Aplicación de la teoría de la complejidad en optimización combinatoria

◆ **Marco Antonio Cruz Chávez**
Pedro Moreno Bernal
Jesús del Carmen Peralta Abarca

Hoy en día, a pesar de que las computadoras han evolucionado rápidamente y de que son capaces de procesar una gran cantidad de operaciones por segundo, existen problemas cuya solución puede tardar años en obtenerse. Un ejemplo de este tipo de problema es el del "agente viajero" —conocido como TSP (*traveling sales problem*), por sus siglas en inglés. Trata de un vendedor que debe visitar cierto número de ciudades. Partiendo de su ciudad de origen, debe visitar una vez cada ciudad y regresar al punto de partida. El objetivo es encontrar una ruta que reduzca la distancia recorrida. La figura 1 muestra el caso en que el agente tiene que pasar por cuatro ciudades, cada una representada por un nodo, así como la distancia respectiva entre ellas. Este problema tiene aplicación real en empresas que necesitan reducir costos de transporte y presenta las tres posibles rutas que puede tomar el agente viajero. La mejor es la del orden C1, C2, C4, C3, dado que la distancia de 27 es la más corta.

Una forma de resolver este problema sería evaluar todas las posibles combinaciones de recorridos y escoger la de menor costo; pero tan solo para doce ciudades hay 19 958 400 rutas posibles, así que sería casi imposible evaluar manualmente cada recorrido para escoger el mejor. Si el número de ciudades a visitar fuera de cincuenta, el número de recorridos sería tan grande que no alcanzaría a resolverse ni siquiera en varios meses.

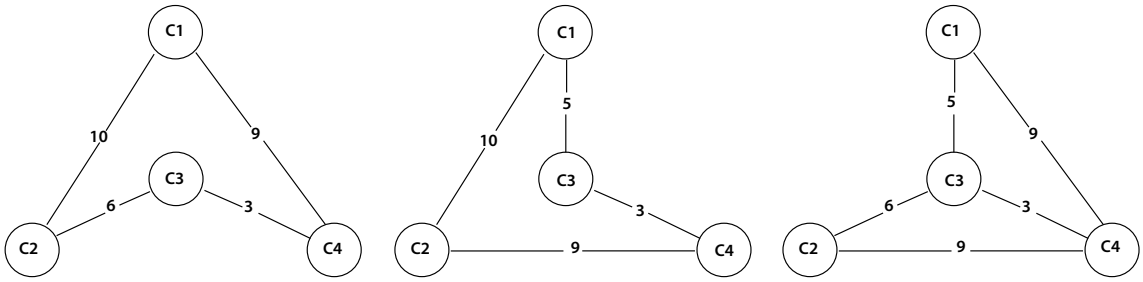
Otro problema de optimización es el de la mochila. En este se plantea llenar una mochila cuya restricción principal es no soportar más que un peso determinado al cargar una parte de un conjunto de objetos, cada uno de ellos con un peso y valor determinados. Debe maximizarse la carga de objetos en la mochila sin exceder el peso permitido. Su formulación es sencilla, pero la forma de resolverlo es muy compleja. Este problema se presenta como aplicación real en las empresas que requieren almacenar una gran cantidad de productos.

Se dice que un problema es fácil de resolver cuando es posible encontrar un algoritmo (método de solución) cuyo tiempo de ejecución en una computadora crezca de forma "razonable" o moderada de acuerdo con el tamaño del problema. Por el contrario, es difícil cuando el algoritmo que lo resuelve tiene un tiempo de ejecución que crece exponencialmente con el tamaño del problema.

Bajo algunas condiciones, el uso de un algoritmo computacional para resolver un problema que sea adecuado a sus características es suficiente, pero para otras circunstancias se tienen que diseñar algoritmos computacionales muy específicos para las condiciones dadas, que resuelvan la problemática presentada de manera eficaz y eficiente.

◆ Profesor e investigador, Centro de Investigaciones en Ingeniería y Ciencias Aplicadas (Ciicap), UAEM
Doctorado en Ingeniería y Ciencias Aplicadas, Centro de Investigaciones en Ingeniería y Ciencias Aplicadas (Ciicap), UAEM
Profesora e investigadora, Facultad de Ciencias Químicas e Ingeniería (FCQeI), UAEM

Figura 1. Tres posibles rutas para el agente viajero



Fuente: autores

Se puede afirmar que para todos los problemas, existe al menos un algoritmo de solución; sin embargo, generalmente se tienen o proponen varios algoritmos para tratar de resolver un problema. Una forma de escoger el mejor es mediante la comparación de la eficiencia, que permite evaluar de alguna forma el costo en tiempo o en recursos computacionales que consume un algoritmo para encontrar la solución, lo que significa que se elegirá aquel que utilice de forma más eficiente dichos recursos (computadoras personales, servidores, *clusters*, supercomputadoras y otros) para su ejecución. Otra medida relevante para la elección de un algoritmo es la eficacia, la cual es la medida de la capacidad que tiene el algoritmo para acercarse a la solución de un problema.

Complejidad algorítmica

Es difícil realizar un análisis simple de un algoritmo que determine la cantidad exacta de tiempo que este requiere para ser ejecutado, porque depende en gran parte del algoritmo y de la computadora en que se ejecute. Además, conocer el tiempo exacto que tardará un programa de cómputo en dar resultados es una tarea difícil de determinar. En su lugar, es mejor calcular la cantidad de operaciones que se realizan de acuerdo con los datos de entrada del problema a tratar, lo que se conoce como cálculo de la función temporal.

Así, una vez que se cuenta con un algoritmo que funciona de manera idónea, es necesario definir los criterios que permitan medir su rendimiento o comportamiento. Estos deben considerar el uso eficiente de los recursos y la simplicidad del algoritmo. El que sea sencillo no le demerita calidad, ya que su simplicidad facilita su mantenimiento, su verificación y su eficiencia.

Al hablar del uso eficiente de los recursos, este puede medirse en función de dos indicadores: espacio (cantidad de memoria que utiliza) y tiempo (lo que tarda en ejecutarse). Si para resolver un problema P un algoritmo A requiere de poca memoria del equipo de cómputo o ejecuta un pequeño número de instrucciones comparado con el

resto de los algoritmos conocidos que resuelven P, entonces se puede afirmar que A es más eficiente que los restantes cuando se resuelve P.

Un programa es eficiente si su costo es mínimo en cuanto a:

El costo espacial (espacio), que es la medida de la cantidad de memoria necesaria para ejecutarlo hasta su término.

El costo temporal (tiempo), que es una medida del tiempo empleado por el programa para ejecutarse y dar resultado a partir de los datos de entrada, y que considera una aproximación al número de pasos de ejecución que el algoritmo emplea para resolver un problema.[1]

Así, el rendimiento de un programa se mide con dos variables: la memoria ocupada y el tiempo de ejecución. Cuando un cálculo necesita más tiempo que otro se dice que es más complejo, y se le llama a esto complejidad temporal; si requiere más espacio que otro es complejidad espacial.[2]

Memoria ocupada

La eficiencia de la memoria o complejidad espacial de un algoritmo muestra la cantidad de memoria que ocupan todas las variables utilizadas por este, es decir, la suma del almacenamiento necesario para ejecutarlo, que está constituida por la memoria estática y la memoria dinámica.

Para el cálculo de la memoria estática solo hay que sumar la memoria ocupada por cada una de las variables declaradas en el algoritmo. El cálculo de la memoria dinámica depende de la ejecución del algoritmo y puede ser liberada, se modifica de forma permanente; concretamente, es un espacio de almacenamiento que se solicita en el tiempo de ejecución.

La complejidad en función de la memoria utilizada radica en el consumo del espacio de esta en la computadora. Un problema con costo espacial elevado gastará una cantidad de memoria con incremento exponencial conforme el problema aumente en tamaño, pues el número de variables a utilizar también aumentará. Esto en algún momento causaría que el algoritmo no se realice correctamente por falta de memoria en la computadora.

Tiempo de ejecución

La complejidad temporal de un algoritmo computacional se evalúa generando una función, llamada función temporal, la cual define el número de instrucciones ejecutadas por el programa cuando se resuelve un problema. De este número de instrucciones se puede obtener el tiempo aproximado si se conoce el tiempo que el equipo de cómputo en uso tarda en ejecutar una instrucción. De esa manera se puede representar el número de unidades de tiempo requeridas para que un programa o algoritmo de cualquier entrada de datos de tamaño *n* produzca un resultado. El tiempo de ejecución de un algoritmo depende del número de datos de entrada *n* del problema y de la velocidad que el equipo de cómputo posea.

---

[1] Laureano Santamaría Arana y Alejandro Rabasa Dolado, *Metodología de programación. Principios y aplicaciones*, ECU, San Vicente, 2004, p. 8.

[2] Luis Joyanes Aguilar e Ignacio Zahonero Martínez, *Algoritmos y estructuras de datos. Una perspectiva en C*, McGraw-Hill Interamericana, Madrid, 2004, p. 33.

La complejidad temporal se expresa normalmente utilizando la "notación O", o de la "O grande", con la cual se dispone de un medio para expresar la cota asintótica de una función en el peor de los casos. El término asintótico se refiere al tamaño de la entrada del problema, en este caso, para un valor de *n* grande. El peor de los casos indica cuando el algoritmo lleva a cabo el total de las instrucciones que este puede ejecutar. Es una expresión aproximada de la relación entre el tamaño de un problema y la cantidad de instrucciones necesarias en el algoritmo para obtener un resultado.

El ejemplo en la tabla 1 es una muestra de un análisis detallado que debe realizarse en una serie de instrucciones que se aplican en un algoritmo computacional para resolver un problema.

Para evaluar la eficiencia de los algoritmos se requiere evaluar su complejidad, y el beneficio de hacerlo es que se puede trabajar en el análisis para el diseño de un algoritmo que pueda tener mejor calidad y eficiencia.

Tabla 1

Función temporal F(N)	Complejidad asintótica
$3.5\ n^2$	$O(n^2)$
$2\ n^3$	$O(n^3)$
n^4	$O(n^4)$
$2\ n^2+3\ n-1c$	$O(n^2)$

Joyanes y Zahonero[3] presentan los siete tipos de complejidades más comunes, las cuales se muestran en la tabla 2.

Complejidad de problemas

La teoría de la complejidad estudia la manera de clasificar problemas de acuerdo con la dificultad propia para resolverlos, basándose en los recursos necesarios y requeridos para establecer su grado de complejidad. Un cálculo resulta complejo si es difícil de realizar.[4]

Se puede definir "complejidad" como la cantidad de recursos necesarios para efectuar un cálculo. Si este es complicado, requerirá de más recursos que uno de menor dificultad. Un algoritmo que puede resolver un problema pero que se tarda mucho tiempo en hacerlo, no es muy útil; la misma consideración si ocupa un *gigabyte* o memoria en exceso.

Pero la tarea de encontrar un algoritmo que resuelva con pocos recursos un problema, no es lo que realmente importa; el inconveniente es saber si existe una solución o no a esta problemática presentada y qué tan compleja resultará su solución.

Para saber el grado de complejidad que puede tener un problema, nos apoyamos en el modelo computacional de la máquina de Turing,[5] con el cual se obtiene una clasificación de los problemas con base en el grado de complejidad inherente para resolverlos.

---

[3] *Ibid*., p. 44.

[4] Augusto Cortez, "Teoría de la complejidad computacional y teoría de la computabilidad", *RISI. Revista de Investigación de Sistemas e Informática*, vol. 1, núm. 1, 2004, pp. 102-105, http://bit.ly/1ol5a2N, consultado en marzo de 2014.

[5] Sanjeev Arora y Boaz Barak, *Complexity theory. A modern approach*, Cambridge University Press, Nueva York, 2009, pp. 40-41; Michael R. Garey y David S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*, Macmillan Higher Education, Nueva York, 1979, p. 10.

Tabla 2

Tipo de complejidad	Aplicación
Constante O(1)	Para algoritmos sin bucles
Logarítmica O(log n), O(n logn)	Algoritmos de búsqueda binaria, de ordenación
Lineal O(n)	Bucles simples
Cuadrática O(n^2)	Dos bucles anidados, matrices
Cúbica O(n^3)	Para número de datos (n) muy grande, complejidad notable
Exponencial O(2^n)	El crecimiento es muy rápido, aplicaciones muy complejas
O(k^n)	Algoritmos como los de las ocho reinas, salto de caballo y complejidad elevada

La máquina de Turing, inventada por el matemático inglés Alan M. Turing en 1937, es una máquina autómata artesanal que se utiliza para clasificar los problemas de acuerdo con el tipo de máquina de Turing que puede existir para resolverlos, y es el principal modelo computacional que soporta la "teoría de la complejidad de los problemas".

El modelo computacional de Turing clasifica los problemas por el grado de complejidad para resolverlos. A través de este modelo se han detectado problemas intratables, clasificados como NP (*nondeterministic polynomial time*), que se piensa son imposibles de resolver en un tiempo razonable cuando el número de variables que los componen es una cantidad extremadamente grande. Este grupo incluye problemas como el del agente viajero, el de la mochila, el transporte, la asignación de horarios para cursos universitarios o la asignación de maquinaria en talleres de manufactura.

Dentro de este grupo de problemas NP se encuentran dos subconjuntos de problemas:

Problemas P,[6] para los cuales existe una máquina de Turing determinista que los puede resolver en tiempo polinómico.[7] Esto indica que existe un algoritmo determinista[8] con complejidad polinomial que los puede resolver. Se consideran como la clase de problemas de reconocimiento relativamente sencillos, aquellos para los que existen algoritmos eficientes o exactos.[9]

Los NP-completos. No existe una máquina de Turing determinista que pueda resolverlos en tiempo polinómico. En su lugar, se puede encon-

[6] R. C. Y. Castañeda, *Estudio comparativo de diversos métodos de solución al problema del agente viajero (PAV)*, tesis de maestría, Universidad de las Américas Puebla (UDLAP), Cholula, pp. 1-3, http://bit.ly/1fWgz8A, consultado en marzo de 2014.

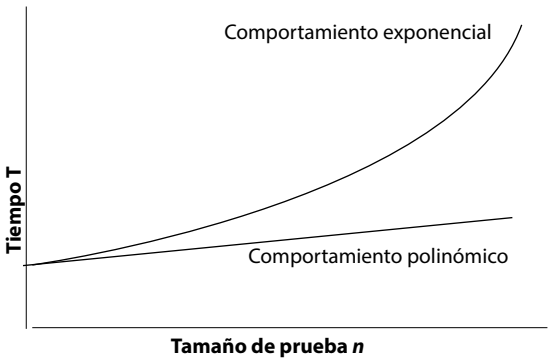
[7] En computación, cuando el tiempo de ejecución de un algoritmo (mediante el cual se obtiene la solución de un problema) es menor que cierto valor calculado a partir del número de variables implicadas (variables de entrada) usando una fórmula polinómica, se dice que dicho problema se puede resolver en un tiempo polinómico.

[8] En ciencias de la computación, un algoritmo determinista es un algoritmo del cual se conocen las variables de entrada y, por consiguiente, siempre producirá el mismo resultado, por ejemplo, el algoritmo simplex.

[9] Christos H. Papadimitriou y Kenneth Steiglitz, *Combinatorial optimization. Algorithms and complexity*, Dover Publications, Nueva York, 1998, p. 351.

trar un valor próximo a la solución del problema mediante una máquina de Turing no determinista[10] acotando polinomialmente el tiempo. Estos problemas NP son aquellos cuya solución, hasta la fecha, no se ha resuelto de manera exacta por medio de algoritmos deterministas en tiempo polinomial. En su lugar, se tratan de resolver por algoritmos no deterministas[11] acotados en tiempo polinomial, cuya solución deseable sea de complejidad polinomial. Esta clase de algoritmos se conoce como "heurísticas computacionales (en la figura 2 se muestra la diferencia entre una solución obtenida en tiempo polinomial y una en tiempo exponencial)."[12]

Figura 2. Comportamiento de soluciones en tiempo polinomial y exponencial



Fuente: autores

Acerca de por qué los problemas P están incluidos dentro de los considerados difíciles de resolver NP, esto es porque los algoritmos no determinísticos usados para los problemas NP y NP-completo, también pueden usarse en los problemas P; pero no es posible, en caso contrario, que un algoritmo determinístico que resuelva un problema P en tiempo polinomial, también pueda resolver un NP.[13] De ser esto cierto, entonces tendríamos que P=NP.[14] La relación entre la clase P y la clase NP es estrecha: P $\subseteq$ NP. Cualquier problema de decisión P resuelto por un algoritmo determinístico en tiempo polinomial, también puede ser resuelto por un algoritmo no determinístico en el mismo tiempo.

La figura 3 muestra la clasificación de los problemas con base en la complejidad para resolverlos, a partir del modelo de Turing.

El esfuerzo necesario para resolver un problema de forma eficiente puede variar enormemente. Un problema muy complejo se denomina "NP-completo" si su solución requiere de una cantidad significativa de recursos computacionales sin importar el algoritmo utilizado, lo cual significa que es imposible encontrar un algoritmo eficiente que compruebe que se alcanzó la mejor solución.

---

[10] Manuel Alfonseca Moreno, "La máquina de Turing", *Números, Las matemáticas del siglo XX: una mirada en 101 artículos*, núms. 43-44, 2000, pp. 165-168, http://bit.ly/OX1cmP, consultado en marzo de 2014.

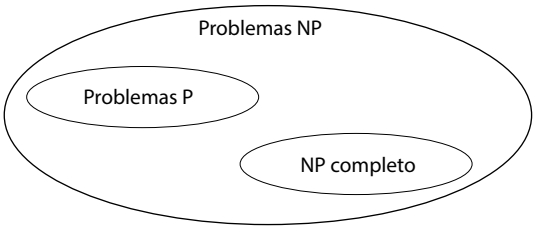
[11] Es un algoritmo que con las mismas variables de entrada ofrece muchos resultados posibles. No se puede saber de antemano cuál será el resultado de la ejecución de un algoritmo no determinista.

[12] Heurística es un método con reglas empíricas para encontrar soluciones para problemas basados en la experiencia, el cual sirve para encontrar soluciones aproximadas a la mejor solución en diversos problemas complejos, si bien no ofrece una manera de comprobar que se alcanzó la mejor solución.

[13] Christos H. Papadimitriou y Kenneth Steiglitz, *Combinatorial optimization*…. *op. cit*.

[14] Michael R. Garey y David S. Johnson, *Computers and intractability*…, *op. cit*., p. 13.

Figura 3. Clasificación de los problemas NP



Fuente: Papadimitriou y Steiglitz[15]

De ahí el uso de algoritmos heurísticos no determinísticos acotados en tiempo polinomial para este tipo de problemas.

El impacto de esta teoría en la investigación computacional es significativo porque permite determinar el grado de complejidad de un problema para ser resuelto, así como saber si se puede encontrar un algoritmo eficiente para el problema, o bien, tratarlo por el camino de las heurísticas.

De acuerdo con la teoría de la complejidad y apoyándose en la figura 2, si el problema se puede clasificar como P, entonces se podría hacer uso, encontrar o desarrollar algoritmos eficientes que comprueben si obtienen la mejor solución para dicho problema.

En caso de que el problema estuviera clasificado como NP o NP-completo, entonces se perdería el tiempo tratando de encontrar algoritmos eficientes. En lugar de eso, el camino más adecuado es trabajar con heurísticas computacionales, por lo que solo quedaría tratar de proponer nuevas heurísticas de baja complejidad.

En el ámbito internacional, se han enfocado al estudio de la complejidad computacional investigadores como Gilbert Laporte, en Francia; Christos H. Papadimitriou, en Grecia; Kenneth Steiglitz, en Estados Unidos, entre otros. En la UAEM, el Cuerpo Académico de Optimización y Software[16] ha diseñado y aplicado heurísticas computacionales en las cuales la teoría de la complejidad ha permitido que los algoritmos propuestos sean eficientes en cuanto al uso de recursos. Con ello se han obtenido soluciones para problemas planteados en tiempos computacionales rápidos y aceptables para el uso que se les da.

Tal es el caso del problema de ruteo vehicular con ventanas de tiempo, asignación y calendarización de tareas en máquinas en un taller de manufactura, asignación de tareas en máquinas paralelas, entre otros, para los cuales ya se han diseñado o mejorado heurísticas computacionales que permiten obtener buenas soluciones. Otro campo de trabajo es el del diseño de algoritmos cuya complejidad permita la solución eficiente y eficaz de problemas, en el cual se han retomado otras investigaciones novedosas.[17]

---

[15] Christos H. Papadimitriou y Kenneth Steiglitz, *Combinatorial optimization…. op. cit*.

[16] Grupo de investigación interdés, en el cual participan investigadores de las facultades de Ciencias Químicas e Ingeniería (FCQel); Ciencias (FC), y Contaduría, Administración e Informática (FCAel), así como del Centro de Investigaciones en Ingeniería y Ciencias Aplicadas (Ciicap). El grupo se apoya en este último centro para el desarrollo de sus proyectos.

[17] Enrique Alba, "Experiencias paralelas (en la solución de problemas complejos)", 9º Congreso Internacional de Cómputo en Optimización y Software 2012, 27 al 30 de noviembre de 2012, Cuernavaca, http://bit.ly/1kUScsx, consultado en marzo de 2014.