades. Una posibilidad es que se encuentre un problema NP-completo que se resuelva en tiempo polinomial pero con un exponente muy grande, del tipo $O(n^{100})$. Un mundo en estas condiciones podría no ser utópico como el que caracterizamos antes (el costo para obtener una solución seguiría siendo significativamente mayor que el costo para verificarla). Una segunda posibilidad es, de nuevo, que se demuestre la pertenencia de un problema NP-completo a la clase P pero con una prueba no constructiva. La cuestión en esta situación es que aún sabiendo que existe un algoritmo polinomial para resolver el problema, no tendríamos idea alguna sobre su estructura. Y todavía queda una tercera posibilidad, aún más problemática que la anterior (que de todos modos la opinión generalizada descarta): que la lógica que manejamos no sea lo suficientemente potente como para permitirnos probar la relación entre P y NP. En otras palabras, que las dos conjeturas, $P \neq NP$ y $P = NP$, sean *independientes* en el marco lógico existente. En este caso podría suceder que contemos con un algoritmo que resuelva eficientemente un problema NP-completo pero que no lo podamos comprobar.

La clase NPI

Si $P \neq NP$, además de P y NPC la clase NP incluye, de acuerdo al Teorema de Ladner (Ladner, 1975), una tercera clase disjunta de problemas, la clase NPI de los problemas de dificultad intermedia (de ahí su nombre), problemas no tan difíciles como los de NPC ni tan fáciles como los de P.

Un problema de NPI se puede pensar como la intersección de un problema de NPC y un problema de P. Por ejemplo, restringiendo el problema NP-completo del circuito hamiltoniano a las cadenas de un problema de grafos de P, podemos obtener un problema de grafos de NPI. La idea subyacente es la siguiente: quitándole a un problema NP-completo una cantidad suficiente de cadenas, ni muchas ni pocas, se genera otro problema con un grado de dificultad menor; no se le sacan muchas cadenas como para que se transforme en un problema de P, ni se le sacan pocas cadenas como para que preserve su NP-completitud.

Dos problemas de NP candidatos a estar en NPI son el *problema del isomorfismo de grafos* y el *problema de factorización*. Al día de hoy no se ha podido probar su pertenencia a P ni a NPC.

El problema del isomorfismo de grafos (de ahora en más también problema ISO) consiste en determinar si dos grafos G_1 y G_2 de m vértices (vértices 1 a m) son isomorfos, es decir si son iguales salvo por la denominación de sus arcos (pares de vértices). Formalmente, dos grafos G_1 y G_2, cada uno con vértices 1 a m, son isomorfos, si existe una permutación π de (1, …, m), expresada con (π(1), …, π(m)), tal que (i, j) es un arco de G_1 sii (π(i), π(j)) es un arco de G_2. Notar que dicha permutación hace las veces de certificado sucinto. El problema ISO mereció especial atención hace unos años, cuando L. Babai anunció que había encontrado un algoritmo de tiempo cuasipolinomial para resolverlo (Babai, 2016).

Por su parte, el problema de factorización en su forma de problema de decisión plantea decidir, dada una terna de números (N, M_1, M_2), si N tiene un factor *primo* en el intervalo [M_1, M_2], siendo un número primo un número natural mayor que 1 divisible solamente por 1 y por si mismo.

La factorización es un muy buen ejemplo de cómo se puede sacar provecho de la dificultad de un problema. En este caso, la supuesta dificultad permite implementar un sistema de seguridad de uso muy extendido, el *sistema criptográfico de clave pública*, cuya variante más popular es el *Sistema RSA*. Descripto muy simplificadamente, el sistema se basa en la hipótesis de que la multiplicación es una *función de un solo sentido*, lo que significa que es fácil de computar (tiempo polinomial) pero en general difícil de invertir (tiempo no polinomial). Por ejemplo, el cálculo de $N = N_1.N_2$, siendo N_1 y N_2 dos números primos muy grandes, es fácil, pero en cambio obtener N_1 y N_2 de N sería difícil. Tomando esto en consideración, se utilizan dos claves, una *clave pública e* conocida por todos los emisores de mensajes, que incluye un número como N, y una *clave privada* d conocida sólo por el receptor de los mensajes, que incluye dos números como N_1 y N_2, y el mecanismo de emisión y recepción de mensajes es el siguiente: el emisor le envía al receptor un mensaje encriptado v, obtenido mediante un algoritmo de encriptación E de tiempo polinomial aplicado sobre un par (w, e), siendo w el mensaje original; y el receptor recibe el mensaje encriptado v y lo desencripta mediante un algoritmo de desencriptación D también de tiempo polinomial, aplicado sobre el par (v, d), obteniendo el mensaje original w. Así, la única forma que tiene un *hacker* de interceptar el mensaje w eficientemente es conociendo la clave privada d.

Curiosamente, el *problema de primalidad*, que plantea establecer si un número es primo, tiene resolución polinomial. Es interesante observar el camino recorrido para llegar a dicho resultado: en 1975 se probó la pertenencia del problema a NP, entre 1977 y 1992 se encontraron algoritmos probabilísticos eficientes para resolverlo, y finalmente en 2002 se encontró una resolución polinomial, mejorada un par de años más tarde (Agrawal, Kayal y Saxena, 2004). Otra curiosidad

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

es que en 1994, P. Shor resolvió con un algoritmo cuántico de tiempo polinomial el problema de factorización en su forma de problema de búsqueda, es decir el problema de obtener los factores primos de un número (con él, la comunidad científica se había esperanzado con la posibilidad de que los algoritmos cuánticos resolvieran eficientemente los problemas NP-completos – volvemos a esto en la sección 10 –).

La clase CO-NP

En esta última subsección sobre la jerarquía temporal dentro de la clase EXP presentamos la clase CO-NP, la clase de los complementos de los problemas de NP.

La conjetura ampliamente aceptada es que NP ≠ CO-NP. Intuitivamente, complementos como los del problema SAT, por ejemplo, no parecen contar con certificados sucintos: para decidir si una fórmula booleana de m variables es insatisfactible no alcanza con una asignación de valores de verdad, sino que se deben considerar las 2^m asignaciones posibles. Lo mismo se puede decir de los complementos de los problemas de NP del cubrimiento de vértices, clique, circuito hamiltoniano, coloración de grafos, etc. Otro ejemplo clásico de problema de CO-NP que no estaría en NP es el *problema de las fórmulas booleanas tautológicas*, es decir el problema en el que se busca establecer si una fórmula booleana es satisfactible con todas las asignaciones de valores de verdad posibles.

En el caso de los problemas de NPI descriptos en la subsección anterior, el complemento de ISO tampoco estaría en NP, pero en cambio sí lo está el complemento del problema de factorización en su forma de problema de decisión, lo que sugiere que encontrar un factor primo de un número sería menos difícil que encontrar un isomorfismo entre dos grafos. Esto podría justificarse por el hecho de que mientras las instancias de ISO pueden no tener solución o tener varias, un número siempre se puede factotrizar y de una sola manera, de acuerdo al Teorema Fundamental de la Aritmética.

La clase CO-NP también tiene problemas completos. La *CO-NP-completitud* se define de la misma manera que la NP-completitud: un problema A es *CO-NP-completo* (o pertenece a la clase CO-NPC) sii A pertenece a CO-NP y todos los problemas de CO-NP se reducen polinomialmente a él. Se cumple que el complemento de un problema NP-completo es CO-NP-completo. Por ejemplo, el complemento de SAT, es decir el problema que plantea determinar si una fórmula booleana es insatisfactible, es CO-NP-completo. También pertenece a CO-NPC el problema en el que se busca decidir si una fórmula booleana es una tautología (se puede probar con una reducción polinomial desde el complemento de SAT).

Asumiendo NP ≠ CO-NP, se demuestra que NP ⋂ CO-NP no incluye problemas NP-completos ni CO-NP-completos. Intuitivamente, los problemas de NP ⋂ CO-NP son más fáciles que el resto de los problemas de NP ⋃ CO-NP, porque a diferencia de éstos, cumplen que tanto ellos como sus complementos cuentan con verificadores eficientes. De la clausura de P con respecto al complemento se deriva P ⊆ CO-NP. De la relación entre P y NP ⋂ CO-NP sólo se puede decir que vale P ⊆ NP ⋂ CO-NP. Y otra inclusión que se cumple es CO-NP ⊆ EXP, porque para decidir la pertenencia de una cadena w de tamaño n al complemento de un problema A de NP, alcanza con chequear sus $2^{poly(n)}$ certificados sucintos posibles, lo mismo que se requiere en el peor caso para decidir la pertenencia de w a A.

En síntesis, considerando las distintas clases temporales descriptas en el marco de la clase EXP, se puede establecer el siguiente orden entre las mismas, de menor a mayor grado de dificultad:

1. La clase P.
2. La clase (NP ⋂ CO-NP) – P.
3. La clase NPI – (NP ⋂ CO-NP).
4. La clase NPC.
5. La clase CO-NPI – (NP ⋂ CO-NP).
6. La clase CO-NPC.

4. La jerarquía espacial

Tratamos ahora la complejidad espacial de los problemas. Como sucede con el tiempo, en más espacio se pueden resolver más problemas. La analogía con la complejidad temporal también se manifiesta en las convenciones que hemos venido utilizando: el espacio se mide con respecto al tamaño de las cadenas de entrada; se considera el peor caso; se utilizan órdenes de magnitud, ahora mediante *funciones espaciales* S(n); el modelo computacional es el de las MT determinísticas con varias cintas; y de la representación de las cadenas se excluye la codificación unaria (seguiremos considerando la codificación binaria).

Formalmente, se define que una MT M se ejecuta o es de espacio S(n) sii para toda cadena de entrada w tal que $|w| = n$, M a partir de w ocupa a lo sumo S(n) celdas en cualquiera de sus cintas, *sin considerar su cinta de entrada*, que es de *sólo lectura*, es decir que su cabezal únicamente se mueve a lo largo de la entrada w (más los dos símbolos blancos que la delimitan a izquierda y derecha) y los símbolos de w no pueden modificarse. La cinta de entrada se excluye de las mediciones para lograr espacio logarítmico (si no, ya se va a ocupar por lo menos espacio lineal), espacio necesario como vamos a justificar enseguida. El tratamiento de la *cinta de salida*, si fuese necesario utilizarla, también es especial: es de *sólo escritura*, es decir que sobre ella únicamente se puede escribir un símbolo y avanzar un lugar a la derecha. Estas características de las cintas de entrada y salida tienen su correlato en las computadoras reales, en las que se distingue la memoria externa de la memoria interna. Otras convenciones propias de la complejidad espacial son que toda función S(n) satisface la relación $S(n) \geq \log_2 n$, para que mínimamente puedan almacenarse índices de celdas, y que como una MT con una cinta puede simular una MT con varias cintas en el mismo espacio, se utilizan por defecto MT con una cinta de entrada de sólo lectura y una única cinta de entrada/salida, identificada habitualmente como *cinta de trabajo*.

Al igual que en la complejidad temporal, en la complejidad espacial se define una jerarquía, la *jerarquía espacial*, que describimos a continuación y relacionamos con la jerarquía temporal para terminar convergiendo en una única jerarquía espacio-temporal. La jerarquía espacial se estructura en *clases espaciales* SPACE(S(n)), cada una agrupando a todos los problemas que se pueden resolver en espacio O(S(n)). También en este caso la jerarquía es *densa*, dada una clase $SPACE(S_1(n))$ siempre existe otra clase $SPACE(S_2(n))$ que la incluye estrictamente. Otra similitud con la jerarquía temporal es que los factores constantes de las funciones espaciales se pueden eliminar (Teorema de Compresión de Cintas).

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

Espacio polinomial y espacio logarítmico

Los problemas que se pueden resolver en espacio polinomial conforman la clase PSPACE, y los que se pueden resolver en espacio logarítmico, la clase LOGSPACE.

Como una MT que tarda tiempo T(n) no ocupa más que espacio T(n), porque en T(n) pasos se pueden recorrer a lo sumo T(n) celdas, se cumple $P \subseteq PSPACE$, lo que significa que los problemas tratables se pueden resolver en espacio polinomial.

Como contrapartida, una MT que ocupa espacio S(n) puede tardar mucho más que tiempo S(n), tanto como exp(S(n)), producto de la gran cantidad de configuraciones distintas que puede recorrer en dicho espacio: s estados, t símbolos, una cinta de entrada de sólo lectura y una cinta de trabajo, determinan en total $K = (n + 2).S(n).s.t^{S(n)}$ configuraciones distintas, por las n + 2 posiciones del cabezal de la cinta de entrada, las S(n) posiciones del cabezal de la cinta de trabajo, los s estados y los $t^{S(n)}$ contenidos de la cinta de trabajo (la cinta de entrada tiene un solo contenido). Se prueba que K se puede acotar con $4.s.t^{S(n)}$, por lo que para simplificar se suele utilizar dicha expresión, o directamente $c^{S(n)}$, siendo c una constante que depende de la MT. De este modo, espacio S(n) implica tiempo a lo sumo $c^{S(n)}$, y por lo tanto, en espacio poly(n) el tiempo puede llegar a ser $c^{poly(n)}$, es decir que se cumple $PSPACE \subseteq EXP$, y en espacio $\log_2 n$ el tiempo puede llegar a ser $c^{log2n} = n^{log2c}$, es decir poly(n), y así $LOGSPACE \subseteq P$, lo que significa que los lenguajes de LOGSPACE son tratables.

En lo que hace a la relación de la clase temporal NP con las clases espaciales PSPACE y LOGSPACE, una característica fundamental de la complejidad espacial la explica: *el espacio se puede reutilizar*. Podemos verlo por ejemplo con el problema SAT. SAT no podría resolverse en tiempo polinomial, pero sí puede resolverse en espacio polinomial: el espacio requerido para procesar un certificado sucinto posible es polinomial, y se puede reutilizar en el procesamiento de otro. De esta capacidad de iterar sobre todos los certificados sucintos posibles, reutilizando cada vez el mismo espacio polinomial, se deriva la inclusión $NP \subseteq PSPACE$.

Así llegamos a las inclusiones $LOGSPACE \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP$, y al menos una inclusión entre LOGSPACE y PSPACE es estricta, porque se cumple $LOGSPACE \subset PSPACE$. Lo mismo sucede con las inclusiones entre P y EXP, porque se cumple $P \subset EXP$. La conjetura más aceptada es que todas las inclusiones son estrictas.

Otra clase relevante de la jerarquía espacial es NLOGSPACE, definida como NP en términos de verificadores y certificados (también, y más habitualmente, en términos de MT no determinísticas con computaciones que ocupan espacio logarítmico). Formalmente, un problema A pertenece a NLOGSPACE si cuenta con una MT M de espacio logarítmico (*verificador eficiente*), que acepta todas y sólo las cadenas w de A, con la ayuda de certificados x, tal que $|x| \leq poly(|w|)$ (*certificados sucintos*). En este caso, los certificados se encuentran en otra cinta especial de la MT M, la cual es de sólo lectura como la cinta de entrada, y además tiene la restricción de que su cabezal se mueve únicamente a la derecha. Tal restricción se debe a que el procesamiento de un certificado tiene que hacerse en simultáneo con su lectura, símbolo a símbolo, porque de lo contrario excedería el espacio logarítmico. Naturalmente, se cumple $LOGSPACE \subseteq NLOGSPACE$, y la conjetura más aceptada es que la inclusión es estricta. Adicionalmente, siendo CO-NLOGSPACE la clase de los complementos de los problemas de NLOGSPACE, como consecuencia del Teorema de Immerman (Immerman, 1980) se cumple NLOGSPACE = CO-NLOGSPACE, igualdad para nada intuitiva (en la jerarquía temporal se cumpliría $NP \neq CO\text{-}NP$).

Análogamente, se define la clase NPSPACE de los problemas que cuentan con verificadores de espacio polinomial basados en certificados, con las mismas restricciones que describimos en el caso de NLOGSPACE (también, y más habitualmente, se define con MT no determinísticas con computaciones de espacio polinomial). Tampoco intuitiva, como consecuencia del Teorema de Savitch (Savitch, 1970) se prueba la igualdad PSPACE = NPSPACE, que contrasta con la conjetura más aceptada P ≠ NP. Notar que dicha igualdad expresa que si en espacio polinomial se puede verificar la pertenencia de una cadena w a un problema A, leyendo y procesando en simultáneo un certificado posible de w, en espacio polinomial directamente se puede decidir la pertenencia de w a A, sin ayuda de ninguna cadena adicional.

Un problema representativo de la clase LOGSPACE es el problema de accesibilidad, pero particularmente en grafos no dirigidos. La prueba la presentó en 2005 O. Reingold (Reingold, 2005), hito muy relevante en la historia de la complejidad espacial. Curiosamente, el problema de accesibilidad en grafos dirigidos no estaría en LOGSPACE, pareciera ser más difícil. Más aún, en la práctica queda *condenado* a no pertenecer a LOGSPACE, porque se prueba que está entre los problemas más difíciles de NLOGSPACE (mecanismo de razonamiento que recuerda el de las pruebas de NP-completitud, y de hecho el problema es *NLOGSPACE-completo*, definición que formalizamos más adelante). Por otra parte, el problema de accesibilidad en grafos dirigidos está en P, lo que sugiere que todos los problemas de NLOGSPACE pertenecen a P, es decir que NLOGSPACE $\subseteq$ P, lo cual se cumple efectivamente. Por lo tanto, también los problemas de NLOGSPACE son tratables. Así, la serie de inclusiones que presentamos anteriormente se puede ampliar del siguiente modo: LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXP, y la conjetura más aceptada sigue siendo que todas las inclusiones son estrictas.

Entre los problemas representativos de la clase PSPACE se encuentra el problema de satisfactibilidad, pero ahora con fórmulas booleanas totalmente cuantificadas. Se lo suele denominar QSAT (*SAT cuantificado*), y consiste entonces en determinar si una fórmula booleana totalmente cuantificada es verdadera. A diferencia de SAT, QSAT no estaría en NP. En realidad, ni siquiera estaría en CO-NP (los certificados correspondientes no son secuencias sino árboles de valores de verdad). Además, QSAT está entre los problemas más difíciles de PSPACE, es *PSPACE-completo*, definición que formalizamos después, y forma parte de una familia de problemas que capturan la esencia de la clase. Nos referimos a los problemas que plantean decidir si en un juego de dos jugadores, con información perfecta, el primer jugador cuenta con una estrategia ganadora. Más precisamente, dados dos jugadores J_1 y J_2 que hacen sus jugadas alternadamente sobre un tablero u otra estructura visible en todo momento por ámbos (por eso se dice que la información es perfecta), hay que determinar si existe una jugada de J_1 tal que para toda jugada de J_2 existe una jugada de J_1 tal que para toda jugada de J_2 … J_1 gana. Algunos ejemplos de juegos de esta naturaleza son el ajedrez, las damas, el go, el juego del hexágono y el juego de geografía, pero con ciertas adaptaciones, siendo una de ellas la especificación del tamaño del tablero de juego o de la estructura que corresponda en términos de n, para lograr un tratamiento asintótico. En lo que hace particularmente a QSAT visto como un juego, los jugadores son $\exists$ y $\forall$, la estructura que se utiliza es una fórmula booleana φ sin cuantificadores con variables x_1, x_2, x_3, x_4, …, y el juego consiste en que $\exists$ y $\forall$ asignen alternadamente un valor de verdad a una variable de φ, primero $\exists$ a x_1, después $\forall$ a x_2, después $\exists$ a x_3, después $\forall$ a x_4, etc., ganando $\exists$ si φ resulta verdadera y ganando $\forall$ si φ resulta falsa. Así, para que haya una estrategia ganadora para $\exists$, tiene que haber alguna asignación a x_1 tal que para toda asignación a x_2 exista alguna asignación a x_3 tal que para toda asignación a x_4 … la fórmula φ resulte verdadera, lo que significa que $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots$:

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

φ tiene que pertenecer a QSAT. En cualquier caso, para determinar si existe una estrategia ganadora para el primer jugador, se deben considerar todas las posibles alternativas a partir de su primera jugada, lo que recuerda las búsquedas exhaustivas asociadas a la clase NP, pero ahora con certificados no sucintos.

Problemas completos de la jerarquía espacial

La completitud no sólo se considera en las clases NP y CO-NP. Es un concepto que las trasciende, fundamental de la complejidad computacional. Permite identificar el grado de dificultad de una clase de problemas, temporal o espacial, a través de problemas representativos, *completos* de dicha clase. En esta subsección incluimos nociones generales sobre la completitud, y nos referimos particularmente a problemas completos de la jerarquía espacial,

Vimos que si un problema es NP-completo, es decir si pertenece a NP y es NP-difícil (todos los problemas de NP se reducen polinomialmente a él), entonces se encuentra entre los problemas más difíciles de NP: no pertenece a P a menos que P = NP, y así identifica el grado de dificultad de NP. Este mismo mecanismo se puede generalizar a todas las clases, con la salvedad de que en las clases relacionadas con el espacio logarítmico, las reducciones polinomiales utilizadas no pueden exceder el espacio logarítmico. Formalmente, llamando *poly-time* a las reducciones de tiempo polinomial generales y *log-space* a las reducciones poly-time particularmente de espacio logarítmico, y utilizando las notaciones $A \leq_p B$ y $A \leq_{log} B$ para expresar que existe una reducción poly-time o una reducción log-space del problema A al problema B, respectivamente, se define que un problema B es *C-difícil* con respecto a las reducciones poly-time (log-space) sii para todo problema A de la clase C se cumple $A \leq_p B$ ($A \leq_{log} B$), y si además B pertenece a la clase C, entonces B es *C-completo* con respecto a las reducciones correspondientes. Así, dadas dos clases de problemas C_1 y C_2 tales que $C_1 \subseteq C_2$, si un problema es C_2-completo con respecto a las reducciones correspondientes, no pertenece a C_1 a menos que $C_1 = C_2$. De este modo, instanciando estas definiciones a clases de problemas que hemos descripto, y siguiendo un orden de menor a mayor grado de dificultad, podemos establecer:

- Si un problema es NLOGSPACE-completo con respecto a las reducciones log-space, no pertenece a LOGSPACE a menos que LOGSPACE = NLOGSPACE.
- Si un problema es P-completo con respecto a las reducciones log-space, no pertenece a NLOGSPACE a menos que NLOGSPACE = P.
- Ya formulado en la sección anterior, si un problema es NP-completo con respecto a las reducciones poly-time, no pertenece a P a menos que P = NP.
- Si un problema es PSPACE-completo con respecto a las reducciones poly-time, no pertenece a NP a menos que NP = PSPACE.

Para definir la completitud en la clase LOGSPACE no alcanza ni siquiera con las reducciones log-space; se debe recurrir a otra alternativa porque todos los problemas de LOGSPACE se pueden reducir entre sí con reducciones log-space, lo que se prueba de la misma manera que se prueba que todos los problemas de P se pueden reducir entre sí con reducciones poly-time.

Como anticipamos, el problema de accesibilidad en grafos dirigidos es NLOGSPACE-completo, y el problema QSAT es PSPACE-completo (las pruebas correspondientes siguen la misma idea general de la prueba de la NP-completitud del problema SAT). En especial, la PSPACE-completitud

de QSAT, probada en 1973 (Stockmeyer y Meyer, 1973), es una evidencia más de la íntima relación existente entre la complejidad computacional y la lógica, y así de la relevancia de esta última para encarar problemas actualmente abiertos.

5. Más allá de la clase PSPACE

Completamos a continuación la descripción de la clásica jerarquía espacio-temporal de la complejidad computacional que fuimos definiendo, con una rápida mirada a la región de los problemas que se extiende más allá de la frontera de la clase PSPACE.

La clase EXP también tiene asociadas clases NEXP y CO-NEXP, definidas mediante verificadores eficientes y MT no determinísticas. NEXP incluiría estrictamente a EXP, lo mismo que CO-NEXP, repitiéndose así el mismo tipo de conjeturas que mencionamos en relación a las clases P, NP y CO-NP. A su vez, incluyendo a NEXP y CO-NEXP estaría la clase EXPSPACE de los problemas que se pueden resolver en espacio $O(2^{poly(n)})$. Y luego seguirían las clases 2-EXP (tiempo doble exponencial), 3-EXP (tiempo triple exponencial), 4-EXP (tiempo cuádruple exponencial), etc., sin solución de continuidad hasta la misma frontera de la clase de los problemas decidibles.

Mientras que se conjetura que dentro de PSPACE, los problemas NP-completos, CO-NP-completos y PSPACE-completos serían intratables, en lo que hace al resto de la jerarquía espacio-temporal se afirma que la habitan problemas efectivamente intratables, es decir que su intratabilidad se puede probar sin apelar a ninguna conjetura: como la clase EXP incluye estrictamente a la clase P, los problemas por lo menos EXP-completos (con respecto a las reducciones poly-time) no pueden ser resueltos en tiempo polinomial. Entre ellos hay dos grupos que se destacan:

El primer grupo corresponde a problemas de grafos y fórmulas booleanas que se representan sucintamente. En particular, los problemas de grafos sucintos son de sumo interés en el área del diseño de circuitos integrados, donde se requieren sofisticadas técnicas de codificación. Un ejemplo es el problema del circuito hamiltoniano en un grafo sucinto, el cual se prueba que es NEXP-completo (con respecto a las reducciones poly-time). Otro ejemplo es el problema de satisfactibilidad de las fórmulas booleanas codificadas mediante circuitos sucintos, también NEXP-completo.

El segundo grupo corresponde a problemas de decisión en sistemas axiomáticos decidibles. Por ejemplo, el problema de decisión en la lógica de predicados con *fórmulas de Schönfinkel-Bernays* (fórmulas que empiezan con los cuantificadores, primero los existenciales y luego los universales, y no tienen símbolos de función ni de igualdad), es NEXP-completo. Otro ejemplo es el problema de decisión en la teoría de los números reales con la suma, que es NEXP-difícil (la teoría también es decidible con la multiplicación, lo que resulta contraintuitivo dada la indecibilidad en la aritmética con la multiplicación, pero se explica porque con los enunciados de primer orden no se puede expresar la propiedad de *ser un número natural*). También el problema de decisión en la aritmética con la suma, conocido como *aritmética de Presburger*, tiene un grado de dificultad muy grande; en este caso se requiere mínimamente tiempo doble exponencial.

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

6. Los problemas de optimización y las aproximaciones polinomiales

Los *problemas de optimización* son casos especiales de los problemas de búsqueda, y han sido una de las motivaciones principales para el desarrollo de la complejidad temporal en general y de la NP-completitud en particular. Un problema de optimización es un problema de búsqueda que consiste en obtener, dada una instancia, una *solución óptima* según una medida o costo que se asocia a las soluciones, de valor máximo (*problema de maximización*) o mínimo (*problema de minimización*), como por ejemplo una asignación de valores de verdad que satisfaga el máximo número de cláusulas de una fórmula booleana en forma normal conjuntiva, un cubrimiento de vértices de tamaño mínimo de un grafo, etc.

Obviamente, que un problema de optimización tenga resolución eficiente implica que el problema de decisión asociado también la tenga. Por ejemplo, si una MT M_1 resuelve eficientemente el problema en el que se busca encontrar un cubrimiento de vértices de tamaño mínimo de un grafo G, entonces existe una MT M_2 que resuelve eficientemente el problema de determinar si un grafo G tiene un cubrimiento de vértices de tamaño K: dados un grafo G y un número K, M_2 ejecuta M_1 sobre G y acepta sii M_1 devuelve un cubrimiento de vértices de G de tamaño a lo sumo K.

De esta manera, un problema de optimización cuyo problema de decisión asociado sea NP-completo no puede resolverse eficientemente, a menos que P = NP (es el caso del problema anterior). Y entonces, a los fines prácticos, para resolver un problema de optimización con un problema de decisión asociado NP-completo se suele recurrir, cuando existe, a una *aproximación polinomial*, que es un algoritmo de tiempo polinomial que obtiene para toda instancia del problema una solución *cercana* a la óptima, de acuerdo a un criterio determinado.

Formalmente, dado un problema de optimización A y una MT M de tiempo polinomial que resuelve el problema de búsqueda general correspondiente, si opt(w) es la medida de una solución óptima de una instancia w y m(M(w)) es la medida de la solución obtenida por M al ejecutarse a partir de w, se define que M es una *ε-aproximación polinomial* para A sii para toda instancia w se cumple $|m(M(w)) - opt(w)| / \max(m(M(w)), opt(w)) \leq \varepsilon$. El valor ε, conocido como *error relativo*, varía entre 0 y 1, y lo que se pretende es que sea lo más chico posible.

Se prueba que si P ≠ NP, existen problemas de optimización que no son aproximables polinomialmente, es decir que no cuentan con ε-aproximaciones polinomiales para ningún ε. Un ejemplo es el problema del viajante de comercio en su versión de problema de optimización, consistente en encontrar en un grafo ponderado un circuito hamiltoniano cuya suma de sus arcos sea mínima. La clase de los problemas de optimización aproximables polinomialmente se denomina APX. El problema de optimización del cubrimiento de vértices de un grafo es uno de ellos. Algunos problemas de APX cuentan con ε-aproximaciones polinomiales para todo ε. Dichos problemas conforman la clase PAS de los *esquemas de aproximación polinomial*. Un ejemplo de problema de PAS es el *problema de la mochila*, que consiste en obtener, dados un conjunto C de m pares (v_i, p_i) y un número K, un subconjunto de C que maximice la suma de los v_i en tanto la suma de los p_i asociados no supere K. Llamando OP y ONP a las clases de los problemas de optimización asociadas a P y NP, respectivamente, se cumple que P ≠ NP implica las inclusiones $OP \subset PAS \subset APX \subset ONP$.

Como en las jerarquías que describimos antes, para poblar la jerarquía de los problemas de optimización se recurre a las reducciones y al concepto de completitud. En este caso se utilizan reducciones que preservan la propiedad de *aproximabilidad polinomial*, denominadas *APX-reducciones* (existen varios tipos), que por lo tanto permiten probar la no aproximabilidad polinomial de los problemas de optimización de ONP que sean completos con respecto a las mismas, a menos que P = NP. En comparación con los problemas NP-completos, los problemas ONP-completos con respecto a las APX-reducciones que se conocen al día de hoy son escasos, lo que se justifica por la mayor restrictividad de dichas reducciones, y porque los problemas de optimización asociados a muchos problemas NP-completos son aproximables polinomialmente. Un camino alternativo para obtener resultados en este ámbito es aplicando el Teorema PCP, que mencionamos en la sección 9.

7. Los problemas con resoluciones paralelas eficientes

Retomamos los problemas de decisión para describir una jerarquía temporal de problemas que se solapa con la clásica jerarquía espacio-temporal que presentamos en las secciones anteriores. Es la jerarquía NC (*Nick's Class*, por Nicholas Pippenger, quien la definió), dentro de la clase P, creada para estudiar los problemas que se pueden resolver mediante *algoritmos paralelos eficientes*, es decir, los problemas de P con algoritmos paralelos significativamente más rápidos que los algoritmos secuenciales con los que cuentan, de acuerdo a un criterio determinado.

El modelo computacional más sencillo y extendido empleado para dicho estudio es el *circuito booleano*, o directamente *circuito*, que es un grafo dirigido sin ciclos con vértices que representan variables booleanas $x_1, \ldots, x_n$, que pueden adoptar los valores 1 o 0, o bien puertas lógicas, correspondientes a operaciones de conjunción (*and*), disyunción (*or*) o negación (*not*). Los primeros vértices constituyen la entrada del circuito, sin arcos de entrada, y entre los otros vértices, con uno o dos arcos de entrada según sea la puerta lógica que representen, se encuentra el que constituye la salida del circuito, sin arcos de salida. Se define que un circuito C acepta una cadena de entrada w de 1 y 0 (cada dígito asignado a una variable booleana) si luego de aplicar a partir de ella las puertas lógicas representadas, obtiene como salida el valor 1, lo que se expresa con $C(w) = 1$.

La cantidad de vértices de un circuito determina su *tamaño*, y la longitud del camino más largo desde uno de sus vértices de entrada a su vértice de salida determina su *profundidad*. Intuitivamente, el tamaño de un circuito representa el número de procesadores de la máquina paralela que modeliza, y su profundidad, el tiempo de ejecución de la misma. Así, sólo los *circuitos polinomiales*, es decir los circuitos de tamaño polinomial, son físicamente realizables, que entonces no pueden resolver eficientemente ningún problema cuya resolución secuencial más rápida sea de tiempo mínimamente exponencial (en particular, ningún problema NP-completo, a menos que P = NP).

Como un circuito procesa una cantidad fija de variables de entrada, para contemplar cadenas de todos los tamaños se utilizan *familias de circuitos*. Formalmente, una familia de circuitos $\{C_n\}_{n\geq 1}$ es una secuencia infinita de circuitos C_n, cada uno de $n \geq 1$ vértices de entrada, que resuelve un problema A sii todo circuito C_n acepta todas y sólo las cadenas de A de tamaño n. Si además existe una MT de espacio logarítmico que para todo n construye la descripción del circuito C_n, la

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

familia de circuitos se dice *uniforme*. Tal condición de constructividad eficiente evita que puedan resolverse problemas muy difíciles con circuitos muy simples, incluso indecidibles, y por otra parte asegura que todos los circuitos implementen en conjunto un único algoritmo, como lo hace una MT. La descripción estándar de un circuito consiste en una secuencia de ternas ordenadas e identificadas adecuadamente, cada una con un operador lógico y a lo sumo dos operandos.

La clase de los problemas que se pueden resolver mediante *familias de circuitos polinomiales* se denomina $P_{/poly}$, y se prueba que cuando estas familias son uniformes, la subclase de problemas correspondiente no es otra que P. En este marco se define que un problema cuenta con un algoritmo paralelo eficiente, cuando se puede resolver mediante una familia uniforme de circuitos polinomiales con profundidad $O((\log_2 n)^k)$, con $k \geq 1$. Como dijimos, la jerarquía que reúne a estos últimos problemas se denomina NC. La integran infinitas clases NC^1, NC^2, …, con problemas que se pueden resolver a través de familias uniformes de circuitos polinomiales de profundidad $O(\log_2 n)$, $O((\log_2 n)^2)$, … La definición de NC, en lo que hace a su frontera dentro de la clase P, no está tan aceptada. No obstante, al igual que en otras jerarquías, en NC interesan principalmente los problemas de sus primeras clases.

Un típico ejemplo de problema con algoritmo paralelo eficiente es la suma de dos números; el algoritmo aprendido en la escuela es secuencial, de tiempo lineal, consistente en sumar desde los dígitos menos significativos y mantener un dígito de acarreo a lo largo de todo el cálculo, pero existe un algoritmo más eficiente, paralelo, de tiempo $O(\log_2 n)$, con $O(n)$ procesos, cada uno dedicado a una posición de dígito, y con comunicaciones que se ocupan de los acarreos. También cuentan con algoritmos paralelos eficientes las operaciones aritméticas elementales restantes; los ordenamientos y las búsquedas en los arreglos; la multiplicación, el determinante y la inversa de las matrices; la clausura transitiva y la búsqueda del camino mínimo en los grafos; etc. Todos estos ejemplos corresponden a problemas de búsqueda, resueltos con circuitos con varios vértices de salida, pero pueden ser llevados a la forma de problemas de decisión como mostramos previamente.

Por lo dicho antes, se cumple $NC \subseteq P$, y la conjetura más aceptada es que la inclusión es estricta, es decir que no todos los problemas de P cuentan con algoritmos paralelos eficientes. De esta manera, como NC es cerrada con respecto a las reducciones log-space (si $A \leq_{\log} B$ y $B \in NC$ entonces $A \in NC$), si un problema es P-completo con respecto a dichas reducciones en la práctica se asume que no pertenece a NC.

Un ejemplo de problema P-completo con respecto a las reducciones log-space es el *problema del máximo flujo en una red* (llevado a la forma de problema de decisión). El problema consiste en encontrar el flujo máximo posible en una red, representada por un grafo dirigido ponderado. El problema es muy representativo de los problemas con resolución eminentemente secuencial: definible su resolución en etapas, si bien éstas se pueden paralelizar satisfactoriamente, pueden llegar a sumar una cantidad mayor que un valor polilogarítmico. Otro problema P-completo con respecto a las reducciones log-space es el *problema de evaluación de circuitos*, que plantea establecer, dado un par (C, w) tal que C es la descripción de un circuito y w una cadena de 1 y 0, si $C(w) = 1$.

Se prueba que $NC^1 \subseteq LOGSPACE \subseteq NLOGSPACE \subseteq NC^2$, de lo que se deriva que los problemas P-completos no sólo no admitirían resoluciones eficientes paralelas con respecto al tiempo, sino

tampoco resoluciones eficientes secuenciales con respecto al espacio (esto último ya mencionado en la sección anterior). Inclusiones como éstas se pueden generalizar a distintos modelos paralelos con adecuadas restricciones (uniformidad, cantidad de procesadores, etc.), mediante la Tesis de la Computación Paralela, que sostiene que el tiempo paralelo se relaciona polinomialmente con el espacio secuencial: $PTIME(T(n)) \subseteq SPACE(p_1(T(n)))$ y $SPACE(S(n)) \subseteq PTIME(p_2(S(n)))$, siendo p_1 y p_2 dos polinomios y $PTIME(T(n))$ una clase de problemas con resoluciones de tiempo paralelo $O(T(n))$ considerando un determinado modelo computacional. El modelo computacional de las familias uniformes de circuitos polinomiales satisface la tesis.

Los circuitos también se consideran para encarar la resolución del problema P vs NP: se prueba que la gran mayoría de las funciones de las cadenas de 1 y 0 al conjunto {1, 0} no pueden ser computadas por circuitos de tamaño polinomial, lo que alienta a encontrar algún problema NP-completo asociado a una función de estas características, porque así se demostraría que $P \neq NP$. Desafortunadamente, hasta el momento no se han logrado avances importantes en este sentido, pero este camino de prueba sigue siendo de interés, máxime que se conjetura que los problemas NP-completos no se pueden resolver ni siquiera con familias de circuitos polinomiales no uniformes, es decir que ni siquiera pertenecerían a la clase $P_{/poly}$.

8. La complejidad temporal de los algoritmos probabilísticos

Los *algoritmos probabilísticos* constituyen un área de estudio muy importante en la complejidad computacional. Uno de los motivos, sobre el que nos concentramos en esta sección, es que resuelven muchos problemas de interés computacional de una manera más eficiente, o al menos más simple, que los algoritmos determinísticos con los que se cuenta (en este contexto, entenderemos por algoritmo determinístico uno no probabilístico). Históricamente, comenzaron a recibir especial atención después de los trabajos (Solovay y Strassen, 1977) y (Rabin, 1980), que exhibían resoluciones eficientes del problema de primalidad.

Lo que caracteriza a un algoritmo probabilístico es que en ciertos pasos efectúa una *elección aleatoria*, que determina una continuación entre varias alternativas posibles. Esto hace que con una misma entrada, en distintas ejecuciones pueda generar distintas salidas. Dado este comportamiento, sus aceptaciones y rechazos se definen en base a un análisis probabilístico sobre la distribución de sus respuestas (análisis distinto al que mencionamos antes, basado en la distribución de las entradas, con otro propósito que es el de calcular la complejidad computacional del caso promedio).

El modelo computacional que se utiliza es la *máquina de Turing probabilística* (abreviada con MTP). Formalmente, una MTP:

- A diferencia de las MT determinísticas (MT no probabilísticas), a partir de la cadena de entrada avanza paso a paso siempre eligiendo aleatoriamente una entre dos continuaciones posibles, más allá de que pueda hacer lo mismo en las dos, siendo toda elección independiente de la anterior y de probabilidad de ocurrencia 1/2 (por eso se la refiere como un *lanzamiento de moneda*).
- Al igual que las MT determinísticas, termina aceptando o rechazando la cadena de entrada.

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

El criterio de aceptación se basa en la relación entre la cantidad de computaciones de aceptación y la cantidad total de computaciones, que vamos a precisar enseguida, y el tiempo de ejecución y el espacio ocupado se definen como antes. Notar que de acuerdo a la definición, la *sintaxis* de una MTP es la misma que la de una MT no determinística, incluyendo la restricción de las dos continuaciones, dado que para toda MT no determinística M_1 existe una MT no determinística M_2 equivalente con a lo sumo dos continuaciones por estado y símbolo. En cambio, su *semántica* es bien distinta: sus bifurcaciones son equiprobables, y sus aceptaciones se basan en un criterio probabilístico. Además, la MTP, a diferencia de la MT no determinística, constituye un modelo computacional físicamente realizable; de hecho, cualquier típico lenguaje de programación incluye un *generador aleatorio* de números.

La clase de problemas que se toma como la clase de los problemas tratables en el ámbito de los algoritmos probabilísticos es BPP (*tiempo polinomial probabilístico acotado*), considerada así la *clase P probabilística*. Dadas una MTP M y una cadena de entrada w, denotando con $\alpha(M, w)$ a la relación entre la cantidad de computaciones de aceptación y la cantidad total de computaciones de M a partir de w, es decir la *probabilidad de que M acepte w*, y con $\beta(M, w)$ a la relación entre la cantidad de computaciones de rechazo y la cantidad total de computaciones de M a partir de w, es decir la *probabilidad de que M rechace w*, se define que un problema A pertenece a BPP sii existe una MTP M de tiempo polinomial que cumple, para toda cadena w:

- Si w pertenece a A, entonces $\alpha(M, w) \geq 2/3$.
- Si w no pertenece a A, entonces $\beta(M, w) \geq 2/3$.

La cota 2/3 se puede reemplazar por otras, mayores que 1/2, por la posibilidad, según la *cota de Chernoff* de la teoría de las probabilidades, de reducir la probabilidad de error efectuando varias ejecuciones independientes y quedándonos con la respuesta mayoritaria,

Ejemplos clásicos de problemas de la clase BPP son el problema de primalidad, ya referido, y el *problema de composicionalidad*, consistente en determinar si un número es *compuesto*, es decir no primo. Otros algoritmos probabilísticos clásicos de BPP son el de la *búsqueda de la mediana*, en su forma de problema de decisión (de tiempo lineal como los algoritmos determinísticos más rápidos conocidos pero más simple), y el *testeo de igualdad de dos polinomios multivariados* (al día de hoy no se le conoce una resolución determinística eficiente).

La conjetura más aceptada actualmente no es la más intuitiva: P = BPP. Efectivamente, de investigaciones recientes se ha adquirido mucha evidencia, considerando ciertas asunciones, de que hay un modo de *desaleatorizar*, es decir transformar en un algoritmo determinístico, todo algoritmo probabilístico correspondiente a la clase BPP, con un retardo sólo polinomial. El componente central de dichos trabajos es el *generador pseudoaleatorio*, algoritmo determinístico capaz de generar grandes cadenas que parecen aleatorias amplificando pequeñas cadenas aleatorias conocidas como *semillas*. Como lograr verdadera aleatoriedad es en general difícil, en los algoritmos probabilísticos es común utilizar estos generadores como fuentes aleatorias sustitutas. Se han obtenido generadores pseudoaleatorios muy sofisticados, que asumiendo la existencia de las funciones de un solo sentido que mencionamos antes, permiten producir cadenas indistinguibles de las cadenas aleatorias en chequeos de tiempo polinomial.

Asumiendo P = NP, se prueba que BPP = P. No se sabe si BPP $\subseteq$ NP, pero sí que BPP $\subseteq P_{/poly}$. También se cumple BPP $\subseteq$ EXP: BPP se puede definir alternativamente, como la clase NP, con MT determinísticas que reciben una cadena adicional, en este caso con dígitos 1 y 0 que representan lanzamientos de moneda consecutivos; de esta forma, como en tiempo polinomial p(n) se deben considerar $2^{p(n)}$ cadenas de este tipo, toda MTP que resuelve un problema de BPP puede simularse en tiempo exponencial con una MT determinística de las características señaladas. No se conocen problemas completos en BPP (que existen, obviamente, si BPP = P). La dificultad para encontrarlos se relaciona con la naturaleza semántica del criterio de aceptación de las MTP.

9. La complejidad temporal de las pruebas interactivas probabilísticas

La definición de los problemas de la clase NP con verificadores eficientes y certificados sucintos, resalta la importancia de poder chequear eficientemente una prueba, independientemente del esfuerzo que se necesite para construirla.

Considerando en particular la caracterización que hicimos de un problema A de NP en términos de un *verificador* V (MT de tiempo polinomial), tal que para toda cadena de entrada w, si w pertenece a A entonces existe un probador P (MT de poder ilimitado) que puede convencer a V con un certificado sucinto de que w pertenece a A, y si w no pertenece a A entonces ningún probador P tiene chance alguna de convencer a V de lo contrario cualquiera sea el certificado sucinto que utilice, una pregunta razonable para hacerse es si con varias interacciones entre un probador y un verificador podría ampliarse la clase NP a una clase mayor. Intuitivamente pareciera que sí. Por ejemplo, en el plano real, es razonable pensar que un profesor, al explicar a sus alumnos un teorema, tiene más chance de que lo entiendan si permite que le hagan preguntas durante su exposición en lugar de postergarlas hasta el final, porque así se evitaría tener que contar por adelantado con todas las respuestas a todas las preguntas posibles. Sin embargo, esto no ocurre: un problema de NP también se puede caracterizar mediante un modelo interactivo, con una cantidad polinomial de mensajes sucintos de ida y de vuelta entre un probador de poder ilimitado y un verificador de tiempo polinomial.

¿Y si potenciamos al verificador con elecciones aleatorias, es decir, si lo convertimos en una MT probabilística de tiempo polinomial? ¿Ahora sí podemos saltar a una clase de problemas mayor? En efecto, en este caso la respuesta es afirmativa. No sólo eso, sino que el salto es mayúsculo, porque alcanzamos la clase PSPACE, lo que nuevamente desafía a la intuición: a pesar de que todo algoritmo probabilístico eficiente podría sustituirse por un algoritmo determinístico eficiente, y de que las pruebas con interacciones como las que acabamos de describir no agregan problemas a la clase NP, al combinar lo probabilísitco con lo interactivo pareciera obtenerse un modelo computacional mucho más potente que ambos paradigmas considerados por separado, conocido como *sistema de pruebas interactivas probabilísticas*, o directamente *sistema de pruebas interactivas*.

Formalmente, se define que un problema A pertenece a la clase IP (*pruebas interactivas*) si *admite* una prueba interactiva (P, V), es decir una prueba con interacciones entre un probador P y un verificador V que termina con una respuesta de V y satisface las siguientes condiciones:

- V es una MTP de tiempo polinomial.

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

- Para toda cadena de entrada w se cumple que si w pertenece a A, entonces existe una MT P de poder ilimitado tal que (P, V) acepta w con una probabilidad al menos 2/3, y si w no pertenece a A, entonces para toda MT P de poder ilimitado, (P, V) rechaza w con la misma probabilidad.
- Los intercambios entre P y V constan de una cantidad polinomial de cadenas de tamaño polinomial con respecto al tamaño de las cadenas de entrada w.

Como en la definición de la clase BPP, la cota 2/3 se puede reemplazar por otra, ahora mayor que 1/2, por el recurso de las múltiples ejecuciones independientes que permiten reducir la probabilidad de error.

Un ejemplo de problema de la clase IP es el complemento del problema ISO, consistente en decidir si dos grafos no son isomorfos. Es decir, si bien con verificadores determinísticos los certificados naturales del problema son de tamaño exponencial, dado que tienen que incluir las m! permutaciones correspondientes al conjunto de vértices {1, …, m}, con verificadores probabilísticos los certificados son sucintos, permitiendo establecer la pertenencia o no pertenencia de una cadena al problema con una probabilidad de error muy baja.

Así como a la clase BPP se la considera la clase P probabilística, a la clase IP se la considera la *clase NP probabilística*. En este caso, la conjetura más aceptada es que IP incluye estrictamente a NP. Del estudio de IP se han derivado importantes resultados útiles para la criptografía y las aproximaciones polinomiales, entre otras áreas. Como anticipamos, se cumple IP = PSPACE, igualdad para nada intuitiva, teniendo en cuenta, por ejemplo, que IP incluye al complemento del problema QSAT, cuando el complemento del problema SAT ni siquiera estaría en NP.

Una característica importante para analizar en una prueba interactiva es cuánto *aprende* de ella el verificador, o en otras palabras, qué información adicional adquiere además de la pertenencia de una cadena a un problema. Un caso extremo es que el verificador reciba directamente una solución (una permutación de vértices, una asignación de valores de verdad, etc). Y el otro extremo es que el verificador no aprenda absolutamente nada, que la prueba interactiva sea de *conocimiento cero*, como se la conoce, propiedad sumamente relevante en áreas como la criptografía, por ejemplo para los procesos de autenticación. Demostrar que una prueba es de conocimiento cero requiere verificar que lo que el verificador puede aprender interactuando con un probador, también lo puede aprender aisladamente, es decir, que lo mismo que se puede obtener con alguna estrategia de verificación interactiva de tiempo polinomial probabilístico, se siga o no el protocolo establecido en la prueba interactiva, se puede obtener con alguna estrategia no interactiva del mismo tiempo de ejecución. Asumiendo entre otras cosas la existencia de las funciones de un solo sentido, se prueba que todo problema de NP cuenta con una prueba de conocimiento cero.

Todavía existe otra variante de verificaciones probabilísticas eficientes que caracteriza a la clase NP. Se trata de las *pruebas chequeables probabilísticamente*, con las que se vuelve a la idea de la exposición completa desde el principio de una prueba x (cadena de 1 y 0) de una cadena de entrada w, con un verificador probabilístico eficiente V y sin ningún probador P, pero ahora con la restricción de que V puede acceder sólo a algunas partes de x. Más en detalle, V puede utilizar una cantidad determinada de dígitos 1 y 0 aleatorios, y puede hacer una cantidad determinada de lecturas independientes de un solo dígito sobre x. En base a esta variante se define una familia de

clases de problemas denominadas PCP (r(n), q(n)). Formalmente, un problema A pertenece a la clase PCP (r(n), q(n)), o lo que es lo mismo, *admite* una prueba chequeable probabilísticamente de tipo (r(n), q(n)), siendo r y q dos funciones de los números naturales a los números naturales, sii existe un verificador V de tiempo probabilístico polinomial tal que, para toda cadena w, con |w| = n, cumple lo siguiente:

- Sobre una prueba x de dígitos 1 y 0, utilizando O(r(n)) dígitos 1 y 0 aleatorios, efectúa O(q(n)) lecturas independientes de un solo dígito, luego de lo cual acepta o rechaza.
- Si w pertenece a A, entonces existe una prueba x tal que la probabilidad de que acepte w es 1.
- Si w no pertenece a A, entonces para toda prueba x, la probabilidad de que acepte w es a lo sumo 1/2.

La constante 1/2 se puede reemplazar por cualquier otra, recurriendo como siempre a las múltiples ejecuciones independientes. Se prueba (Teorema PCP) que todo problema de NP admite una prueba chequeable probabilísticamente de tipo ($\log_2 n$, 1), caracterización de NP contraintuitiva: considerando un sistema axiomático determinado con pruebas verificables eficientemente, el problema que consiste en establecer, dados una fórmula ψ y un número K, si existe una prueba de ψ de a lo sumo K pasos, pertenece a NP (en una famosa carta, K. Gödel le preguntaba a J. von Neumann si acaso podría directamente construirse en tiempo polinomial una prueba de esas características en el marco de la lógica de predicados, con la idea de que en dicho caso la indecidibilidad del Entscheidungsproblem no era tan grave); pero entonces, por el Teorema PCP, el problema cuenta con certificados en los que es suficiente examinar unos pocos dígitos, certificados tan válidos como las pruebas que deben revisarse paso por paso.

Más allá de la última observación, producto de un modelo de computación ciertamente especial, el Teorema PCP es muy importante en la complejidad computacional por permitir derivar resultados negativos en el área de las aproximaciones polinomiales, en general difíciles de obtener como comentamos en la sección 6. La prueba típica de no aproximabilidad polinomial de un problema de optimización consiste en generar, mediante una reducción polinomial, instancias tales que la brecha entre la medida de una solución óptima cuando la instancia original pertenece al problema de decisión asociado, y la medida de una solución óptima cuando la instancia original no pertenece a dicho problema. es muy grande. En este sentido, las pruebas chequeables probabilísticamente de los problemas de NP facilitan la tarea, por su vinculación con los problemas de optimización. La idea general consiste en relacionar la brecha que una prueba chequeable probabilísticamente requiere entre la probabilidad de aceptar correcta e incorrectamente, con la brecha entre las medidas de las soluciones óptimas de las instancias que la reducción correspondiente referida antes debe generar. De esta forma, por ejemplo, se puede probar que el problema de búsqueda de un clique de tamaño máximo no es aproximable polinomialmente, y que el problema de búsqueda de un cubrimiento de vértices de tamano mínimo no se puede aproximar polinomialmente con cualquier ε

10. La complejidad temporal de los algoritmos cuánticos

A comienzos de los años 1980, R. Feynman planteó la necesidad de construir computadoras cuánticas, por la posible incapacidad de las computadoras clásicas (no cuánticas) para simular eficientemente los procesos cuánticos. Así surgió el estudio de este tercer paradigma en la

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

complejidad computacional, centrado en un modelo computacional, la *máquina cuántica*, que podría refutar la Tesis Fuerte de Church-Turing mencionada antes, dado que permite resolver eficientemente algunos problemas que hoy día los mejores algoritmos clásicos resuelven en tiempo mínimamente exponencial.

Entre las características de la mecánica cuántica que adopta este modelo computacional se destaca la de que todo parámetro físico de una partícula elemental como el electrón (posición, energía, etc.), *en tanto no se mida* no tiene como valor un simple número sino una *onda de probabilidad*, interpretada como la *superposición* de todos los valores posibles. Recién al medirse el parámetro adquiere un valor definitivo, lo que se conoce como *colapso de la onda de probabilidad*. En realidad, las ondas de probabilidad se asocian no a partículas aisladas sino a conjuntos de partículas, que pueden alcanzar cantidades exorbitantes. Aún en este caso se considera que la máquina cuántica es físicamente realizable, asumiendo que se la puede aislar suficientemente para evitar interferencias indeseables (*ruido*), y complementar con mecanismos de corrección de errores adecuados.

En la máquina cuántica, el objeto análogo al dígito binario o *bit* de la máquina clásica es el *dígito binario cuántico* o *cúbit*. Como los bits, los cubits pueden estar en el estado básico 0 o en el estado básico 1, pero a diferencia de los bits, los cubits también pueden estar en un *estado de superposición* de los dos estados básicos, o dicho de otra forma, pueden estar en el estado 0 *o* en el estado 1, con una determinada probabilidad. Los estados básicos de un cúbit se denotan con $|0>$ y $|1>$, y los estados de superposición con $\alpha_0|0> + \alpha_1|1>$, combinación lineal en la que las α_i son *amplitudes*, números complejos que satisfacen la ecuación $|\alpha_0|^2 + |\alpha_1|^2 = 1$. De este modo, el estado de un cúbit se puede representar con un *vector de amplitudes* (α_0, α_1). En concordancia con lo dicho previamente, un cúbit permanece en un estado de superposición hasta que es medido, momento en el que pasa al estado $|0>$ con probabilidad $|\alpha_0|^2$ o al estado $|1>$ con probabilidad $|\alpha_1|^2$, y las amplitudes resultantes quedan, según el caso, con los valores $\alpha_0 = (1, 0)$ y $\alpha_1 = (0, 0)$ o $\alpha_0 = (0, 0)$ y $\alpha_1 = (1, 0)$. Ya con amplitudes reales se observan los efectos cuánticos. En adelante, para simplificar la presentación, las consideramos exclusivamente.

Los cubits se agrupan en *registros cuánticos*. De lo anterior se deriva que un registro cuántico de m cubits puede estar en una superposición de 2^m estados básicos. Por ejemplo, un registro cuántico de dos cubits puede estar en un estado básico $|00>$, $|01>$, $|10>$ o $|11>$, o en un estado de superposición $\alpha_{00}|00> + \alpha_{01}|01> + \alpha_{10}|10> + \alpha_{11}|11>$, con $\alpha_{00}^2 + \alpha_{01}^2 + \alpha_{10}^2 + \alpha_{11}^2 = 1$. En este último caso, el estado de superposición del registro se representa mediante el vector de amplitudes $(\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11})$, y cuando el registro es medido pasa con probabilidad α_{d1d2}^2 al estado $|d_1d_2>$, siendo $d_i = 0$ o 1, quedando la amplitud correspondiente en 1 y las restantes en 0. Lo mismo sucede con los registros cuánticos de tres cubits, cuatro cubits, etc. En todos los casos, las coordenadas de los vectores de amplitudes se disponen según el orden lexicográfico, y a toda medición le sigue el colapso de la superposición. Cabe remarcar que el estado de un registro clásico de m bits de un algoritmo probabilístico también se puede representar con un vector de dimensión 2^m, asociando cada coordenada a la probabilidad de que el registro incluya una cadena determinada. Lo que en verdad distingue a las máquinas cuánticas de las MTP son sus *interferencias destructivas*, producto de amplitudes negativas, que serían las que les dan una capacidad computacional superior.

Los registros cuánticos pasan de un estado a otro por medio de *operaciones cuánticas*, que se aplican sobre uno o más cubits. Una manera de representar las operaciones cuánticas es

mediante matrices, de modo tal que la aplicación de una operación cuántica sobre un registro cuántico consiste en multiplicar la matriz que representa la operación por el vector que representa el estado del registro, dando como resultado otro vector que representa el nuevo estado del registro. Las matrices tienen que ser *unitarias*, es decir que al multiplicarlas por sus transpuestas (transpuestas conjugadas en el caso más general de los números complejos) se debe obtener la matriz identidad. Esta restricción es inevitable para que se cumpla la propiedad de que los cuadrados de las amplitudes sumen 1, y hace *reversibles* a las operaciones: aplicando una operación y luego la operación correspondiente a la matriz transpuesta de la primera, se vuelve al estado original. Esto también puede suceder a veces repitiendo una misma operación cuántica. Las operaciones cuánticas también se conocen como *puertas cuánticas*, porque se pueden representar alternativamente mediante *circuitos cuánticos*. Se puede definir una gran variedad de operaciones cuánticas, que por razones de factibilidad técnica sólo se consideran sobre conjuntos de hasta tres cubits. Por lo tanto, se representan con matrices de 2 x 2, 4 x 4 u 8 x 8.

Una computación cuántica, entonces, es una secuencia de puertas cuánticas aplicadas sobre un registro cuántico con una cadena de entrada w. La computación acepta o rechaza w según, medido al final el registro, el estado del cúbit designado para guardar el resultado es $|1>$ o $|0>$, respectivamente. En lo que hace a la complejidad temporal, se define que un problema A se puede resolver en tiempo cuántico T(n) sii existe una MT M de tiempo polinomial, que para todo n genera la descripción de un conjunto de puertas cuánticas que permiten aceptar, con probabilidad de error a lo sumo 1/3 (cota que como siempre se puede sustituir por otra), todas y sólo las cadenas w de tamaño n de A, por medio del siguiente proceso canónico:

- Inicializar un registro cuántico de m cubits de la forma $|w0^{m-n}>$, con $m \leq T(n)$.
- Aplicar al registro, una tras otra, T(n) puertas cuánticas.
- Medir el registro, y responder de acuerdo al estado del cúbit correspondiente al resultado.

Los problemas que se pueden resolver en tiempo cuántico polinomial conforman la clase BQP (*tiempo polinomial cuántico con error acotado*), considerada así la *clase P cuántica*. La MT M de tiempo polinomial de la definición anterior se requiere por la misma necesidad de uniformidad que mencionamos cuando definimos la clase NC en la sección 7. Justamente, otra forma de definir BQP es mediante *familias uniformes de circuitos cuánticos polinomiales*. Formalmente, se define alternativamente que un problema A pertenece a BQP sii existe una familia uniforme de circuitos cuánticos polinomiales $\{C_n\}_{n\geq 1}$, tal que para toda cadena w de tamaño n, C_n acepta o rechaza $|w0\ldots0>$ (se dejan cubits en $|0>$ como espacio de trabajo) con probabilidad de error a lo sumo 1/3, según w pertenezca o no a A, respectivamente. Ahora las puertas de los circuitos no son lógicas sino cuánticas. En cuanto a la variedad de las puertas cuánticas, se demuestra que un grupo reducido es suficiente para describir cualquier algoritmo cuántico; por ejemplo, las puertas denominadas *de Hadamard* y *de Toffoli*, de la misma forma que alcanzan las puertas *and* y *not*, u *or* y *not*, en el caso de los circuitos booleanos.

Actualmente se conocen pocos algoritmos cuánticos mejores que sus contrapartes clásicas, en el sentido de la complejidad temporal, lo que puede explicarse por la dificultad para diseñarlos, y también por nuestra intuición mucho más cercana a los algoritmos clásicos. Se destacan básicamente tres grupos de algoritmos: un primer grupo reúne a los algoritmos de búsqueda, basados en el *algoritmo de Grover* (Grover, 1996), que aceleran cuadráticamente los algoritmos de búsqueda clásicos más veloces; en el segundo grupo se ubican los algoritmos que utilizan la

Revista Abierta de Informática Aplicada/vol 8 Nº1 (2024): 109-139
ISSN 2591-5320

transformada cuántica de Fourier, como por ejemplo el *algoritmo de Shor* para factorizar (Shor, 1994), en este caso con mejoras temporales más sustanciales, del orden exponencial; y el tercer grupo corresponde al de las simulaciones de los sistemas cuánticos. Estos ejemplos corresponden a problemas de búsqueda, resueltos con circuitos cuánticos que devuelven secuencias de cubits, pero se pueden adaptar, como hicimos antes, a problemas de decisión.

Se cumple $P \subseteq BQP$, porque las MT de tiempo polinomial se pueden simular con circuitos booleanos polinomiales, que a su vez se pueden simular con circuitos cuánticos polinomiales. También se cumple $BPP \subseteq BQP$, porque los lanzamientos de moneda se pueden simular con aplicaciones de la puerta de Hadamard. No se cumpliría la inclusión recíproca $BQP \subseteq BPP$, teniendo en cuenta que no se ha encontrado hasta el momento ningún algoritmo probabilístico eficiente para el problema de factorización. Otra inclusión que se prueba es $BQP \subseteq PSPACE$, recurriendo a un procedimiento recursivo y la reutilización de espacio. Con respecto a la relación entre las clases BQP y NP, la conjetura más aceptada es que son incomparables: los algoritmos cuánticos sólo podrían acelerar en forma cuadrática los algoritmos clásicos más rápidos correspondientes a los problemas NP-completos, al tiempo que existe un lenguaje en BQP que no estaría en NP.

11. Observaciones finales

La complejidad computacional, y también la computabilidad, nos ayudan a entender por qué algunos problemas son fáciles y otros son difíciles. El grado de dificultad de un problema proviene de su estructura, y por eso conocerla en profundidad permite resolver el problema de la manera más eficiente posible.

En los últimos años se han logrado importantísimos descubrimientos relacionados con la criptografía, las pruebas matemáticas (probabilísticas, interactivas, de conocimiento cero, chequeables probabilísticamente), la teoría del aprendizaje automático, la computación cuántica, etc. Sin embargo, la complejidad computacional, y más en general la algorítmica, aún están en su infancia, sólo se han logrado responder pocas preguntas derivadas de su estudio.

Por razones de espacio, nos hemos limitado a enumerar ejemplos de problemas de distintas clases temporales y espaciales y relaciones entre las mismas. Los siguientes son excelentes libros en los que se pueden encontrar las pruebas correspondientes: (Bovet y Crescenzi, 1994), (Papadimitriou, 1994), (Goldreich, 2008), (Arora y Barak, 2009) y (Moore y Mertens, 2011). Sobre los algoritmos cuánticos, específicamente, recomendamos (Nielsen y Chuang, 2010) y (Aaronson, 2013a). Para profundizar en las distintas secciones del artículo también se pueden consultar, del autor de este artículo: (Rosenfeld e Irazábal, 2010), (Rosenfeld e Irazábal, 2013) y (Rosenfeld, 2024).

Otra lectura que recomendamos es (Aaronson, 2013b), ensayo cuyo objetivo es ilustrar cómo la filosofía se puede enriquecer con los resultados de la complejidad computacional. Incluye discusiones sobre la inteligencia artificial fuerte, el computacionalismo, el problema de la inducción, la evolución, el comportamiento de la economía y el problema lógico de la omnisciencia.

Referencias

» Aaronson, S. (2013a). *Quantum computing since Democritus*. Cambridge University Press.

» Aaronson, S. (2013b). *Why Philosophers Should Care About Computational Complexity*. En Computability: Gödel, Turing, Church, and Beyond, MIT Press, 2013.

» Agrawal, M., Kayal, N. y Saxena, N. (2004). *Primes is in P*. Annals of Mathematics, 160(2), 781-793.

» Arora, S. y Barak, B. (2009). *Computational complexity: a modern approach*. Cambridge University Press.

» Babai, L. (2016). *Graph isomorphism in quasipolynomial time*. STOC '16, Proc. of the 48th annual ACM Symp. on Theory of Computing, 684-697.

» Bovet, D. y Crescenzi, P. (1994). *Introduction to the theory of complexity*. Prentice-Hall.

» Church, A. (1936a). *An unsolvable problem of elementary number theory*. American Journal of Mathematics, 58(2), 345-363.

» Church, A. (1936b). *A note on Entscheidungsproblem*. The Journal of Symbolic Logic, (1)1, 40-41.

» Cook, S. (1971). *The complexity of theorem-proving procedures*. Proc. of the 3rd IEEE Symp. on the Foundations of Computer Science, 151-158.

» Gasarch, W. (2002). *The P =? NP poll*. ACM SIGACT News, 33(2), 34-47.

» Gasarch, W. (2012). *The second P =? NP poll*. ACM SIGACT News, 43(2), 53-77.

» Gasarch, W. (2019). *The third P =? NP poll*. ACM SIGACT News, 50(1), 38-59.

» Gödel, K. (1931). *Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I*. Monatshefte fur Math, Und Physik, 38, 173-198. En español: *Sobre sentencias formalmente indecidibles de Principia Mathematica y sistemas afines*. Obras Completas de Kurt

» Goldreich, O. (2008). *Computational complexity: a conceptual perspective*. Cambridge University Press.

» Grover, L. (1996). *A fast quantum mechanical algorithm for database search*. STOC, 212-219, ACM.

» Immerman, N. (1988). *Nondeterministic space is closed under complementation*. SIAM Journal on Computing, 17, 935-938.

» Ladner, R. (1975). *On the structure of polynomial-time reducibility*. JACM, 22, 155-171.

» Levin, L. (1973). *Universal'nyie perebornyie zadachi*. Problemy Peredachi Informatsii, 9, 115,116. En inglés: *Universal search problems*. Problems of Information Transmission, 9, 265-266.

» Moore. C. y Mertens, S. (2011). *The nature of computation*. Oxford University Press.

» Nielsen, M. y Chuang, I. (2010). *Quantum computation and quantum information*. Cambridge University Press.

» Papadimitriou, C. (1994). *Computational complexity*. Addison-Wesley.

» Rabin, M. (1980). *A probabilistic algorithm for testing primality*. Journal of Number Theory, 12(1), 128-138.

» Reingold, O. (2005). *Undirected ST-connectivity in log-space*. STOC, 376-385, ACM.

» Rosenfeld, R. (2024). *Computabilidad y complejidad computacional (versión preliminar)*. EDULP.

» Rosenfeld, R. e Irazábal, J. (2010). *Teoría de la computación y verificación de programas*. EDULP, McGraw-Hill.

» Rosenfeld, R e Irazábal, J. (2013). *Computablidad, complejidad computacional y verificación de programas*. EDULP.

» Savitch, W. (1970). *Relationship between nondeterministic and deterministic tape complexities*. Journal

of Computer and System Sciences, 4, 177-192.

» Shor, P. (1994). *Algorithms for quantum computation: discrete logarithms and factoring*. Proc. 35th Annual Symp. on Foundations of Computer Science, IEEE Press.

» Solovay, R. y Strassen,V. (1977). *A fast Monte-Carlo test for primality*. SIAM Journal on Computing, 6, 84-85.

» Stockmeyer, L. y Meyer, A. (1973). *Word problems requiring exponential time*. Proc. 5th ACM Symp. on the Theory of Computing, 1-9.

» Turing, A. (1936). *On computable numbers, with an application to the Entscheidungsproblem*. Proc. London Mathematical Society, 2(42), 230-265.