Problemas NP-Completo  
  
Introducción

El concepto de problemas NP-Completo no es algo nuevo. Su concepción remonta a la década de los 70, y desde ese momento hasta el día de hoy, ha sido un tema sujeto a varios debates y posturas diferentes entre la comunidad de científicos de la computación.   
  
Los problemas de este tipo no son más que problemas de decisión en los que sus soluciones pueden ser verificadas en tiempo polinómico por un algoritmo. Sin embargo, no existen o se conocen algoritmos que encuentren dichas soluciones en un tiempo equivalente.

El concepto en sí no es lo que llevó a décadas de debate, lo que provocó tal hecho es la siguiente pregunta:

**¿Ser capaz de reconocer las respuestas correctas rápidamente, implica que haya una forma igualmente rápida de encontrar las respuestas?**

O, en términos de complejidad computacional, **¿es P = NP?**  
Esta pregunta es uno de los problemas de ciencias de la computación que aún siguen sin resolverse. La postura de varios científicos de la comunidad es variada, aunque existe una tendencia a declarar que P ≠ NP.  
  
El impacto que tendría la comprobación de dicha igualdad sería muy grande, tanto que si se comprobase, la criptografía basadas en claves públicas resultaría imposible, la optimización de problemas se volvería extremadamente fácil, e implicaría una gran simplificación de las matemáticas.   
  
Pero para incurrir en este tema, primero debemos explicar sus orígenes, y zambullirnos (preferiblemente, de forma no tan profunda) en la *Teoría de la Complejidad Computacional.*

**Teoría de la Complejidad Computacional**

La teoría de la complejidad computacional se centra en la clasificación de problemas computacionales de acuerdo a su dificultad inherente. Los problemas computacionales en sí no son más que tareas a ser resueltas por una computadora, bajo un cierto conjunto de reglas y pasos a seguir (es decir, resueltos mediante un algoritmo).   
  
La dificultad de dichos problemas está basada en la cantidad de recursos necesarios para resolverlos, cualesquiera que el algoritmo haya empleado en la resolución del mismo. Si bien existen diferentes medidas utilizadas para calcular la complejidad, las más prevalentes al momento de tratar con algoritmos son el ***tiempo*** y el ***espacio***.  
  
Es bajo estos criterios que se realiza una clasificación (llamada *Clase de Complejidad*) de los diversos problemas que existen en torno a la computación. La cantidad de clasificaciones y las relaciones entre ellas es tan amplia y variada (y confusa, de a ratos) como la cantidad de problemas (y sus variantes dificultades) que existen.  
  
En este documento solo se hará ilusión a unas pocas de estas clasificaciones, mayormente centradas en los Problemas de Decisión.

**Problemas de Decisión**

Un problema de decisión es aquel que puede ser presentado como una pregunta del tipo “Sí-No”, dependiendo de cuáles fuesen son valores de entrada. Uno de los ejemplos más comunes es decidir si un número natural dado es primo. La respuesta se reduce simplemente a un sí o un no.  
  
Sin embargo, por más que parezcan fáciles, los problemas de decisión pueden volverse altamente complicados y escalar en dificultad dependiendo de qué tan grande sea el conjunto de datos con el que estemos trabajando.   
  
Por ejemplo, conocemos y actualmente contamos con algoritmos que pueden calcular correcta y eficientemente los factores de un número entero. Sin embargo, estos algoritmos empiezan a fallar cuando el número entero a calcular se vuelve inmensamente grande. Por lo que la pregunta: ¿posee este número entero un factor? Ya no puede responderse con tanta facilidad, a pesar de sus respuestas sigan siendo sí y no.  
  
**Procedimiento de Decisión**

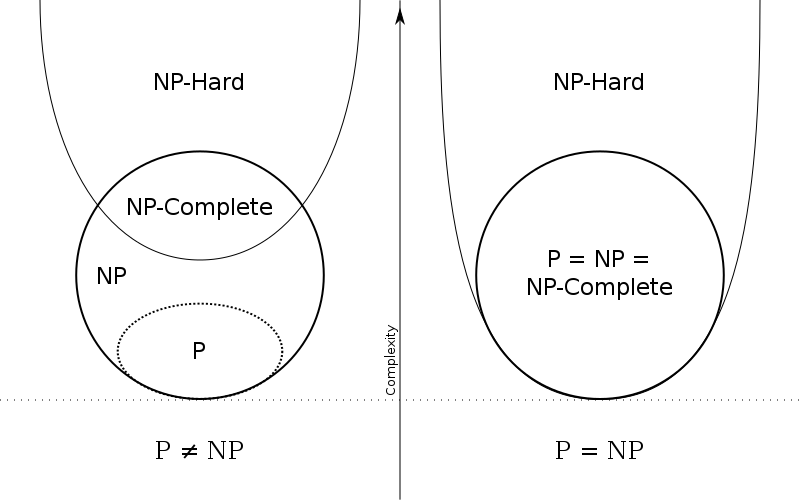
El procedimiento de decisión no es más un método para resolver el problema. Los procedimientos de decisión son los algoritmos empleados para resolverlos.

Si un algoritmo puede resolver el problema de forma correcta (es decir, la solución que arroja es verificada y concuerda), se dice que el problema es **decidible**. Son estos tipos de problemas los que la Teoría de Complejidad Computacional toma en cuenta para realizar las clasificaciones en diferentes clases de complejidad.  
  
En cambio, resulta imposible construir un algoritmo que siempre conduzca a una respuesta *sí o no* correcta, el problema se clasifica como **indecidible**. Un ejemplo de problema indecidible es el Problema de la parada, el cual en términos simples afirma que no existe una manera automática computable de saber si todos los programas del mundo terminan.

**Complejidad NP**

La complejidad NP es una de las clases definidas en la **TCC** usada para clasificar a los Problemas de Decisión. NP es el conjunto de problemas de decisión en los cuales las instancias del problema donde la respuesta es un “sí” tienen pruebas verificables en un tiempo polinómico.

Una definición equivalente de NP es el conjunto de problemas que pueden ser resueltos por una Máquina de Turing no-determinística\* (o equivalentemente, algoritmos no determinísticos). De ahí surge el nombre completo de la clase: *non-deterministic polynomial time.*



*Diagrama que muestra la división de las clases. Cortesía de Wikipedia.*

En términos simples, la clasificación NP engloba a aquellos problemas que tengan una difícil solución, pero un método fácil por el cual chequear esa solución.

Es por esto que la clasificación NP engloba a la **clase P**. La clase P son aquellos problemas de decisión que pueden solucionarse en un tiempo polinómico. Si pueden solucionarse en un tiempo polinómico, la verificación del problema resulta ser bastante simple: es la misma solución.

Los problemas **NP-Completo**, en cambio, no poseen una solución que pueda darse en tiempo polinómico, pero sí pueden verificarse fácilmente en ese tiempo.

Los problemas **NP-Hard** son otra clasificación presente en el diagrama que no necesariamente incluyen a problemas de decisión. Este tipo de problemas no pueden ser fácilmente resueltos, ni verificados, en un tiempo polinómico.

**Problemas NP-Completo**Un problema de decisión C es NP-completo si:

* C está contenido en el conjunto NP, y
* Todo problema de NP es reducible a C en tiempo polinómico.

Los problemas NP-Completo pertenecen tanto a la clase NP como a la NP-Hard. Si bien las soluciones para un problema NP-Completo pueden chequearse rápidamente, no se conocen formas de encontrar esas soluciones en la misma velocidad. Además, todos estos problemas comparten un punto importante en común: la dificultad de los mismos escala a medida que el problema se hace más y más grande.

A medida que fueron pasando los años, la cantidad de problemas NP-Completo fue aumentando en conjunto con la relevancia de los mismos.

Si bien los problemas NP-Completo pueden parecer abstractos e intangibles a simple vista, varios de ellos tienen aplicaciones en la vida real, las cuales, si fuesen resueltos los mismos, harían muchísimo más eficaces nuestras vidas.

Problemas como las rutas que deben tomar los vehículos (adaptación del problema Travelling Salesman), problemas relacionados a las bases de datos (feedback vertex set), diseño de circuitos (SAT), incluso el plegamiento de proteínas, son todos problemas que entran bajo la clasificación de NP-Completo, lo que significa que las computadoras no pueden resolverlos de forma fácil en un tiempo adecuado.  
  
También, muchos juegos que conocemos y jugamos día a día son problemas NP-Completos. Se ha demostrado que juegos como Sodoku, Tetris, Super Mario Bros y Metroid son juegos que pueden convertirse en problemas NP-Completo.

Lograr resolver estos problemas en un tiempo polinómico y la gran importancia que eso conllevaría es lo que lo convierte en uno de los principales problemas de la computación en los últimos años.  
  
Los siguientes conceptos erróneos son frecuentes:

* **Los problemas NP-Completo son los problemas conocidos más difíciles:** Ya que los problemas NP-Completo se encuentran dentro de la clase NP, su tiempo de ejecución es como mucho exponencial. Sin embargo, algunos problemas pueden requerir probablemente más tiempo.
* **Los problemas NP-Completo son difíciles porque hay muchas soluciones diferentes posibles:** Por una parte, hay muchos problemas que tienen un conjunto de solución grande, pero que pueden ser solucionados en tiempo polinómico (por ejemplo, árbol recubridor mínimo). Por otro lado, hay problemas NP con a lo sumo una solución que son NP-Hard bajo una reducción polinómica de tiempo aleatorizada (teorema de Valiant-Vazirani).
* **Resolver problemas NP-Completo requiere tiempo exponencial:** Esto implicaría que P ≠ NP, lo cual todavía es una pregunta sin responder. Además, algunos problemas NP-Completo tienen algoritmos que se ejecutan en un tiempo super-polinómico pero sub-exponencial, como O(2√nn). Por ejemplo, el conjunto independiente y los problemas de conjunto dominantes para los gráficos planares son NP-Completo, pero se pueden resolver en tiempo sub-exponencial utilizando el teorema del separador planar.
* **Si P = NP, todos los cifrados criptográficos podrían ser quebrados:** Un problema de tiempo polinómico puede ser muy difícil de resolver en la práctica si el grado del polinomio o las constantes son lo suficientemente grandes. Por ejemplo, cifrados con una clave de longitud fija, como AES, pueden ser todos quebrados en un tiempo constante (y son por lo tanto pertenecientes a la clase P), sin embargo, con la tecnología actual, esa constante puede superar la edad del universo. Además, existen métodos criptográficos que no pueden ser quebrados incluso con poder de computación ilimitado.

**¿P = NP?**

Esta pregunta sigue sin resolverse, y aparentemente, no estamos más cerca de saber la respuesta.   
  
En el año 2000, el Instituto Matemático Clay presentó 7 problemas, y una recompensa de 1 millón de dólares para cada uno. Hasta el día de hoy, solo uno de ellos pudo ser resuelto, la hipótesis de Poincaré.  
  
Scott Aaronson, un investigador de complejidad en el MIT, dice los siguiente acerca de P = NP:

*“Si P = NP, entonces el mundo sería un lugar profundamente diferente al que estamos acostumbrados a asumir que es. No habría un valor especial para las “ráfagas de creatividad”, ninguna brecha fundamental entre solucionar un problema y reconocer la solución de un problema una vez que se encuentra. Todo aquel que pudiese apreciar una sinfonía sería Mozart, todo aquel que pudiese seguir paso a paso una serie de instrucciones sería Gauss.”*

Si pudiésemos resolver problemas NP como si fueran P (efectivamente convirtiéndolos en clase P), si conociésemos un atajo que nos permitiese hacerlo, varias cosas de nuestras vidas se verían simplificadas. Encontraríamos soluciones para problemas que nos afectan a diario, y podríamos aumentar aún más la eficacia con la que realizamos nuestras tareas. Pero también, muchos problemas se volverían simples, se perdería el sentido de complejidad. La comunidad científica tiende a aceptar que P ≠ NP, pero realmente no estamos seguros como para poder afirmarlo. Quizás nos hace falta descubrir un atajo esencial que nos ayude a ver lo que no estamos viendo. Quizás necesitamos esperar un Einstein que haga por la computación lo que él hizo por la física.

**Bibliografía y enlaces útiles**

* [Turing Machines Explained – Computerphile](https://www.youtube.com/watch?v=dNRDvLACg5Q)
* [P vs. NP and the Computational Zoo – hackerdashery](https://www.youtube.com/watch?v=YX40hbAHx3s)
* [David Maier, Profesor de Ciencias de la Computación](https://www.quora.com/How-can-a-non-deterministic-algorithm-exist-how-can-one-state-lead-to-possible-two-different-states), respuesta dada en el sitio Quora.com
* Computers and Intractability: A Guide to the Theory of NP-Completeness, Garey, M.R.; Johnson, D.S. (1979)