En Chile también hay ciencia

Problemas contemporáneos en Ciencias Básicas vistos por científicos chilenos

Leopoldo Bertossi D.

Editores

JAIME IHODA
JORGE MELNICK
SERGIO MELNICK

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS ECONOMICAS Y ADMINISTRATIVAS

12

PROBLEMAS EN LA TEORIA DE COMPLEJIDAD COMPUTACIONAL

Leopoldo Bertossi

*Profesor del Departamento de Matemáticas de la Facultad de Matemáticas de la Pontificia Universidad Católica de Chile. Magister en Ciencias Exactas, 1982, en la misma Universidad; tiene, además, estudios de postgrado en la Universidad de Freigburg, Alemania Federal. Candidato a Doctor, Universidad Católica de Chile, 1985.

PROBLEMAS EN LA TEORIA DE COMPLEJIDAD COMPUTACIONAL

Entre los muchos temas de investigación que aparecen en la informática teórica y en la lógica matemática, está el de la **Complejidad Computacional**, que, tal como su nombre lo indica, investiga el grado de dificultad que ofrece un problema al ser tratado con la ayuda de un computador. Este grado de dificultad podría ser expresado, por ejemplo, a través del tiempo que requiere una computación, de la cantidad de memoria, de la complejidad estructural de la máquina, etc. Como hay una enormidad de problemas que pueden ser resueltos con la ayuda de un computador, debemos restringirnos a un cierto tipo de ellos. Aquí nos preocuparemos especialmente de un tipo que es de interés más bien teórico: el de complejidad de problemas computacionales asociados a toda una teoría lógico–matemática, y no, por ejemplo, de la complejidad de algoritmos aritméticos particulares.

Las teorías lógico–matemáticas son expresadas mediante los lenguajes formales usuales de la lógica simbólica. Si $\mathfrak{A}$ es una estructura

matemática, y ya hemos fijado un lenguaje formal apropiado para hablar sobre $\mathfrak{A}$ o sobre otras estructuras matemáticas similares, entonces **la teoría de** $\mathfrak{A}$ es la familia de todas las oraciones escritas en el lenguaje formal que, al ser interpretadas en $\mathfrak{A}$, se convierten en proposiciones verdaderas. Si la interpretación de $\mathfrak{A}$ de una oración formal ϕ es una proposición verdadera, escribimos $\mathfrak{A} \models \phi$. Entonces, la teoría de $\mathfrak{A}$ es $\mathrm{Th}(\mathfrak{A}) = \{\phi : \mathfrak{A} \models \phi\}$.

1. Complejidad en teorías matemáticas

En 1960-1961, Büchi y Elgot demostraron que la teoría de segundo orden, monádica, débil de los naturales con la función sucesor es decidible. Aquí, **monádica débil de segundo orden** significa que las cuantificaciones de segundo orden son sólo sobre subconjuntos finitos del universo de la estructura.

Denotemos con $T = Th_W(\mathbb{N},S)$, esta teoría. Un ejemplo de una oración que está en **T** es $\forall X(\exists x\, Xx \rightarrow \exists y(Xy \wedge \neg XSy))$, ya que interpretada en la escritura $(\mathbb{N},S)$ dice que todo subconjunto no vacío finito de $\mathbb{N}$ contiene un elemento cuyo sucesor no está en el conjunto, lo que es cierto. Con esta notación, podemos formular el resultado inicial en la siguiente forma: "existe un algoritmo Al, tal que, para cada oración ϕ escrita en el lenguaje correspondiente a **T**, Al permite decidir si ϕ está en **T** o no está en **T**, es decir, si $(\mathbb{N},S) \models \phi$ ó $(\mathbb{N},S) \not\models \phi$". Diremos que Al decide ϕ.

El ejemplo de la teoría **T** es un caso en una lista de teorías cuya decidibilidad ha sido demostrada

Después del esfuerzo desplegado para demostrar que una teoría particular es decidible, parece natural, tal vez, pensar que el **problema** está resuelto, es decir, ya se dispone de un algoritmo general, que, aplicado a cualquier oración escrita en el lenguaje formal de la teoría en cuestión, decide después de un número finito de pasos si ella, está en la teoría o no. Sin embargo, nada se ha dicho sobre la posibilidad real de aplicar el algoritmo, es decir, sobre la posibilidad de implementarlo en un computador digital común, obteniendo la respuesta deseada después de un tiempo razonable. Al preguntarnos esto con respec-

to a una teoría
negativas posibles
posible que haya
la teoría es decidi
ble en la práctica"

La primera re
da, precisamente p
una constante c>0
teoría **T**, existen i
de modo que el nú
que $2^{2^{\cdot^{\cdot^{\cdot^{2|\phi|}}}}}$. do
de veces que apar

De acuerdo
quier algoritmo d
que podrá decidir
lo que, como algo
mos que esta teor
per exponencial.

Otro caso fa
llamada aritmética
dad. El algoritmo
ca de eliminación
complejidad expo
de hecho, esta teo
nencial (en el tie
$c > 0$, tal que par
ten infinitas oraci
por lo menos $2^{2^{c}}$

2. Problemas inma

Por su inher
cio, o memoria,
decimos que los
Th($\mathbb{N}$, +) son **in**

to a una teoría decidible, tenemos fundamentalmente dos respuestas negativas posibles: "el algoritmo exhibido no es útil en la práctica y es posible que haya otro más eficiente que lo sustituya" o bien, "aunque la teoría es decidible, ningún algoritmo de decidibilidad es implementable en la práctica".

La primera respuesta negativa del segundo tipo para una teoría fue dada, precisamente para la teoría $Th_W(N,S)$, por A. Mayer en 1972: "existe una constante $c>0$, tal que, para todo algoritmo de decisión Al para la teoría **T**, existen infinitas oraciones ϕ, escritas en el lenguaje de la teoría, de modo que el número de pasos que requiere Al para decidir ϕ es mayor que $2^{2^{\cdot^{\cdot^{\cdot^{2^{|\phi|}}}}}}$, donde $|\phi|$ denota la longitud de la oración ϕ y el número de veces que aparece el 2 es el mayor entero que no excede a $c|\phi|$."

De acuerdo con este resultado, vemos que necesariamente cualquier algoritmo de decisión para la teoría **T** se encontrará con oraciones que podrá decidir sólo después de un tiempo exponencialmente largo, lo que, como algoritmo general, lo hace ineficiente en la práctica. Diremos que esta teoría tiene una complejidad computacional inherente super exponencial.

Otro caso famoso es el de la teoría de primer orden de $(\mathbb{N}, +)$, la llamada aritmética de Presburger, el cual probó en 1929 su decidibilidad. El algoritmo exhibido por Presburger, correspondiente a la técnica de eliminación de cuantificadores, es ineficiente en la práctica por su complejidad exponencial. En 1974, Fischer y Rabin demostraron que, de hecho, esta teoría tiene complejidad computacional inherente exponencial (en el tiempo o número de pasos): "existe una constante $c > 0$, tal que para todo algoritmo Al de decisión para $Th(\mathbb{N}, +)$, existen infinitas oraciones ϕ en el lenguaje de la teoría, tales que Al necesita por lo menos $2^{2^{c|\phi|}}$ pasos computacionales para decidir ϕ".

2. Problemas inmanejables

Por su inherente alta complejidad en tiempo (también el espacio, o memoria, podría ser, entre otras, una medida de complejidad), decimos que los problemas de decisión para las teorías $Th_W(\mathbb{N},S)$ y $Th(\mathbb{N}, +)$ son **inmanejables**. Por otra parte, hay unanimidad en consi-

derar a un problema manejable (con respecto al tiempo) si hay un algoritmo Al que lo resuelve en tiempo polinomial, es decir, si existe un polinomio **q**, tal que el número de pasos de Al con una entrada **w** de largo |w| no es mayor que q(|w|).

Denotemos con **P** la clase de los problemas solubles mediante algoritmos en tiempo polinomial. Aquí entendemos por algoritmo un procedimiento que puede ser implementado en una computadora. Sin embargo, una definición precisa de algoritmo puede ser dada dentro del formalismo de las funciones recursivas o de las máquinas de Turing determinísticas. Una máquina de Turing no es otra cosa que un modelo matemático de lo que es un computador y, por consiguiente, pretende capturar los elementos esenciales, digamos, de los computadores digitales actuales, sin limitarse a un cierto tipo particular.

El concepto de algoritmo no determinístico puede, a su vez, ser definido dentro del formalismo de las máquinas de Turing no determinísticas. Para intuir mejor estos conceptos, digamos que, en la aplicación de un algoritmo determinístico, cada paso de la computación da lugar a lo sumo a un único paso siguiente. Sin embargo, cada paso en un algoritmo no determinístico da lugar a un número finito de posibles pasos siguientes. Vemos entonces que el formalismo es tal, que toda máquina determinística es también una máquina no determinística, pero no recíprocamente. Dicho de otra forma, todo algoritmo determinístico es también no determinístico. Tenemos entonces, en particular, que todo problema soluble (o decidible) en tiempo polinomial mediante un algoritmo determinístico es también soluble en tiempo polinomial en forma no determinística.

Si **NP** denota la clase de los problemas solubles en tiempo polinomial con un algoritmo no determinístico, tenemos entonces que $P \subseteq NP$. Por otra parte, veremos ejemplos de problemas que están en **NP**, pero cuya pertenencia a **P** no está ni establecida ni refutada. Lo más que se tiene, son problemas (importantes en un sentido que precisaremos) que están en **NP** y para los cuales hay **hasta ahora** sólo algoritmos determinísticos exponenciales en tiempo. Esto conduce a la conjetura $P \neq NP$, la llamada **conjetura de Cook**, que es, tal vez, el problema no resuelto más famoso de la teoría de complejidad computacional.

3. Complejidad de algoritmos

Un problema importante es el llamado de satisfacibilidad, que denotaremos con **s**. Este consiste en determinar en forma efectiva, es decir, en forma algorítmica, si dada una fórmula cualquiera del cálculo de proposiciones, por ejemplo $((p \wedge \neg q) \vee (r \rightarrow p)) \vee (\neg r \wedge t)$, existe una asignación de valores de verdad **V** o **F** a las proposiciones elementales **p, q, r, t** de modo que la fórmula completa tome el valor **V**.

Obviamente existe un algoritmo general que resuelve este problema: **"examine la tabla de verdad en las proposiciones elementales que componen la fórmula"** (la tabla de verdad será siempre finita).

Debería ser claro, por lo menos intuitivamente, que este algoritmo es muy complejo, de hecho, exponencial en tiempo con respecto al largo de la fórmula de entrada. Sin embargo, no se sabe si $s \in P$, es decir, si hay un algoritmo determinístico, polinomial en tiempo, que lo resuelve. Por otra parte, se puede exhibir algoritmos no determinísticos que resuelven el problema **s** en tiempo polinomial. Tenemos así la siguiente situación: $P \subseteq NP$, $s \in NP$, ¿$s \in P$? Luego, si tenemos una demostración de $s \notin P$, tendremos una demostración de la conjetura de Cook. Obviamente, cualquier problema que esté en **NP** y que, a la vez, no esté en **P** nos confirmará la conjetura. Pero, como ya dijimos, el problema **s** de satisfacibilidad es importante, por lo menos dentro del contexto de la complejidad computacional, ya que Cook demostró que si $s \in P$, entonces todos los otros problemas que están en **NP** también están en **P** (por esto se dice que el problema **s** es NP-Completo). O sea, si llegamos a tener una demostración de $s \in P$, tendremos una demostración de $P = NP$. Sin embargo, el que hasta ahora no se haya encontrado un algoritmo determinístico polinomial en tiempo para **s** da un respaldo a la conjetura de Cook.

Digamos finalmente que **s** no es el único problema (natural) que es NP-completo, es decir, que está en **NP** y cuya pertenencia a **P** implicaría $P = NP$. De hecho, se ha seguido encontrando problemas NP-completos, por ejemplo, en teoría de grafos, en programación entera, en teoría de juegos, etc. Con seguridad, esta búsqueda de nuevos problemas NP-completos está inspirada, aparte de la simple curiosidad mate-

mática, por la esperanza de encontrar un problema más manejable que el de satisfacibilidad y que también conduzca a una solución a la conjetura de Cook.

4. Una jerarquía de complejidad para algoritmos

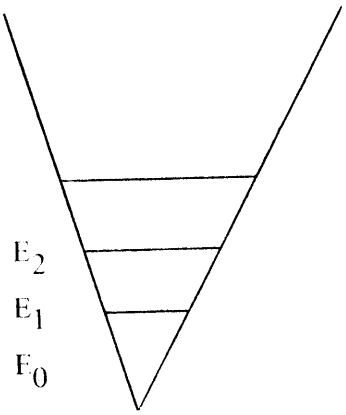
En forma casi unánime se acepta la llamada **Tesis de Church** que, en esencia, expresa que las funciones computables, y ésta es una noción intuitiva, son exactamente las calculables mediante máquinas de Turing. Como todo algoritmo puede ser concebido también como una función computable, tenemos, aceptando la tesis de Church, que los algoritmos son exactamente los procedimientos que pueden ser ejecutados mediante el uso de una máquina de Turing.

Otra formalización alternativa del concepto intuitivo de función computable es la de las funciones recursivas. Sin embargo, se demostró que las formalizaciones del concepto de algoritmo a través de las máquinas de Turing y a través de las funciones recursivas son equivalentes, es decir, una función es recursiva si y sólo si es calculable mediante una máquina de Turing.

Hay subclases de las funciones recursivas que son de interés especial. Una de éstas es la de las funciones recursivas primitivas, las cuales aparecen en la mayoría de las aplicaciones. Por esto último, es posible restringirse con frecuencia a esta subclase.

Las funciones recursivas primitivas pueden ser ordenadas en una jerarquía de complejidad; refiriéndonos aquí a la complejidad de la formación de una función, partiendo de funciones más simples. Hay varias jerarquías posibles. Una que se ha logrado imponer es la de Grzegorczyk: $E_0 \subseteq E_1 \subseteq E_2 \ldots$

Cada nivel está constituido por funciones recursivas primitivas, y un nivel superior significa mayor **complejidad** de sus elementos en el sentido recién descrito. También es posible caracterizar niveles en términos de complejidad en tiempo y/o espacio.



Teniendo así una jerarquía de complejidad para las funciones recursivas primitivas, es posible asociar una complejidad a cada problema soluble por medio de un algoritmo (recursivo primitivo): decimos que un problema es E_n-soluble (o decidible, según corresponda) si hay un algoritmo $Al \in E_n \setminus E_{n-1}$ que lo resuelve, y ningún otro algoritmo que lo resuelve está en E_{n-1}.

El paso siguiente natural consiste en estudiar la complejidad computacional en términos de esta jerarquía, o de cualquier otra, de problemas específicos que son solubles en forma algorítmica (primitiva). Algunos pasos más adelante, podríamos formularnos el problema inverso: ¿Es posible exhibir problemas de cierto tipo que sean E_n-solubles, cuando **n** está dado previamente en forma arbitraria? Con respecto a esta pregunta, digamos solamente que ha habido respuestas positivas para problemas de decisión en teoría de grupos, y que no se sabe qué pasa con el problema de decisión de teorías lógicas finitamente axiomatizables, es decir, si es posible construir teorías lógicas, decidibles, finitamente axiomatizables, cuyo problema de decisión sea E_n-soluble, donde **n** es arbitrario.

Apéndice: Un algoritmo no determinístico para el problema s

Es posible construir una máquina **M** que, al entregarle una fórmula del cálculo proposicional cualquiera, decide en forma no determinística si es satisfacible o no lo es. Esta máquina trabaja, digamos, con el alfabeto A = {**p, 0, 1, V, F,** $\wedge$ **,** v, $\neg$,), (}. Entonces, las fórmulas del cálculo proposicional son palabras sobre este alfabeto, es decir, sucesiones finitas de símbolos de A: la fórmula $(p_1 \wedge p_2) \vee (p_1 \vee \neg p_3)$ se codifica con el alfabeto A en la forma (p1 $\wedge$ p10) v (p1 v $\neg$p11), esto es, la subfórmula elemental p_i se reemplaza por **p** seguida de la representación binaria de **i**. Notemos que, por ejemplo, p1$\neg$) también es palabra sobre A, pero no es una fórmula del cálculo proposicional. Entonces lo primero que hace la máquina, cuando se le introduce como entrada una palabra **w** sobre A es verificar si **w** es fórmula del cálculo de proposiciones. Si no lo es, la rechaza. Si lo es, hace una lista de todas las proposiciones elementales que forman **w**, sin repeticiones. En seguida, **elige** un valor de verdad **V** o **F** para cada uno de los elementos de la lista, los reemplaza en lugar de las proposiciones elementales en **w**. Finalmente comprueba si **w** toma el valor **F** o toma el valor **V** (de acuerdo a las tablas de verdad usuales), y acepta **w** en este último caso.

Dado que la máquina es no determinística, puede haber varias computaciones posibles que conducen a la aceptación de **w**. Por definición, **w** es aceptada por la máquina **M**, es decir, la decide como satisfacible, si hay una computación en **M** con entrada **w**, que la acepta. Además, definimos el tiempo que toma la aceptación de **w** como el tiempo que demora la más corta de las computaciones aceptantes. Es posible construir una máquina **M** que conduce a la aceptación o rechazo de una fórmula **w** cualquiera en tiempo polinomial.

BIBLIOGRAFIA

Sección 1

BÜCHI, J.R. *Weak second order arithmetic and finite automata*, Z. Math. Logik U. Grund. der Math., 6, 1960: 66–92.

ELGOT, C.C. *Decision problems of finite automata design and related arithmetics,* Trans. Am. Math. Soc. 98, 1961: 21–51.

FISCHER, M.J., M.O. RABIN. *Super exponencial complexity of presburger's arithmetic*, SIAM-AMS Procedings, 7, 1974: 27–41.

MEYER, A.R. *Wark SIS cannot be decided*, Not. Am. Math. Soc., 19, 1972, A-598.

PRESBURGER, M. *Ueber die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt,* Comptes rendus du 1er Congres des Mathematiciens des Pays Slaves, Warszawa, 1929.

Sección 2

COHEN, J. *Non-deterministic algorithms*, computing surveys, 11, 1979: 79-94.

COOK, S.A. *The complexity of theorem proving procedures*, Proceedings of the 3rd annual ACM symposium on theory of computing, 1971: 151-158.

Sección 3

COOK, S.A. Op. cit.

KARP, R.M. *Reducibility among combinatorial problems* en *Complexity of computer. Computations,* Miller, R.E., Thatcher, J.W. (eds.). Plenum Press, N.Y., 1972: 85–104.

Sección 4

AVENHAUS, J., K. MADLENER. *Subrekursive Komplexität bei Gruppen*, Acta Informática, 9, 1977: 87-104.

GATTERDAM, R.W. *The computability of group constructions II,* Bull. Austral. Math. Soc., 8, 1973: 27–60.

GRZEGORCZYK, A. *Some classes of recursive functions*, Rozprawy Math., 4, 1953: 1-45.

WEIHRAUCH, K. *Teilklassen primitiv-rekursiver Wortfunktionen*, GMD Bonn, 91, 1974.