Algunos límites de la capacidad de cálculo de los ordenadores

Lección inaugural
del curso académico 2008-2009
pronunciada por el

Prof. Dr. José Ramón Garitagoitia Padrones
Catedrático de Lenguajes y Sistemas Informáticos
de la Universidad Pública de Navarra

Pamplona, 24 de septiembre de 2008

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Edita: Universidad Pública de Navarra : Nafarroako Unibertsitate Publikoa

Coordinación: Dirección de Comunicación

Fotocomposición: Pretexto. pretexto@pretexto.es

Imprime: Ona Industria Gráfica

Depósito Legal: NA 2.963/2008

Distribución: Dirección de Publicaciones
Universidad Pública de Navarra
Campus de Arrosadía
31006 Pamplona
Fax: 948 169 300
Correo: publicaciones@unavarra.es

Excelentísimo Sr. Presidente del Gobierno de Navarra,
Sr. Rector Magnífico,
Excelentísimas e Ilustrísimas Autoridades,
Miembros de la Comunidad Universitaria,
Señoras y Señores:

Es para mí un honor ser hoy el encargado de dictar la lección inaugural del curso 2008-2009 en nuestra Universidad. Cuando nuestro Rector me confió esta tarea, mi primer sentimiento fue de agradecimiento por la oportunidad que se me daba de dirigirme a todos ustedes y poder así hablarles de informática, disciplina a la que he dedicado toda mi vida profesional. Mi segundo sentimiento fue de preocupación, al darme cuenta de que la elección del tema de hoy no iba a ser sencilla. Muchos de ustedes trabajan con ordenadores y creo que les haría perder su tiempo si dedicase esta lección a una primera aproximación a la informática, pero por otro lado difícilmente se soportaría en este acto un tema especializado de los que se tratan en un curso avanzado. No es fácil hablar de informática a un público general sin repetir temas mil veces escuchados. Los medios de comunicación presentan, prácticamente a diario, noticias de nuevos avances y nuevos productos relacionados con la informática. Muchos de los nombres propios de la informática y muchos de sus hitos fundamentales son parte de la cultura general de nuestro tiempo.

Los ordenadores son omnipresentes en nuestras vidas y parecen ser omnipotentes. Al menos, esa es la sensación que en muchas ocasiones se transmite a través de la literatura y el cine de ciencia ficción, donde los ordenadores muestran habilidades sorprendentes. Pero esa imagen no es real, los ordenadores tienen sus limitaciones, hay cálculos que un ordenador necesitaría un tiempo extraordinario para realizar y otros que simplemente no puede hacer. Este es un tema más desconocido por el público y sobre el que existe más confusión. Así que en vez de disertar sobre lo que pueden hacer los ordenadores, decidí hablarles sobre lo que no pueden hacer.

No quiero que vean nada negativo en esto, nada más lejos de mi ánimo que hablar mal de mi profesión. Todo lo contrario, conocer los límites es fundamental para un profesional en el ejercicio de su trabajo y le puede ahorrar más de un fracaso. Para los no profesionales, conocer dónde están las fronteras de la informática les servirá para confirmar lo extenso del territorio. Un territorio que abarca la oficina, la industria, la enseñanza, la investigación, la medicina, el hogar, las comunicaciones, el transporte, el ocio..., prácticamente todos los rincones de nuestras vidas.

Antes de entrar en materia, quiero que sepan que si hoy me encuentro en este lugar es, como reconocía Newton, *porque me he aupado en hombros de gigantes*[1], es decir, de aquellos que ensancharon y engrandecieron la Informática –unos pocos serán nombrados en esta lección–, pero también de mis profesores, en especial los que me iniciaron en esta materia, de los compañeros en mis distintas etapas en la universidad, y de todas aquellas personas que alguna vez me han ayudado. Si hoy perciben alguna luz desde mi lugar, sepan que no es propia, sólo estarán viendo en mí el reflejo, espero que fiel, de todos ellos.

Comenzaremos con una definición. La palabra informática proviene de la contracción de las palabras *información* y *automática*. Según el Diccionario de la Lengua Española publicado por la Real Academia Española[2], la informática es el *conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de ordenadores.*

Un invento capital en la historia de los ordenadores y de gran importancia en el desarrollo de la sociedad actual, es el de la codificación de la información. Hollerith[3], con su Máquina Tabuladora, fue el primero en lograr el tratamiento automático de la información mediante la codificación[4]. El objetivo de los constructores de las primeras máquinas de cálculo era crear máquinas capaces de efectuar cálculos repetitivos. Hollerith amplió el objetivo. Trabajando para el Departamento Estatal del Censo de

1. Isaac Newton (Woolsthorpe, Lincolnshire, Inglaterra, 4-1-1643 – Londres, 31-3-1727): "Si he conseguido ver más lejos, es porque me he aupado en hombros de gigantes". La cita no es del todo original de Newton, éste se inspira en Bernardo de Chartres.

2. Diccionario de la Lengua Española. Vigésima segunda edición. Real Academia Española, 2001.

3. Herman Hollerith, Buffalo, Nueva York, 29-2-1860 – Washington, 17-11-1929. Ingeniero de la Universidad de Columbia. Hollerith fundó en 1896 la empresa Tabulating Machine Company, que finalmente se transformó en la International Business Machines Corporation (IBM).

4. La primera en darse cuenta de que las máquinas de calcular se podrían utilizar para otros fines fue Ada Byron (Ada Augusta Byron King, condesa de Lovelace, Londres, 10-12-1815 – Londres, 27-11-1852), que, adelantándose a su tiempo, sugirió que la Máquina Analítica podría llegar a utilizarse para componer elaboradas piezas de música [Copeland00].

los Estados Unidos en la realización del censo de 1890, codificó las respuestas de la población al cuestionario propuesto, mediante perforaciones en una tarjeta de papel, logrando obtener en sólo dos años los resultados del censo, que hasta entonces tardaban en conocerse casi una década.

Toda información que pueda ser codificada, puede ser almacenada y procesada por un ordenador[5]. Los ordenadores son tan importantes como lo es la información en la sociedad actual.

En un principio, los límites de las máquinas de cálculo estuvieron impuestos por la tecnología disponible. Cuando en los cursos de introducción a la informática se repasa su historia, siempre se resaltan las importantes aportaciones de Babbage[6]. Babbage trabajó en dos máquinas mecánicas: La Máquina de Diferencias y la Máquina Analítica. Esta última puede considerarse el auténtico precursor de los ordenadores digitales modernos, ya que, en su diseño, disponía de las mismas unidades con que cuentan los ordenadores actuales[7]. Sin embargo, su construcción fue un fracaso a causa fundamentalmente del estado de la tecnología de la época. La Máquina Analítica nunca se concluyó.

No fueron mejor las cosas cuando años después se utilizó la tecnología electrónica. Entre 1943 y 1946 se contruyó el ENIAC[8], considerado como el primer ordenador electrónico de propósito general de la historia. El ENIAC se estropeaba, de media, una vez cada siete minutos, así que cuando conseguía terminar un cálculo nunca se estaba seguro de la corrección del resultado.

5. En realidad, los ordenadores almacenan datos y no informaciones. Los datos se refieren generalmente a situaciones reales y se representan por medio de símbolos. Al ser interpretados, adquieren un significado, que se puede llamar información. No es posible obtener información del conjunto de datos disponibles sin conocer el contexto en el cual éstos adquieren un significado: es el contexto el que confiere significado a los datos.

6. Charles Babbage, Teignmouth, Devonshire, Gran Bretaña, 26-12-1791 – Londres, 18-10-1871. Matemático e ingeniero, profesor de matemáticas en la Universidad de Cambridge.

7. La Máquina Analítica tenía dispositivos de entrada basados en las tarjetas perforadas de Jacquard (Lyon, 1752 - Oullins, Rhône, 1834), un procesador aritmético que calculaba números, una unidad de control que determinaba qué tarea debía ser realizada, un mecanismo de salida que imprimía en papel los resultados y una memoria donde los números podían ser almacenados hasta ser procesados. Idéntica organización que los ordenadores actuales. Se considera que la Máquina Analítica de Babbage fue el primer ordenador del mundo.

8. ENIAC: Electronic Numerical Integrator And Computer. Sus constructores fueron J. Presper Ecker y J. Mauchly, de la Escuela Moore de la Universidad de Pennsylvania. Contenía 18.000 válvulas de vacío y unos 1.500 relés, cuando se ponía en marcha provocaba una bajada general de tensión en la red eléctrica de la ciudad de Filadelfia, donde estaba instalado. El ENIAC pesaba más de 30 toneladas y requería para su instalación más de 140 m^2 de superficie [Goldstine96].

Afortunadamente, los avances de la electrónica[9] han permitido la rápida evolución de los ordenadores a modelos cada vez más fiables, más potentes, de mayor capacidad, de menor consumo, de menores dimensiones y más económicos, como es sabido. Sin embargo, esta evolución no ha conseguido romper algunas barreras de las máquinas de cálculo.

Un ejemplo típico, que pone de manifiesto los límites en la capacidad de cálculo de los ordenadores, es el denominado "problema del viajante de comercio", que se plantea de la manera siguiente[10]: *Un viajante de comercio debe visitar a un conjunto de clientes que residen en lugares diferentes separados por distancias conocidas; se desea encontrar el itinerario más corto que le permita visitar una sola vez a todos los clientes y volver al punto de partida*.

Se trata de un problema real y de evidente interés económico para el viajante y su empresa. Es posible idear un algoritmo sencillo que resuelva el problema. En lo fundamental, la estrategia consiste en generar todos los itinerarios posibles y calcular la distancia recorrida en cada uno de ellos para, finalmente, seleccionar áquel que tenga la distancia más corta.

El algoritmo propuesto debe generar todas las posibles ordenaciones de los elementos del conjunto de clientes, es decir, las permutaciones del conjunto. El número de ordenaciones posibles de un conjunto con n elementos viene dado por la expresión $n!$, esto es, el factorial del número de elementos del conjunto. Así que, si el ordenador utilizado tarda un tiempo T en procesar una ordenación, el tiempo que necesita[11] nuestro algoritmo es $n!$ veces T.

9. En diciembre de 1947, los físicos John Bardeen, Walter Brattain y William Schockley, que trabajaban en los Laboratorios Bell, desarrollaron el primer transistor. En 1958 Jack Kilby desarrolló un prototipo de circuito integrado para Texas Instruments, mientras que Robert Noyce lo hizo para Fairchild Semiconductor. En 1971 se desarrolló el primer microprocesador, el Intel 4004. El periodo de 1960 a 1965 es el de los circuitos integrados a baja escala (SSI: Small Scale Integration), con hasta centenares de componentes. Desde 1965 hasta 1975 dura el periodo de media y gran escala de integración (MSI-LSI: Medium and Large-Scale Integration), con hasta 10.000 componentes. En 1975 empieza el periodo de la alta escala de integración (VLSI: Very Large-Scale Integration), con más de 10.000 componentes.

10. Formalmente, el problema se enuncia sobre un grafo no dirigido con pesos. Se trata de encontrar un ciclo con coste mínimo que visite cada uno de los vértices del grafo exactamente una vez. Alternativamente, se puede plantear como la búsqueda de un ciclo de Hamilton de coste mínimo. La referencia a los ciclos de Hamilton aparece por primera vez en un estudio de Euler, sin embargo fue popularizado en el año 1839 por Hamilton en su juego "Alrededor del mundo" [Skiena98].

11. Es posible hacer un cálculo más ajustado teniendo en cuenta que un camino y su inverso tienen la misma distancia, es decir, sólo es necesario estudiar la mitad de los caminos, ademáas el punto

Es posible hacerse una idea del tiempo de cálculo requerido con un ejemplo. Suponiendo que un programa que implemente el algoritmo en un determinado ordenador, necesitase tan sólo un milisegundo para resolver el problema con diez clientes, entonces el mismo programa, en el mismo ordenador, necesitaría para resolver el problema con 50 clientes alrededor de $8{,}38 \times 10^{55}$ milisegundos o, también de forma orientativa, $2{,}65 \times 10^{45}$ años. Una cantidad mayor que la que representa la edad del universo[12].

Parece que no hay manera de resolver el problema del viajante de comercio sin probar todos los itinerarios posibles y calcular su peso. Un algoritmo más elaborado podría eliminar algunos itinerarios especialmente malos, pero en cualquier caso, se tendría que examinar un número exponencial de itinerarios antes de llegar a la conclusión.

Por otra parte, es muy improbable que el aumento exponencial de la velocidad de los ordenadores a consecuencia de la evolución de la electrónica[13] tenga un impacto significativo en la capacidad del ordenador para resolver problemas como el del viajante de comercio. Recientemente, el profesor Echenique anunciaba[14] que como resultado de una investigación en la que ha participado, se podría aumentar en casi 100.000 veces la velocidad de la electrónica actual. Admitiendo que este incremento se pudiera trasladar directamente a la capacidad de cálculo de los ordenadores, nuestro programa, en uno de esos ordenadores del futuro, necesitaría más de 10^{40} años.

de comienzo está fijado, por lo tanto se trata de ordenar n - 1 puntos, resultando un tiempo proporcional a $(n - 1)!/2$.

12. El proyecto WMAP (Wikinson Microwave Anisotropy Probe) de la NASA estimó la edad del universo en $(13{,}7 \pm 0{,}2) \times 10^9$ años.

13. En 1965 Gordon E. Moore (1929 Pescadero, California) afirmó en un artículo aparecido en la revista Electronics [Moore65], que "la complejidad de los componentes se ha multiplicado aproximadamente por dos cada año. A corto plazo, se puede esperar que esta tasa se mantenga, o incluso que aumente. A largo plazo, la tasa de aumento es un poco más incierta, aunque no hay razón para creer que no permanecerá constante por lo menos durante diez años". Esta afirmación se conoce como Ley de Moore. En 1975, Moore modificó su propia ley al afirmar que el ritmo bajaría, y que la capacidad de integración se duplicaría aproximadamente cada 24 meses. En los años 80, sus compañeros de Intel modificaron la ley para tener en cuenta que el incremento en la capacidad de integración permitía también un aumento en la frecuencia de reloj, de tal manera que en la actualidad lo que se entiende como Ley de Moore es la afirmación de que "cada 18 meses la potencia de los ordenadores se duplica" [Mollick06].

14. "Nature dedica su portada a una investigación en la que participa Etxenike", *El Diario Vasco* (San Sebastián), 25-10-2007. La publicación a la que hace referencia el periódico se encuentra en [Echenique07]. Pedro Miguel Echenique es Doctor Honoris Causa por la Universidad Pública de Navarra (ene. 2008).

Existe un gran conjunto de problemas que requieren de un tiempo muy grande para su resolución por un ordenador[15], independientemente de la tecnología de este y, lo que es peor, como se va a ver, existen problemas para los cuales no hay programas que los puedan resolver.

Los estudios acerca de las posibilidades de las máquinas de cálculo comenzaron en la decada de los años 1930, antes de que existieran los ordenadores. En 1936, Turing[16], siendo estudiante de postgrado, publicó un artículo en el que describía una máquina teórica, la Máquina de Computación Lógica, que posteriormente se denominó Máquina de Turing, que podía particularizarse para llevar a cabo cualquier cálculo realizable por alguna máquina real. Este artículo, junto con los trabajos[17] de Church[18], Kleene[19] y Post[20], en la década de 1930, establecieron las bases teóricas de la informática.

Los trabajos de Turing tienen su origen en los de Hilbert[21] que pretendía encontrar un sistema axiomático, del que pudieran deducirse la verdad o falsedad de cualquier proposición matemática[22]. En concreto, Hilbert se preguntaba si existiría un modo de

15. Típicamente se trata de problemas para los cuales los algoritmos existentes requieren tiempos de ejecución que crecen, al menos, de forma exponencial con el tamaño del conjunto de datos sobre el que se aplican. Los algoritmos conocidos para el problema del viajante de comercio requieren de un tiempo proporcional a $n!$, siendo n el número de clientes. A pesar de que el factorial de n crece más deprisa que 2^{cn}, cualquiera que sea la constante c, en este caso, como en otros similares, se habla también de crecimiento exponencial, considerando esta función como una cota inferior para el crecimiento [Hopcroft02].

16. Alan Mathison Turing, Londres, 23-6-1912 – Wilmslow, Cheshire, Gran Bretaña, 7-6-1954. En 2001, en el parque Sackville de Manchester, se inauguró la primera estatua conmemorativa, que le define como "Padre de la ciencia informática, matemático, lógico, quebrador de códigos en tiempos de guerra, víctima del prejuicio". Obtuvo el grado Ph.D. en lógica, álgebra y teoría de números, en la Universidad de Princeton (New Jersey). En su estancia en Princeton publicó su famoso artículo *On computable numbers...* [Turing36], en el que elaboró el concepto teórico de una máquina de calcular universal.

17. Publicados en [Church36], [Kleene36], [Post36].

18. Alonzo Church, Washington DC, 14-6-1903 – Hudson, Ohio, 11-8-1995. Matemático y uno de los mejores lógicos de su tiempo. Fue profesor de Turing en la Universidad de Princeton.

19. Stephen Kleene, Hartford, Connecticut, Estados Unidos, 5-1-1909 – Madison, Wisconsin, 25-1-1994.

20. Emil Leon Post, Augustów, Polonia, 11-2-1897 – New York, 21-4-1954.

21. David Hilbert, Königsberg, Prusia Oriental, 23-1-1862 – Göttingen, Alemania, 14-2-1943. Uno de los más influyentes matemáticos del siglo XIX y principios del XX.

22. El proyecto se remonta a Gottfried Leibniz (Leipzig, 1-7-1646 – Hannover, 14-11-1716), quien en el siglo XVII, después de construir con éxito una máquina mecánica de cálculo, soñaba con construir una máquina que pudiera manipular símbolos para determinar si una frase en matemáticas es un teorema.

determinar si cualquier fórmula del cálculo de predicados de primer orden, aplicada a enteros, es verdadera. Gödel[23] en el año 1931 ya había demostrado con su *teorema de incompletitud*, que la pretensión de Hilbert era irrealizable, ya que probó que en todo sistema finito de axiomas existen proposiciones indecidibles, es decir, proposiciones cuya validez o refutación no se puede decidir dentro del sistema.

El objetivo de Turing era determinar con precisión cuál era la frontera entre lo que podía y lo que no podía calcularse mediante un *procedimiento efectivo*[24]. Si las conjeturas de Hilbert hubieran sido ciertas, los ordenadores habrían podido construir de forma automática algún programa para un problema dado. Turing establece que los ordenadores sólo pueden realizar aquellas labores para las que previamente han sido programados. No es posible una revuelta de los ordenadores sin una confabulación de los programadores[25].

La Máquina de Turing[26] posee la misma capacidad que los ordenadores de hoy en día, al menos desde el punto de vista de lo que son capaces de hacer los ordenadores, por lo que sus conclusiones son aplicables también a las máquinas reales que existen en la actualidad. El artículo de Turing ya mencionado fijó los límites de la informática, pues demostró que, simplemente, no es posible resolver algunos problemas con ningún tipo de ordenador. Las máquinas de Turing que siempre se paran son un buen modelo de algoritmo. Si existe un algoritmo para resolver un problema dado, entonces se dice que el problema es "decidible", es decir, puede ser resuelto por un ordenador.

23. Kurt Friedrich Gödel, Brno (Brünn) Austria-Hungría (ahora República Checa), 28-4-1906 – Princeton, New Jersey, 14-1-1978. Lógico, matemático y filósofo. En 1931 publicó un artículo [Gödel31] en el que demostraba que en todo sistema axiomático que tiene por lo menos la complejidad de la aritmética, hay proposiciones metamatemáticas que no pueden probarse o refutarse mediante deducciones formales basadas en los axiomas del sistema. O dicho sintéticamente: Toda formulación axiomática de teoría de los números incluye proposiciones no decidibles. Esta afirmación se conoce como Teorema de la incompletitud de Gödel.

24. En matemáticas se considera que un método o procedimiento es efectivo para obtener un resultado cuando se cumple que: (i) El método puede ser expresado mediante un número finito de instrucciones concretas; (ii) Cada instrucción puede ser expresada por un número finito de símbolos; (iii) El método puede ser seguido sin error para conseguir el resultado en un número finito de pasos; (iv) El método puede ser seguido (al menos teóricamente) por un humano sin más ayuda que un papel y lápiz, y no exige ninguna habilidad o inteligencia especial por parte de la persona que lo ejecuta.

25. HAL 9000 permanecerá para siempre en el espacio de la ciencia ficción.

26. La Máquina de Turing consta de una cinta infinita y de un cabezal lector/escritor que se mueve atrás y adelante, leyendo y escribiendo uno a uno símbolos en la cinta. Las acciones del cabezal están definidas (programadas) por una tabla de transiciones cuya entrada está determinada por el estado y el símbolo bajo el cabezal en la cinta y su contenido establece el nuevo estado, el símbolo a escribir en la cinta y el movimiento del cabezal (atrás o adelante).

Todas las propuestas serias sobre modelos de computación tienen el mismo potencial. La suposición, no demostrada, de que cualquier forma general de computación permitirá calcular únicamente las funciones recursivas parciales, o, lo que es lo mismo, lo que las máquinas de Turing o los ordenadores modernos pueden calcular, se conoce como *hipótesis de Church* o *tesis de Church-Turing*[27].

Aunque es delicado demostrar que un problema determinado es indecidible, es bastante sencillo comprender por qué casi todos los problemas han de ser indecidibles para los sistemas relacionados con la programación. Para resolver un problema mediante un ordenador es necesario construir un programa que lo resuelva. Creo que se estará de acuerdo en que existen infinitos problemas y que además estos no pueden ser ordenados de acuerdo a ningún criterio. Por otra parte, los programas son numerables, cualquier programa no es más que una cadena finita construida con un conjunto finito de símbolos. Es decir, es posible ordenarlos de acuerdo a su longitud y, para los que tengan la misma longitud, por orden alfabético. En resumen, el conjunto de programas es infinito[28] y numerable, mientras que el de problemas es infinito. En consecuencia, existen infinitos problemas más que programas. La única razón por la que la mayoría de los problemas parece que son decidibles, es porque los problemas elegidos al azar rara vez tienen interés. Más bien, se tiende a considerar problemas bastante sencillos y bien estructurados, y estos problemas suelen ser decidibles. Sin embargo, incluso entre los problemas que nos interesan, y que pueden enunciarse clara y brevemente, es posible encontrar muchos indecidibles.

Uno de esos problemas indecidibles es el de "parada de la Máquina de Turing": no es posible decidir algorítmicamente si una Máquina de Turing dada llegará a pararse o no partiendo de una configuración inicial determinada. El modelo del "Juego de la Vida" de Conway[29] permite hacer una aproximación más gráfica al problema. Desde el punto de vista teórico, el Juego de la Vida tiene la misma potencia que la Máquina de Turing.

27. El trabajo de Church precedió al famoso trabajo de su alumno Turing sobre el problema de parada, que también demostró la existencia de problemas irresolubles por dispositivos mecánicos. Church relaciona la noción de ser efectivamente calculable con el cálculo lambda de la lógica matemática (tesis de Church), mientras que Turing lo hace con la Máquina de Turing (tesis de Church-Turing). Ambos modelos son equivalentes, pero el estudio de Turing es mucho más accesible e intuitivo por tener un enfoque operacional.

28. Los sistemas operativos modernos permiten ejecutar programas de cualquier longitud, aunque ésta sobrepase la capacidad de la memoria del ordenador sobre el que se ejecutan, por lo que se puede obviar un límite en la longitud de los programas impuesto por la capacidad del ordenador; de esta forma, el conjunto de programas será infinito.

29. *Game of Life* es un autómata celular [Sarkar00] presentado por el matemático británico John Horton Conway (Liverpool, 26-12-1937) en 1970 [Gardner70].

El universo del Juego de la Vida es un tablero plano cuadriculado sin límite en sus dimensiones. Una celda puede estar viva, lo que se representa con el color negro, o muerta, lo que se representa con el color blanco. Existe un conjunto de reglas sencillas que, en función del estado de una celda y el de sus ocho celdas vecinas, determinan la supervivencia, muerte y nacimiento de esa celda de una generación a la siguiente[30]. Todas las decisiones sobre las celdas ocurren simultáneamente. Un movimiento o generación incluye una decisión en cada una de las celdas y una historia de vida se construye generación a generación a partir de una población inicial.

Existen poblaciones iniciales que se extinguen rápidamente, es decir, todas sus celdas mueren en unas pocas generaciones, mientras que otras poblaciones se extinguen después de miles de generaciones, también las hay que no se extinguen porque entran en una secuencia cíclica de generaciones y aún hay otras, que, se cree, evolucionan sin fin. El problema es que, hasta la fecha, no se ha podido construir un algoritmo que, dada una población inicial, determine si se va a extinguir o no. Sólo es posible la simulación. Si después de varias generaciones la población se extingue, obviamente la población inicial pertenece a la clase de las extinguibles, pero si después de cientos de miles de generaciones la población no se ha extinguido, no se tiene ninguna garantía de que eso no pueda ocurrir quizás en unas pocas generaciones más. Hay poblaciones iniciales para las cuales no hay un método capaz de clasificarlas como extinguibles o no, o dicho de otra manera, el problema de saber si una población particular acabará o no por extinguirse es indecidible.

El problema va más allá de lo que es un simple juego, ya que tiene relación con muchos problemas de la física, la biología, la economía, las matemáticas e incluso de la filosofía. Cuando un sistema tiene un comportamiento suficientemente simple, puede deducirse una fórmula matemática que lo represente. La respuesta a un problema se puede obtener entonces por un cálculo. Estos sistemas se denominan *computacionalmente reducibles*. Sin embargo, existen sistemas tan complicados que no puede darse una descripción resumida de su evolución, que sólo se podrá determinar por medio de la simulación. En estos sistemas es posible encontrar problemas cuya respuesta está condicionada a que se alcance o no un estado determinado, problemas que son asimilables al de la extinción en el Juego de la Vida y que son igualmente indecidibles.

30. Las reglas son: (1) Supervivencia, cada celda con dos o tres celdas vecinas vivas sobrevive hasta la siguiente generación; (2) Muerte, cada celda con cuatro o más celdas vecinas vivas muere por sobrepoblación. También muere, por aislamiento, toda celda que no tenga celdas vecinas vivas o que tenga a lo más una; (3) Nacimiento, toda celda no viva que tenga exactamente tres celdas vecinas vivas estará viva en la siguiente generación.

En 1969, Cook[31] extendió los estudios realizados por Turing acerca de lo que se puede computar y lo que no. Cook consiguió separar los problemas que se pueden resolver eficientemente en un ordenador, de aquéllos que, aunque en principio pueden ser resueltos, en la práctica llevaría tanto tiempo que los ordenadores no tendrían ninguna utilidad, salvo para la resolución de unos pocos casos particulares. Los problemas que pertenecen a este último tipo se dice que son "intratables". De esta forma, reciben el nombre de "tratables" los problemas que un ordenador puede resolver en un tiempo proporcional a alguna función que crezca lentamente con el tamaño de la entrada. Se supone que las funciones polinómicas son de "crecimiento lento", mientras que se considera que las funciones que crecen más rápido que cualquier función polinómica crecen con excesiva rapidez. Los problemas prácticos que requieren un tiempo polinómico son casi siempre resolubles en un tiempo tolerable, mientras que los que requieren un tiempo exponencial[32], en general no se pueden resolver, excepto en casos sencillos.

Cook presentó el primer problema NP-completo, el de "satisfacción de una expresión booleana", e introdujo la clase de problemas NP-completos[33], un conjunto de problemas intratables. Karp[34] demostró que los problemas NP-completos no son fenómenos aislados, pues aparecen en muchos problemas combinatorios difíciles estudiados desde hace años por la investigación operativa y otras disciplinas. El problema del viajante de comercio es uno de ellos. Hay más de 3000 problemas NP-completos

31. Stephen Arthur Cook, Buffalo, Nueva York, 14-12-1939. Premio Turing en 1982 por sus aportaciones en el campo de la complejidad computacional. Formalizó la noción de NP-completitud en un famoso artículo [Cook71] que contenía el Teorema de Cook, una prueba de que el problema de satisfacibilidad booleana (SAT) es NP-completo.

32. Se habla de crecimiento exponencial para referirse a cualquier tiempo de ejecución que sea mayor que todos los polinomios.

33. La clase P es el conjunto de problemas resolubles (aceptados) por una máquina de Turing determinista en tiempo polinómico. La clase NP contiene los problemas resolubles (aceptados) por una máquina de Turing no determinista en tiempo polinómico. Es evidente que $P \subseteq NP$ ya que las máquinas deterministas se pueden considerar casos particulares de las no deterministas. Sin embargo, el problema $P = NP$ no se ha resuelto, es decir, no se sabe si todo lo que se puede hacer en tiempo polinómico con una máquina de Turing no determinista se puede hacer con una máquina de Turing determinista en tiempo polinómico, quizá con un polinomio de grado más alto. NP contiene el conjunto de problemas NP-completo. Un problema p está en la clase NP-completo, si está en NP y existe una reducción en tiempo polinómico de todos los problemas de NP al problema p. Hasta la fecha, no se ha encontrado una solución polinómica para ningún problema NP-completo. Si un día se encontrara una solución, entonces habría solución de esas características para todos los problemas en NP y en consecuencia $P = NP$.

34. Richard Manning Karp, Boston, Massachusetts, 3-1-1935. Premio Turing en 1985 por sus contribuciones a la teoría de algoritmos, la identificación de problemas computables en tiempo polinomial y a la teoría de NP-completitud. Publicó un artículo [Karp72] en el que demostró que 21 problemas conocidos, como el conjunto independiente, el recubrimiento de nodos, el circuito hamiltoniano y el problema del viajante de comercio, son NP-completos.

conocidos[35] que comprenden temas de la geometría computacional, la teoría de grafos, el diseño de redes, la partición de conjuntos, el almacenamiento y recuperación de la información, el ordenamiento de tareas, la programación matemática, el álgebra, la lógica, la teoría de autómatas, la optimización de programas y un buen número de juegos[36].

Aún no está claro que los problemas en NP-completo constituyan una clase diferenciada de la de los problemas tratables. El hecho de que hasta el momento no se conozcan algoritmos con tiempos de ejecución polinomiales para problemas en NP-completo, no implica que no se puedan desarrollar en el futuro. Sin embargo, la gran cantidad de expertos que han trabajado el tema en todos estos años, que no han sido capaces de dar con un algoritmo de tiempo polinómico para algún problema en NP-completo, parece avalar la idea de que ese algoritmo no es posible. Tampoco hay una prueba formal al respecto. Si alguien encontrara un algoritmo de tiempo polinómico para alguno de los problemas en NP-completo se podrían obtener algoritmos para todos los problemas clasificados hoy en NP-completo[37]. Uno de los "Siete Problemas del Milenio" aún sin resolver, el denominado "P versus NP", consiste en obtener una demostración formal de la existencia, o no, de dos clases diferenciadas[38].

La computación cuántica puede hacer que en el futuro este dilema pase a un segundo plano, al menos desde el punto de vista práctico. Si un ordenador clásico equivale a una máquina de Turing, un ordenador cuántico equivale a una máquina de Turing indeterminista. Por lo tanto, los problemas en NP-completo podrán ser resueltos en un ordenador cuántico en tiempo polinomial.

Una forma de reducir el tiempo necesario para un cálculo es utilizar más de un ordenador en ese cálculo. Este es uno de los motivos[39] para la utilización de sistemas distri-

35. En [Garey79] se puede encontrar una buena revisión de problemas NP-completos y qué casos especiales se pueden resolver en tiempo polinómico.

36. Mastermind, Sudoku, Tetris, SameGame son algunos de los más conocidos. Gran parte del atractivo de estos juegos radica precisamente en el desconocimiento de una estrategia fija (algoritmo) para su resolución en un tiempo razonable.

37. Si se pueden traducir en tiempo polinómico los casos de un problema a los casos de otro que tenga la misma respuesta, se dice que el primer problema es *reducible* al segundo en tiempo polinómico. Pues bien, los problemas en NP-completo conocidos son reducibles entre sí en tiempo polinómico.

38. El Clay Mathematics Institute (Cambridge, Massachusetts) tiene establecido un premio de un millón de dólares para el que resuelva alguno de los Siete Problemas del Milenio. "Cualquiera que llegue con un algoritmo que resuelva eficientemente un problema NP-completo ganará el millón de dólares" ha dicho Cook, "Pero, el problema real es que pienso que no existe ningún algoritmo. El reto en realidad es probar que no existe tal algoritmo".

39. Otros motivos son la posibilidad de intercambio de información, la compartición de recursos, el incremento de la confianza ante posibles fallos y la simplificación del diseño [Tel94].

buidos de ordenadores. *Un sistema distribuido es una instalación en la que una colección de computadores, procesos, o procesadores interconectados, que tienen capacidad de trabajar de forma autónoma, cooperan de alguna manera*. Esta definición engloba las redes de área ancha, las redes de área local, los multiprocesadores y los procesos cooperantes[40]. Sin embargo, como era de esperar[41], la utilización de más ordenadores no rebaja el tiempo total de cálculo en la misma proporción, sino que en algunos casos los problemas de sincronización hacen que el tiempo resultante aumente e incluso que aparezcan problemas de sincronización irresolubles.

Volviendo al problema del viajante de comercio, se puede hacer que en un sistema distribuido cada ordenador calcule la distancia correspondiente a un trayecto. Un algoritmo para hacer este cálculo necesitaría tan sólo un tiempo proporcional al número de clientes a visitar. Esto representa un coste lineal asumible por cualquier ordenador. Sin embargo, aparecen problemas inexistentes cuando se utilizaba un solo ordenador. Por ejemplo, una vez obtenidas las distancias de todos los itinerarios es inevitable designar un ordenador concreto para que seleccione el itinerario más corto. Es necesario elegir un ordenador entre todos los que componen el sistema distribuido y su identidad debe ser conocida por todos los demás ordenadores, para que éstos le remitan la longitud del trayecto que se les ha asignado. Se denomina "líder" al ordenador seleccionado y el problema perfilado en el párrafo anterior se denomina "elección de líder"[42].

El problema de la elección de líder es muy conocido en computación distribuida y se ha estudiado en profundidad[43]. Pues bien, se sabe que no es posible construir un algoritmo que resuelva el problema si todos los ordenadores del sistema distribuido son idénticos[44]. Esta no es una situación alejada de la realidad, ya que en fábrica todos los

40. En el caso de procesos cooperantes se considera una distribución lógica.

41. Frederick Brooks (Durham, Carolina del Norte, 19-4-1931) en [Brooks75], afirmó: "Añadir personal a un proyecto retrasado lo retrasará aún más". Brooks calcula que considerando las labores de coordinación, el esfuerzo del proyecto se incrementa según la expresión n x (n - 1)/2, donde n es el número de trabajadores. Lo que Brooks afirma sobre trabajadores se puede aplicar también a la colaboración de ordenadores. Algunos resultados sobre el coste de la coordinación de algoritmos distribuidos, en número de mensajes de intercomunicación, corroboran esta idea. Brooks recibió el premio Turing en 1999, por sus contribuciones a arquitectura de computadores, sistemas operativos e ingeniería del software.

42. El problema de la elección de líder fue propuesto por primera vez por LeLann en [LeLann77].

43. En un sistema distribuido algunas tareas deben llevarse a cabo desde un único ordenador del conjunto (iniciar una estructura de datos, generar una salida...). Si no es posible o deseable asignar esta tarea *a priori*, entonces es necesario un algoritmo que elija al proceso. Se pueden encontrar en [Tel94] y [Lynch96] buenos tratados del tema.

44. Imagínense la situación si se tratase de elegir un líder entre un conjunto de personas, todas con el mismo nombre, mismas características físicas, que viven en la misma ciudad, la misma calle, etc.,

ordenadores de un modelo son absolutamente iguales. Sin embargo, es fácil sortear el impedimento[45], simplemente dotando de un identificador diferente a cada uno de los ordenadores del sistema distribuido. Aún así, aparecen nuevos límites. Los ordenadores de un sistema distribuido se comunican entre sí por medio de mensajes a través de alguna red de comunicación que los interconecta con el objetivo de la colaboración. En un sistema distribuido con una red de comunicación *asíncrona*[46] arbitraria, el mínimo número de mensajes que necesita un algoritmo, basado en la utilización de los identificadores de los ordenadores, para resolver el problema de la elección de líder es de N x log$N$+ E, siendo N el número de ordenadores y E el número de canales de comunicación directa entre cualesquiera dos ordenadores existentes en la red. En una red arbitraria se relaciona el valor de E con el cuadrado de N y, en el ejemplo que se viene manejando, el valor de N es el factorial de 50, esto es, aproximadamente 3,04 x 10^{64}.

Los impedimentos para la adopción de la última solución planteada no vienen sólo del manejo de esa cantidad de mensajes, también de la imposibilidad de disponer de ese número de ordenadores. Al parecer, no es factible distribuir un algoritmo, si son necesarios más de n x log n ordenadores, siendo n el tamaño del problema[47].

La programación de sistemas distribuidos es una tarea realmente complicada, ya que el estado global del sistema es desconocido por sus componentes. Además, no siempre es posible establecer un orden total entre los eventos acaecidos en distintos ordenadores. Esto hace que en sistemas distribuidos aparezcan problemas inexistentes en sistemas concentrados.

Con objeto de ilustrar uno de estos problemas, considérese el siguiente escenario: *Andrés y Blanca deben ingresar 100 y 200 euros en la cuenta de Carlos, cuyo saldo es de 300 euros.* Si las operaciones se hacen correctamente, el saldo final en la cuenta de Carlos debería ser de 600 euros.

Si Andrés y Blanca acuden a la sucursal de Carlos y esperan su turno, uno detrás de otro, en ventanilla única, obviamente el saldo final en la cuenta de Carlos será el

es decir, sin ningún rasgo diferenciador, y además sólo pudieran comunicarse entre ellas por medio de mensajes SMS.

45. Es fácil sortear el impedimento en entornos restringidos en los que se tenga control sobre todos los ordenadores del sistema.

46. El sistema de paso de mensajes es asíncrono cuando la operación de envío de un mensaje y la de recepción del mismo mensaje no tienen más restricción temporal que la que impone que la segunda operación, si se produce, debe ser posterior a la primera. Si se asegura que la transmisión del mensaje se produce siempre en un tiempo finito, se habla de sistema asíncrono acotado.

47. Sobre los límites en la computación paralela ver [Greenlaw95].

esperado, independientemente del orden en que hayan sido atendidos. Esta forma de trabajo, en la que las tareas se realizan una detrás de otra, es decir, en forma secuencial, es la forma básica de trabajo de un ordenador, pero si se dispone de varios ordenadores trabajando conjuntamente, es posible que ciertas tareas se realicen de forma paralela.

Operaciones de Andrés:
A1, *envía SMS solicitando saldo de Carlos*;
A2, *recibe SMS con saldo de Carlos*;
A3, *suma 100 al saldo obtenido*;
A4, *envía SMS nuevo saldo de Carlos es el resultado de la suma*.

Operaciones de Blanca:
B1, *envía SMS solicitando saldo de Carlos*;
B2, *recibe SMS con saldo de Carlos*;
B3, *suma 200 al saldo obtenido*;
B4, *envía SMS nuevo saldo de Carlos es el resultado de la suma*.

Supóngase ahora que Andrés, Blanca y Carlos operan en sucursales distintas y que el único medio de comunicación entre éstas es mediante mensajes SMS que portan el saldo de alguna cuenta. Andrés necesitaría solicitar a la sucursal de Carlos el saldo de éste, una vez obtenido el saldo, incrementarlo con el pago que debe realizar y, finalmente, enviar un mensaje con el nuevo saldo a la sucursal de Carlos. De la misma forma debería proceder Blanca. En este caso, según el orden en que se realicen las operaciones, es posible obtener, además del resultado correcto, otros dos resultados diferentes.

Saldo inicial de Carlos: 300.
A1, *envía SMS solicitando saldo de Carlos*
A2, *recibe SMS con saldo de Carlos es 300*
B1, *envía SMS solicitando saldo de Carlos*
B2, *recibe SMS con saldo de Carlos es 300*
B3, *suma 200 al saldo 300*
B4, *envía SMS nuevo saldo de Carlos es 500*
Saldo actual de Carlos: 500.
A3, *suma 100 al saldo 300*
A4, *envía SMS nuevo saldo de Carlos es 400*
Saldo final de Carlos: 400.

Saldo inicial de Carlos: 300.
B1, *envía SMS solicitando saldo de Carlos*
B2, *recibe SMS con saldo de Carlos es 300*
A1, *envía SMS solicitando saldo de Carlos*
A2, *recibe SMS con saldo de Carlos es 300*
A3, *suma 100 al saldo 300*
A4, *envía SMS nuevo saldo de Carlos es 400*
Saldo actual de Carlos: 400.
B3, *suma 200 al saldo 300*
B4, *envía SMS nuevo saldo de Carlos es 500*
Saldo final de Carlos: 500.

Es decir, un programa correcto que se ejecuta secuencialmente en un solo ordenador obtiene siempre el resultado correcto, mientras que un programa que se ejecuta en un sistema distribuido no siempre obtiene el resultado esperado. En este último caso, es necesario desarrollar mecanismos para que el programa obtenga siempre el resultado correcto.

Saldo inicial de Carlos: 300.
A1, *envía SMS solicitando saldo de Carlos*
A2, *recibe SMS con saldo de Carlos es 300*
B1, *envía SMS solicitando saldo de Carlos*
A3, *suma 100 al saldo 300*
A4, *envía SMS nuevo saldo de Carlos es 400*
Saldo actual de Carlos: 400.
B2, *recibe SMS con saldo de Carlos es 400*
B3, *suma 200 al saldo 400*
B4, *envía SMS nuevo saldo de Carlos es 600*
Saldo final de Carlos: 600.

Una forma de resolver el problema es bloquear las operaciones sobre la cuenta de Carlos desde que se envía el mensaje con su saldo hasta que se recibe el mensaje con su saldo actualizado, no permitiendo en ese plazo operaciones desde otro usuario. Una vez enviado el saldo solicitado a Andrés, si Blanca lo solicita a su vez, no se contestará a su mensaje hasta que Andrés haya actualizado la cuenta de Carlos. En ese tiempo Blanca quedará a la espera.

Un problema similar al expuesto se puede dar en sistemas distribuidos, cuando desde varios ordenadores se intenta acceder a algún recurso común que no puede ser utilizado al mismo tiempo por varios de ellos. Este problema se denomina de la "*exclusión mutua*"[48]. La solución, como se ha visto, consiste en hacer que una vez que el recurso está siendo utilizado por un usuario, todos los demás que deseen acceder a él queden a la espera hasta que vuelva a estar libre.

Muchas aplicaciones de sistemas distribuidos utilizan esta solución. Sin embargo, se sabe que en un entorno en el que se puedan producir fallos, este mecanismo no funciona correctamente. Si una vez que Andrés ha obtenido el saldo de Carlos, y Blanca está a la espera, Andrés se retrasa en enviar el SMS con el nuevo saldo, no es posible

48. El problema de la exclusión mutua fue descrito y resuelto por primera vez por Edsger Wybe Dijkstra (Rotterdam, 11-5-1930 – Nuenen, Holanda, 6-8-2002) en [Dijkstra65]. Dijkstra recibió el premio Turing en 1972, por su contribución a la "ciencia y arte" de los lenguajes de programación.

distinguir si se debe a que Andrés es muy lento o a que Andrés no continúa con la operación, por ejemplo porque se ha quedado sin batería en su móvil, en cuyo caso Blanca quedaría en una espera indefinida. La conclusión es que no existen algoritmos basados en esperas que resuelvan el problema, si los fallos se pueden producir en un ordenador en el momento en que dispone del acceso exclusivo al recurso.

Uno de los motivos para el desarrollo de sistemas distribuidos es conseguir sistemas seguros que mantengan la funcionalidad a pesar de que alguno de sus componentes falle. Esto es importante, por ejemplo, en los sistemas de control de tráfico aéreo, y también en los sistemas de bases de datos de las entidades financieras. Una forma de conseguir seguridad es la replicación, es decir, disponer de varios ordenadores realizando la misma tarea en un sistema distribuido. La evolución de todos los ordenadores del sistema debe ser la misma y las decisiones que tomen también, de tal manera que en caso de fallo de uno de ellos, cualquiera de los demás pueda continuar la tarea en el mismo punto. Lamport presentó una abstracción del problema denominada "problema de los generales Bizantinos", con la que demostró que en un sistema distribuido con tres ordenadores, que se comunican por medio de mensajes, que se transmiten sin fallos a través de un sistema de comunicación asíncrono, si se produce un fallo en uno de los ordenadores, no es posible determinar cuál de los tres lo ha hecho[49].

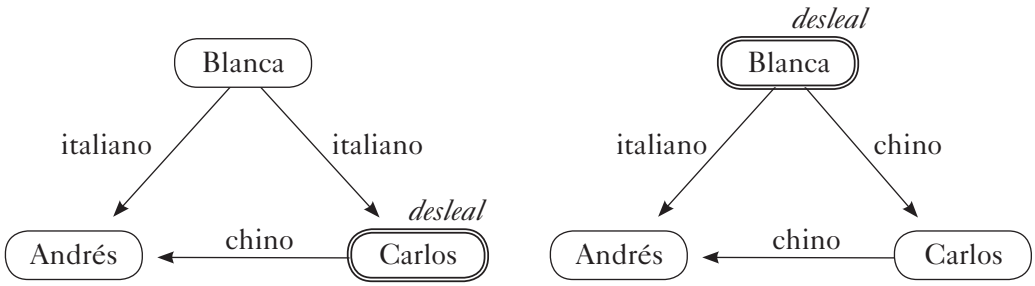
La siguiente historia ilustra el problema. Supóngase ahora que Andrés, Blanca y Carlos desean ir a cenar juntos, pero al elegir restaurante dudan entre dos opciones, acudir al restaurante italiano o bien reunirse en el restaurante chino de su localidad[50]. En principio, dado que no se está hablando de gustos, ambas opciones son igualmente válidas, siempre que los tres acudan al mismo restaurante. Andrés y Carlos han dejado que Blanca decida y que les comunique su decisión mediante un SMS que ellos a su vez se retransmitirán. Sin embargo, es posible que alguno de ellos prefiera una cena a dos y no se comporte de forma leal.

Si Blanca es leal y decide ir al italiano con los dos, enviará sendos mensajes a Andrés y Carlos indicando esta opción. Entonces, si Carlos es desleal, puede enviar un mensaje a Andrés indicándole que la reunión tendrá lugar en el chino. De esta forma, Andrés recibe dos citas contradictorias y no sabrá a qué restaurante acudir, a menos que pueda

49. Lamport presentó el problema de los generales Bizantinos en [Lamport82], donde también dio algunas soluciones y los primeros resultados de imposibilidad del problema. En realidad, demostró que bajo determinadas condiciones, sólo es posible el consenso si en un sistema de $2 \times m + 1$ ordenadores, pueden fallar a lo más m ordenadores.

50. Se supone que los personajes habitan en una extraña población en la que sólo hay un restaurante de los tipos de cocina mencionados.

descubrir al compañero desleal. Supóngase ahora que Blanca es la desleal. Blanca ha decidido acudir al chino, pero envía un mensaje a Andrés indicando que la reunión será en el italiano, mientras que a Carlos le enviará un mensaje dirigiéndole al chino que, si Carlos es leal, reenviará a Andrés. En ambos casos, Andrés recibe los mismos dos mensajes, "italiano" desde Blanca y "chino" desde Carlos. Es decir, Andrés, en base a la información recibida, no puede decidir cuál de sus compañeros le está engañando. Trasladado el problema a la informática, un ordenador "desleal" es un ordenador que ha fallado; el ejemplo establece que, en determinadas condiciones[51], un sistema de tres ordenadores no es suficiente para localizar el fallo en uno de ellos.



En sistemas distribuidos, está abierta una línea de investigación que consiste en descubrir bajo qué condiciones existe o no solución para un problema dado. Las aportaciones se denominan "resultados de imposibilidad". Un pequeño cambio en el modelo del sistema distribuido puede modificar radicalmente el problema hasta el punto de no ser ya resoluble. Hay una gran cantidad de parámetros que se pueden modelar, como la topología de la red de comunicación, las características de los canales de comunicación, el grado de sincronización, el tipo de fallos en las comunicaciones y en los procesos, y el número y tamaño de los mensajes, entre otros. En un artículo publicado en el año 2003, se hablaba ya de "centenares de resultados de imposibilidad"[52]. Los resultados de imposibilidad son importantes porque ayudan a entender la naturaleza de la computación distribuida: cuál es la razón por la que un problema es difícil de resolver, cuál es la razón por la que un modelo es más potente que otro. También indican el punto en el que no se debe seguir buscando soluciones mejores. Si a pesar de los resultados de imposibilidad se debe resolver un problema, éstos indican cómo modificar el entorno para poder llegar a una solución.

51. Se supone un sistema distribuido que consta de tres ordenadores, comunicaciones asíncronas con canales fiables, mensajes sin firma y fallos bizantinos.

52. F. Fich, E. Ruppert; *Hundreds of impossibility results for distributed computing*; Distributed Computing, vol. 16, 2003, pp. 121-163.

Sin embargo, se han desarrollado y comercializado muchos sistemas informáticos que contienen problemas algorítmicos no resueltos, incluso algunos "imposibles", lo que provoca fallos más o menos frecuentes que se resuelven con un "volver a arrancar el sistema". Además de los costes que tienen en sí estas "caídas" del sistema, ésta es una práctica que no sería admisible en ninguna otra rama de la ingeniería. Se está primando una evolución rápida, ofreciendo más y distintos productos antes que su calidad.

En la Universidad, el I+D en el área de la informática no puede entrar en esta espiral frenética del desarrollo de más y más productos de vida efímera. El objetivo no debe ser competir con las empresas de *software*, porque además no puede hacerlo, dado los inmensos recursos que poseen; más bien se debe trabajar en las bases del conocimiento que permitan desarrollar sistemas informáticos fiables y de calidad. Ramón y Cajal[53] dio un consejo a los futuros investigadores: *Otro de los vicios del pensamiento que importa combatir a todo trance es la falsa distinción entre ciencia teórica y ciencia práctica, con la consiguiente alabanza de la última y el desprecio sistemático de la primera... Cultivemos la ciencia por sí misma, sin considerar por el momento las aplicaciones. Estas llegan siempre, a veces tardan años, a veces siglos. Poco importa que una verdad científica sea aprovechada por nuestros hijos o por nuestros nietos.* Hoy nadie duda de que la investigación básica es necesaria. Pero si no la hace la Universidad, ¿quién la va a hacer?

En el año 1964, cuando los miembros de la Asociación Americana de Bibliotecas conocieron el ordenador UNIVAC, hicieron un comunicado en el que afirmaban que *el ordenador no es más que una cosa estúpida pero veloz, no tiene imaginación, no puede originar una acción. Sólo es, y será, una herramienta para los humanos.* La misma idea había sido expresada, esta vez en positivo, por Ada Byron: *La Máquina Analítica no pretende originar nada. Sin embargo, puede hacer cualquier cosa que sepamos ordenarle.* Los límites de los ordenadores, al igual que los de la Máquina Analítica, están en nuestros propios límites. Todo lo que hacen los ordenadores actuales es porque los humanos lo sabemos hacer, y aún harán más tareas cuando sepamos cómo las realizamos.

Cada nueva tarea encomendada a los ordenadores, nos ha obligado a su estudio en profundidad, hasta el punto de descubrir aspectos hasta entonces desconocidos, a pesar de que muchas de esas tareas las veníamos realizando desde hace mucho tiempo. Cuando pretendimos que los ordenadores hicieran traducción automática de idiomas, descubrimos que desconocíamos muchos aspectos del proceso y desarrollamos nuevas

53. Santiago Ramón y Cajal (Petilla de Aragón, 1-5-1852 – Madrid, 17-10-1934). Premio Nobel de Medicina en 1906, por su contribución al conocimiento de los mecanismos que gobiernan la morfología y los procesos conectivos de las células nerviosas. El texto que se cita está extraído del discurso de ingreso de Ramón y Cajal en la Academia de Ciencias en 1897 [CSIC82].

herramientas para la representación y la interpretación del lenguaje, que facilitaron la programación de la tarea en los ordenadores, pero también nos ayudaron a comprender mejor el proceso. Cuando pretendimos realizar sistemas expertos automáticos en distintas áreas del saber, descubrimos que los humanos utilizamos en nuestros razonamientos, para la toma de decisiones, lógicas y modelos complejos, de los que hasta entonces no habíamos dotado a nuestros ordenadores. Cuando quisimos construir sistemas de lectura automática, llegamos a estudiar la dinámica ocular en el proceso de lectura y descubrimos algunos aspectos que influyen en su rendimiento.

La investigación para llevar más lejos las fronteras de la informática nos está ayudando a conocer mejor nuestras propias capacidades. En el futuro, los ordenadores no tendrán más límites que nuestros propios límites.

He dicho.

Referencias

[Brooks75] F. Brooks; *The mythical man-month: Essays on Software Engineering*; Addison-Wesley, 1975.

[Cook71] S. Cook; *The Complexity of Theorem Proving Procedures*; Third ACM Symposium on Theory of Computing, 1971, pags. 151-158.

[Copeland00] B.J. Copeland; *The Modern History of Computing*; Stanford Encyclopedia of Philosophy, http://plato.stanford.edu, 2000.

[CSIC82] S. Ramón y Cajal; *Reglas y consejos sobre investigación científica. Tónicos de la voluntad*; Consejo Superior de Investigaciones Científicas, Madrid, 1982.

[Church36] A. Church; *An undecidable problem in elementary number theory*; American Journal of Mathematica, vol. 58, 1936, pags. 345-363.

[Dijkstra65] E.W. Dijkstra; *Solution of a problem in concurrent programming control*; Communications of the ACM, vol. 8, n. 9, Sept. 1965.

[Echenique07] A. L. Cavalieri, N. Müller, Th. Uphues, V. S. Yakovlev, A. Baltuka, B. Horvath, B. Schmidt, L. Blümel, R. Holzwarth, S. Hendel, M. Drescher, U. Kleineberg, P. M. Echenique, R. Kienberger, F. Krausz, U. Heinzmann; *Attosecond spectroscopy in condensed matter*; Nature, vol. 449, pags. 1029-1032, 25 Oct. 2007.

[Gardner70] M. Gardner; *Mathematical games. The fantastic combinations of John Conway's new solitaire game "life"*; Scientific American, vol. 223, Oct. 1970, pags. 120-123.

[Garey79] M.R. Garey, D.S. Johnson's; *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W. H. Freeman, New York, NY, 1979.

[Goldstine96] H.H. Goldstine, A. Goldstine; *The Electronic Numerical Integrator and Computer (ENIAC)*; IEEE Annals of the History of Computing, vol. 18, n. 1, 1996.

[Gödel31] K.F. Gödel; *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme* (Sobre las proposiciones formalmente irresolubles de los Principia Mathematica y los sistemas afines); Monatshefte für Mathematik und Physik, vol. 38, 1931, pags. 173-198.

[Greenlaw95] R. Greenlaw, H.J. Hoover, W.L. Ruzzo; *Limits to Parallel Computation: P-Completeness Theory*; Oxford University Press, New York, 1995.

[Hopcroft02] J.E. Hopcroft, R. Motwani, J.D. Ullman; *Introducción a la Teoría de Autómatas, Lenguajes y Computación*; Addison-Wesley, Madrid, 2002. Gran parte de la teoría de la computabilidad y de la complejidad que se incluye procede de este texto.

[Karp72] R.M. Karp; *Reducibility Among Combinatorial Problems*; en *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher (ed.), Plenum Press, New York, 1972, pags. 85-103.

[Kleene36] S.C. Kleene; *General recursive functions of natural numbers*; Mathematische Annalen, vol. 112, 1936, pags. 727-742.

[Lamport82] L. Lamport, R. Shostak, M. Pease; *The Byzantine Generals Problem*; ACM Transactions on Programming Languages and Systems, vol. 4, n. 3, Jul. 1982, pags. 382-401.

[LeLann77] G. LeLann; *Distributed systems: Towards a formal approach*; Proc. Information Processing '77, B. Gilchrist (ed.), North-Holland, 1977, pags. 155-160.

[Lynch96] N.A. Lynch; *Distributed Algorithms*; Morgan Kaufman Publishers Inc., 1996.

[Mollick06] E. Mollick; *Establishing Moores Law*; IEEE Annals of the History of Computing, Jul-Sep. 2006.

[Moore65] G.E. Moore; *Cramming more components onto integrated circuits*; Electronics, vol. 38, n. 8, Abr. 1965.

[Post36] E.L. Post; *Finite combinatory processes-formulation 1*; The Journal of Symbolic Logic, vol. 1, n. 3, Sep. 1936, pags. 103-105.

[Sarkar00] P. Sarkar; *A brief history of cellular automata*; ACM Computing Surveys (CSUR), vol. 32, n. 1, Mar. 2000, pags. 80 - 107.

[Skiena98] S.S. Skiena; *The Algorithm Design Manual*; Springer-Verlag, New York, 1998.

[Tel94] G. Tel; *Introduction to Distributed Algorithms*; Cambridge University Press, 1994. Gran parte de la teoría de sistemas distribuidos que se incluye procede de este texto.

[Turing36] A.M. Turing; *On computable numbers, with an application to the Entscheidungproblem* (Sobre los números computables, con una aplicación al problema de decisión); Proceedings of the London Mathematical Society, Series 2, n. 42, Nov. 1936, pags. 230-265.

[Wikipedia] Wikipedia, la enciclopedia libre; http://es.wikipedia.org/. De aquí se han extraído gran parte de los datos biográficos que se incluyen.

[Wolfram84] S. Wolfram; *Programación en ciencias y en matemáticas*; Investigación y Ciencia, n. 98, Nov. 1984, pags. 124-138.