Inteligencia artificial

Ramon López de Mántaras Badia
y Pedro Meseguer González

¿Es posible construir máquinas inteligentes? ¿Es el cerebro una máquina? Estas dos preguntas han sido la obsesión de grandes pensadores durante siglos. El objetivo último de la inteligencia artificial —lograr que una máquina tenga una inteligencia de tipo general similar a la humana— es de los más ambiciosos que se ha planteado la ciencia, comparable a otras grandes metas científicas como explicar el origen de la vida, comprender el universo o desentrañar la estructura de la materia. En este libro se describe el fascinante viaje de la inteligencia artificial desde sus orígenes, a mediados de los años cincuenta, hasta hoy. Se presentan los logros conseguidos y los escollos encontrados a lo largo del camino recorrido, así como también las dificultades y limitaciones a las que debemos enfrentarnos para lograr desarrollar futuras inteligencias artificiales similares a la humana.

Ramon López de Mántaras Badia & Pedro Meseguer González

Inteligencia artificial

¿Qué sabemos de? - 87

ePub r1.1
oronet 02.05.2021

Título original: *Inteligencia artificial*
Ramon López de Mántaras Badia & Pedro Meseguer González, 2017

Editor digital: oronet
ePub base r2.1

Índice de contenido

Introducción

¿Es posible construir máquinas inteligentes? ¿Es el cerebro una máquina? Estas son dos preguntas que han obsesionado a grandes pensadores durante siglos. El desarrollo de la inteligencia artificial (IA) ha acercado ambas cuestiones e incluso para muchos investigadores las ha unificado en el sentido de que se están usando los mismos conceptos, técnicas y experimentos en los intentos de diseñar máquinas inteligentes y en investigar la naturaleza de la mente. Actualmente, todavía sabemos poco acerca del cerebro; sin embargo, estamos siguiendo un camino que pasa por considerarlo un sistema computacional y hemos empezado a explorar el espacio de posibles modelos computacionales que permitan emular su funcionamiento.

El objetivo último de la IA —lograr que una máquina tenga una inteligencia de tipo *general* similar a la humana— es de los más ambiciosos que se ha planteado la ciencia. Por su dificultad, es comparable a otros grandes objetivos científicos como explicar el origen de la vida, el origen del universo o conocer la estructura de la materia. A lo largo de los últimos siglos, este afán por construir máquinas inteligentes nos ha conducido a inventar modelos o metáforas del cerebro humano. Por ejemplo, en el siglo XVII, Descartes se preguntó si un complejo sistema mecánico compuesto de engranajes, poleas y tubos podría, en principio, emular el pensamiento. Dos siglos después, la metáfora fueron los sistemas telefónicos, ya que parecía que sus conexiones se podían asimilar a una red de neuronas. Actualmente, el modelo dominante es el computacional, basado en el ordenador digital y, por consiguiente, es el modelo que se contempla en este libro.

Unas consideraciones sobre el concepto de ‘inteligencia’

La inteligencia no es una característica exclusiva de los humanos. En la naturaleza existen muchos animales que exhiben un comportamiento inteligente, en el sentido de que planifican, son capaces de prever las consecuencias de sus acciones y emplean útiles o herramientas para conseguir sus propósitos. Algunos animales tienen también capacidades, aunque muy limitadas, para procesar el lenguaje. En definitiva, hay una larga lista de observaciones científicas que avalan manifestaciones de inteligencia en chimpancés, delfines, elefantes y otros animales. Por este motivo es más correcto hablar de *inteligencias* que de *inteligencia*, y no sería absurdo pensar que la IA pueda

llegar a constituir un nuevo tipo de inteligencia, aunque, como veremos, distinta de la de animales y humanos.

Centrándonos en la inteligencia humana, que es el referente principal en IA, en este capítulo introducimos brevemente los modelos computacionales más importantes, empezando por la distinción entre *IA débil* e *IA fuerte*, dos visiones que se corresponden, respectivamente, con los dos siguientes intentos de definición:

1. La IA es la ciencia e ingeniería que permite diseñar y programar ordenadores de forma que realicen tareas que requieren inteligencia.
2. La IA es la ciencia e ingeniería que permitirá replicar la inteligencia humana mediante máquinas.

La hipótesis del sistema de símbolos físicos y la inteligencia artificial: IA débil frente a IA fuerte

En una ponencia, con motivo de la recepción del prestigioso Premio Turing en 1975, Allen Newell y Herbert Simón formularon la hipótesis del sistema de símbolos físicos (SSF), según la cual "todo sistema de símbolos físicos posee los medios necesarios y suficientes para llevar a cabo acciones inteligentes". Por otra parte, dado que los seres humanos somos capaces de mostrar conductas inteligentes en el sentido general, entonces, de acuerdo con la hipótesis, nosotros también somos sistemas de símbolos físicos. Conviene aclarar a lo que se refieren Newell y Simón: un SSF consiste en un conjunto de entidades denominadas símbolos que, mediante relaciones, pueden combinarse formando estructuras más grandes —como los átomos que forman moléculas— y que pueden ser transformados aplicando un conjunto de procesos. Estos pueden crear nuevos símbolos, crear y modificar relaciones entre símbolos, almacenarlos, comparar si dos símbolos son iguales o distintos, etc. Estos símbolos son físicos en tanto que tienen un sustrato físico-electrónico (en el caso de los ordenadores) o físico-biológico (en el caso de los seres humanos). Efectivamente, en el caso de los ordenadores, los símbolos se realizan mediante circuitos electrónicos digitales, y en el caso de los seres humanos, mediante redes de neuronas. En definitiva, de acuerdo con la hipótesis SSF, la naturaleza del sustrato (circuitos electrónicos o redes de neuronas) carece de importancia siempre y cuando dicho sustrato permita procesar símbolos. No olvidemos que se trata de una hipótesis y por lo tanto no debe ser ni aceptada ni rechazada *a priori*. En cualquier caso, su validez o refutación se deberá verificar, de acuerdo con el método científico, con ensayos experimentales. La IA es precisamente el campo científico dedicado a intentar comprobar esta hipótesis en el contexto de los ordenadores digitales, es decir, si un ordenador convenientemente programado es capaz o no de tener conducta inteligente de tipo general.

Es importante el matiz de que debería tratarse de inteligencia de tipo *general* y no *especializada*, ya que la inteligencia de los seres humanos es de tipo general. Exhibir inteligencia específica es otra cosa bien distinta. Por ejemplo, los programas que juegan al ajedrez a nivel de Gran Maestro son incapaces de jugar a las damas a pesar de ser un juego mucho más sencillo. Se requiere diseñar y ejecutar un programa distinto e independiente del que le permite jugar al ajedrez para que el mismo ordenador juegue también a las damas. En el caso de los seres humanos no es así, ya que cualquier jugador de ajedrez puede aprovechar sus conocimientos sobre este juego para, en cuestión de segundos, jugar a las damas bien. El diseño y realización de inteligencias artificiales que únicamente muestran comportamiento inteligente en un ámbito muy especializado es lo que se conoce por IA débil en contraposición con la IA fuerte, a la que se referían Newell y Simón. Aunque estrictamente la hipótesis SSF se formuló en 1975, ya estaba implícita en las ideas de los pioneros de la IA, e incluso en las ideas de Alan Turing en sus escritos sobre máquinas inteligentes (Turing, 1950).

Quien introdujo esta distinción entre IA débil e IA fuerte fue el filósofo John Searle en un artículo crítico con la IA, publicado en 1980, que provocó —y sigue provocando— mucha polémica (Searle, 1980). La IA fuerte implicaría que un ordenador convenientemente programado no simula una mente, sino que *es una mente* y por consiguiente debería ser capaz de pensar igual que un ser humano. Searle intenta demostrar que la IA fuerte es imposible. Conviene aclarar que no es lo mismo IA general que IA fuerte. Existe obviamente una conexión, pero solamente en un sentido: toda IA fuerte será necesariamente general, pero puede haber IA generales que no sean fuertes, es decir, que simulen la capacidad de exhibir inteligencia general de la mente pero sin ser mentes.

La IA débil consistiría, según Searle, en construir programas que ayudan al ser humano en sus actividades mentales en lugar de duplicarlas. La capacidad de los ordenadores para realizar tareas específicas mejor que las personas ya se ha demostrado. En ciertos dominios los avances de la IA débil superan en mucho la pericia humana, como, por ejemplo, buscar soluciones a fórmulas lógicas con muchas variables o jugar al ajedrez. También se asocia con la IA débil el hecho de formular y probar hipótesis acerca de aspectos relacionados con la mente (por ejemplo, la capacidad de razonar deductivamente, de aprender inductivamente, etc.) mediante la construcción de programas que llevan a cabo dichas funciones, aunque sea mediante procesos completamente distintos a los que lleva a cabo el cerebro. Todos los avances logrados hasta ahora en el campo de la IA son manifestaciones de la IA débil. A lo largo de este libro hablaremos sobre todo de la IA débil y veremos numerosos ejemplos de dichos avances.

Los principales modelos en inteligencia artificial: simbólico, conexionista, evolutivo y corpóreo

El modelo dominante en IA ha sido el *simbólico*, que tiene sus raíces en la hipótesis SSF. De hecho, sigue siendo muy importante y actualmente se considera el modelo clásico en IA (también denominado GOFAI, de Good Old Fashioned AI). Es un modelo *top-down* que se basa en el razonamiento lógico y la búsqueda heurística como pilares para la resolución de problemas, sin que el sistema inteligente necesite formar parte de un cuerpo ni estar situado en un entorno real. Es decir, la IA simbólica opera con representaciones abstractas del mundo real que se modelan mediante lenguajes de representación basados principalmente en la lógica matemática y sus extensiones. Por este motivo los primeros sistemas inteligentes resolvían principalmente problemas que no requieren interactuar directamente con el entorno como, por ejemplo, demostrar teoremas o jugar al ajedrez (los programas que juegan al ajedrez no necesitan la percepción visual para ver las piezas en el tablero ni actuadores para mover las piezas). Ello no significa que la IA simbólica no pueda ser usada para, por ejemplo, programar el módulo de razonamiento de un robot físico situado en un entorno real, pero, en los primeros años, los pioneros de la IA no disponían de lenguajes de representación del conocimiento ni de programación que permitieran hacerlo de forma eficiente y, por este motivo, los primeros sistemas inteligentes se limitaron a resolver problemas que no requerían interacción directa con el mundo real. Actualmente, la IA simbólica se sigue usando para demostrar teoremas o jugar al ajedrez, pero también para aplicaciones que requieren percibir el entorno y actuar sobre él como, por ejemplo, el aprendizaje y la toma de decisiones en robots autónomos, tal como veremos más adelante.

Simultáneamente con la IA simbólica también empezó a desarrollarse una IA bioinspirada llamada *conexionista*. Los sistemas conexionistas no son incompatibles con la hipótesis SSF, pero, contrariamente a la IA simbólica, se trata de una modelización *bottom-up*, ya que se basan en la hipótesis de que la inteligencia emerge a partir de la actividad distribuida de un gran número de unidades interconectadas que procesan información paralelamente. En la IA conexionista estas unidades son modelos aproximados de la actividad eléctrica de las neuronas biológicas.

Ya en 1943, McCulloch y Pitts propusieron un modelo simplificado de neurona biológica basándose en la idea de que una neurona es esencialmente una unidad lógica. Este modelo es una abstracción matemática con entradas (dendritas) y salidas (axones). El valor de la salida se calcula en función del resultado de una suma ponderada de las entradas, de forma que si dicha suma supera un umbral preestablecido entonces la salida es un 1, en caso contrario es 0. Conectando la salida de cada neurona con las entradas de otras neuronas se forma una red neuronal artificial. Respecto a lo que ya se sabía sobre el reforzamiento de las sinapsis entre neuronas biológicas, se vio que estas redes neuronales artificiales se podían entrenar para aprender funciones que relacionaran las entradas con las salidas mediante el ajuste de los pesos que sirven para ponderar las conexiones entre neuronas. Por este

motivo, se pensó que serían mejores modelos para el aprendizaje, la cognición y la memoria que los modelos basados en la IA simbólica. Sin embargo, los sistemas inteligentes basados en el conexionismo tampoco necesitan formar parte de un cuerpo ni estar situados en un entorno real y, desde este punto de vista, tienen las mismas limitaciones que los sistemas simbólicos. Por otra parte, las neuronas reales poseen complejas arborizaciones dendríticas con propiedades no solo eléctricas, sino también químicas nada triviales. Pueden contener conductancias iónicas que producen efectos no lineales. Pueden recibir decenas de millares de sinapsis variando en posición, polaridad y magnitud. Además, hoy día sabemos que en el cerebro hay unas células llamadas gliales que regulan el funcionamiento de las neuronas, siendo incluso más numerosas que estas. No existe ningún modelo conexionista que incluya a dichas células, por lo que en el mejor de los casos estos modelos son incompletos. En definitiva, toda la enorme complejidad del cerebro queda muy lejos de los modelos actuales y plantea dudas sobre la utilidad de grandes iniciativas, como el proyecto Human Brain de la Unión Europea. Esta inmensa complejidad del cerebro también conduce a pensar que la llamada *singularidad*, es decir, futuras superinteligencias artificiales que basadas en réplicas artificiales del cerebro superaran con mucho la inteligencia humana en un plazo de unos 20 años, es una predicción con muy poco fundamento.

Otra modelización bioinspirada, también compatible con la hipótesis SSF y no corpórea, es la *computación evolutiva*. Los éxitos de la biología evolucionando organismos complejos hizo que a primeros de los años sesenta algunos investigadores se plantearan la posibilidad de imitar la evolución con el fin de que los programas de ordenador, mediante un proceso evolutivo, mejorasen automáticamente las soluciones a los problemas para los que habían sido programados. La idea es que estos programas, gracias a operadores de mutación y cruce de *cromosomas* que los modelan, crean nuevas generaciones de programas modificados, cuyas soluciones son mejores que las de las generaciones anteriores. Dado que podemos considerar que el objetivo de la IA es la búsqueda de programas capaces de producir conductas inteligentes, se pensó que se podría usar la programación evolutiva para encontrarlos dentro del espacio de programas posibles. La realidad es mucho más compleja y esta aproximación tiene muchas limitaciones, aunque ha producido excelentes resultados en problemas de optimización y también en la invención de nuevos dispositivos (por ejemplo, un nuevo tipo de antena).

Una de las críticas más fuertes a estos modelos no corpóreos se basa en que un agente inteligente necesita un cuerpo para poder tener experiencias directas con su entorno (diríamos que el agente está *situado* en su entorno) en lugar de que un programador proporcione descripciones abstractas de dicho entorno codificadas mediante un lenguaje de representación del conocimiento. Sin un cuerpo, estas representaciones abstractas no tienen contenido semántico para la máquina. Sin embargo, mediante la interacción directa con el entorno, el agente puede relacionar

las señales que percibe mediante sus sensores con representaciones simbólicas generadas a partir de lo percibido. Algunos expertos en IA incluso llegaron a afirmar que no era ni siquiera necesario generar dichas representaciones internas, es decir, que no es necesario que un agente deba tener una representación interna del mundo que le rodea, ya que el propio mundo es el mejor modelo posible de sí mismo, y que la mayor parte de las conductas inteligentes no requieren razonamiento, sino que emergen a partir de la interacción entre el agente y su entorno. Esta idea generó mucha polémica y uno de sus principales valedores, Rodney Brooks, unos años más tarde, admitió que hay muchas situaciones en las que una representación interna del mundo es necesaria para que el agente tome decisiones racionales.

La aproximación corpórea con representación interna ha ido ganando terreno en la IA y actualmente muchos la consideramos imprescindible para avanzar hacia inteligencias de tipo general. De hecho, basamos una gran parte de nuestra inteligencia en nuestra capacidad sensorial y motora. En otras palabras, *el cuerpo da forma a la inteligencia* (*the body shapes the way we think*) y por lo tanto sin cuerpo no puede haber inteligencia de tipo general. Esto es así porque el *hardware* del cuerpo, en particular los mecanismos del sistema sensor y del sistema motor, determina el tipo de interacciones que un agente puede realizar. A su vez, estas interacciones conforman las habilidades cognitivas de los agentes, dando lugar a lo que se conoce como *cognición situada*. Es decir, se sitúa la máquina en entornos reales, como ocurre con los seres humanos, con el fin de que tengan experiencias interactivas que, eventualmente, les permitan llevar a cabo algo similar a lo que propone la teoría del desarrollo cognitivo de Piaget, según la cual un ser humano sigue un proceso de maduración mental por etapas y quizá los distintos pasos de este proceso podrían servir de guía para diseñar máquinas inteligentes. Estas ideas han dado lugar a una nueva subárea de la IA llamada *robótica del desarrollo (developmental robotics)*. A lo largo del libro exploraremos con más detalle estos modelos.

CAPÍTULO 1
Orígenes de la inteligencia artificial

Los orígenes de la IA no pueden entenderse sin hablar de Alan Turing. Fue un científico clarividente cuya relación con la IA no se limita al famoso test que lleva su nombre, sino que anticipó futuros desarrollos de la IA y, lo que es más importante, intuyó la importancia que jugaría el aprendizaje automático en el desarrollo de la IA al afirmar que, en lugar de intentar emular mediante una máquina la mente de un adulto, quizá sería más factible intentar emular la mente de un niño y luego someter a la máquina a un proceso de aprendizaje que diera lugar a un desarrollo cognitivo de dicha mente hasta alcanzar el equivalente de una mente adulta; es decir, lo que actualmente propone la robótica de desarrollo. A continuación relatamos el nacimiento oficial de la IA en 1956 en un encuentro científico en el Dartmouth College, aunque, como veremos, un año antes ya había tenido lugar una sesión dedicada al tema del aprendizaje automático en un congreso celebrado en Los Ángeles. Después describimos los primeros programas capaces de demostrar teoremas y resolver una variedad de problemas relativamente sencillos. A continuación hablaremos de los primeros programas capaces de aprender, los primeros intentos de procesar el lenguaje natural y los primeros lenguajes de programación.

Turing

Alan Mathison Turing[1] (1912-1954) fue un genial matemático británico que hizo muchas contribuciones científicas durante su corta vida, pues se suicidó (previsiblemente) pocos días antes de cumplir 42 años. A Turing se le considera el padre de la informática por su famosa máquina de Turing, un mecanismo teórico para modelar cualquier operación de computación. Este elemento ha sido tomado como la base teórica de los ordenadores y como tal se enseña en todas las universidades del mundo. Turing describió dicha máquina en un artículo científico en 1936, cuando había terminado sus estudios superiores de Matemáticas en Cambridge (Reino Unido), pero aún no había comenzado sus estudios de doctorado en Princeton (Estados Unidos) bajo la dirección de Alonzo Church.

Turing trabajó para la inteligencia británica durante la Segunda Guerra Mundial, rompiendo el código secreto que utilizaba el ejército alemán para encriptar sus

comunicaciones mediante la famosa máquina Enigma. Se cree que sus contribuciones fueron decisivas para decidir el curso de la guerra.

Con respecto a la IA, Turing fue un precursor y un visionario. Fue la primera persona en diseñar un programa de ordenador para jugar al ajedrez, a finales de la década de 1940 (y buena parte de las ideas que usaba en su programa se siguen utilizando en los programas de ajedrez por ordenador). En el artículo “Computing machinery and intelligence”, publicado en 1950 en la revista *Mind*. Turing defendía la idea de que los ordenadores podían tener comportamientos inteligentes; y proponía su famoso test de Turing para determinar si un computador exhibía inteligencia, mediante lo que llamaba “el juego de imitación”: un evaluador humano interaccionaba mediante un teletipo con un *ente inteligente* (ordenador u otro humano) que estaba en otra habitación. Al cabo de un tiempo limitado, si el evaluador era incapaz de diferenciar un ordenador de otro humano, Turing consideraba que el ordenador exhibía un comportamiento inteligente. Actualmente, este test se considera demasiado superficial y se han propuesto diversas versiones complementarias del mismo.

Turing se alineaba con lo que después se ha dado en llamar la IA fuerte (aunque en aquellos años iniciales, cuando nadie se atrevía a adscribir la posibilidad de comportamiento inteligente a un ordenador, este detalle no era muy relevante). Turing anticipó muchos de los temas que la IA ha estudiado posteriormente. Además de los juegos por ordenador como el ajedrez, él propuso el aprendizaje como un mecanismo básico para máquinas inteligentes, consideró la importancia de interfaces humanizadas, concibió la idea de creatividad computacional y propuso las ideas que hoy están en la base de la robótica del desarrollo.

El nacimiento de la inteligencia artificial

Poco tiempo después de que Turing publicara sus primeras ideas sobre máquinas inteligentes, al otro lado del Atlántico varios investigadores tenían inquietudes similares. En 1955, hubo una sesión dedicada a *máquinas que aprenden (learning machines)* en la Western Joint Computer Conference, en Los Ángeles. En dicha sesión se presentaron cuatro trabajos, tres sobre reconocimiento de patrones y el cuarto acerca de máquinas para jugar al ajedrez. En uno de los trabajos sobre reconocimiento de patrones, Clark y Farley describían distintas formas de ajustar los pesos de las conexiones entre las neuronas de una red neuronal artificial para que pudiera aprender a reconocer patrones. Este trabajo se inspiraba en el modelo de reforzamiento de las sinapsis entre neuronas biológicas propuesto por el neuropsicólogo Donald Hebb. En otro trabajo, también inspirado en el funcionamiento del sistema nervioso humano, Gerald Dinneen presentó una aproximación al reconocimiento de patrones consistente en detectar los bordes de los objetos, asignando valores numéricos a los distintos niveles de gris, aunque Dinneen

se limitó a la fase de detección de los bordes sin llegar realmente a la fase de reconocer los objetos. El tercer trabajo sobre reconocimiento de patrones lo presentó Oliver Selfridge y, en cierto modo, fue complementario al de Dinneen, ya que clasificaba imágenes identificando características salientes contenidas en ellas —por ejemplo, distinguir entre triángulos y rectángulos detectando las esquinas y contando cuántas hay—. Es interesante señalar que las ideas de Dinneen y Selfridge siguen siendo relevantes para el procesamiento de imágenes. Otro trabajo, el de Allen Newell, esbozaba la posibilidad de programar ordenadores de forma que jugaran al ajedrez al estilo humano, es decir, incorporando conceptos como la descomposición de un objetivo en subobjetivos más simples, criterios para detener el proceso de búsqueda del siguiente movimiento a efectuar y el uso de funciones heurísticas de evaluación de los movimientos para seleccionar un movimiento *suficientemente bueno* de acuerdo con el concepto de *satisficing solution* propuesto por Herbert Simón (1956). Este concepto es una estrategia heurística de toma de decisiones: cuando la decisión óptima es prácticamente imposible de determinar debido a que existe un número demasiado elevado de posibilidades a tener en cuenta, entonces se toma una decisión suficientemente satisfactoria aunque no sea óptima. En su trabajo, Newell daba a entender que estas técnicas, basadas en procesamiento simbólico, podrían servir no solamente para jugar al ajedrez, sino también para tratar otros problemas de alta complejidad. La idea de la *satisficing solution* está muy presente en la *búsqueda heurística*, una técnica fundamental en IA de la que hablaremos en el próximo capítulo.

Un año después, John McCarthy organizó un encuentro en el Dartmouth College en New Hampshire con la idea de que cualquier aspecto del aprendizaje o cualquier otro rasgo de la inteligencia podía, en principio, ser descrito con un nivel de detalle suficiente para ser simulado en una máquina. McCarthy convenció a Claude Shannon, de los laboratorios Bell, y a Marvin Minsky, entonces en Harvard, para redactar una propuesta a partir de esta idea. Se tituló “Summer Research Project in Artificial Intelligence” y solicitaron financiación a la Fundación Rockefeller. Fue financiada y el encuentro, que duró seis semanas, tuvo lugar en verano de 1956. En la propuesta se afirmaba que, en solamente unos pocos meses, un grupo de científicos cuidadosamente seleccionados podrían conseguir avances significativos en aspectos tales como la comprensión del lenguaje, la abstracción de conceptos mediante aprendizaje o la resolución de problemas que hasta entonces solo habían sido resueltos por seres humanos. Era una predicción exageradamente optimista, ya que fueron necesarias varias décadas para poder hablar efectivamente de progresos significativos en dichos temas. Una de las constantes de los pioneros de la IA era precisamente su excesivo optimismo, consecuencia de subestimar la complejidad del problema de modelar los procesos cognitivos. De hecho, en 2006, durante la celebración, también en Dartmouth, del 50 aniversario de la famosa reunión,

McCarthy, Minsky, Selfridge y Solomonoff reconocieron que la IA era un objetivo mucho más difícil de lo que nunca llegaron a imaginar.

Además de los tres proponentes, estuvieron también en Dartmouth los siguientes investigadores: Nathaniel Rochester, Arthur Samuel y Alex Bernstein, de IBM; Oliver Selfridge y Ray Solomonoff, del MIT; Allen Newell, de Rand Corporation; y Herbert Simón, de la Universidad de Carnegie Mellon. Rochester estaba interesado en la aproximación conexionista a la IA; Samuel había diseñado un programa que jugaba a los *checkers* —un juego muy similar a las damas españolas—, de forma que jugando contra una copia de sí mismo era capaz de aprender a mejorar su juego mediante una técnica de actualización de una función matemática que evaluaba las jugadas. Bernstein también estaba interesado en los juegos y había trabajado en un programa para jugar al ajedrez.

Selfridge, que, como hemos dicho, también participó en la sesión sobre *learning machines* un año antes en Los Ángeles, seguía interesado en el problema del reconocimiento de patrones, y Ray Solomonoff trabajaba en una teoría general de la inferencia y sus posibles implicaciones para modelizar inteligencia artificial general. Newell (que también había participado en la sesión de Los Ángeles) y Simón llevaron a Dartmouth algo más tangible: un programa capaz de demostrar teoremas de lógica proposicional del que hablaremos en el próximo apartado. Obviamente, los tres proponentes también expusieron sus ideas e intereses. McCarthy, además de proponer con éxito que el nuevo campo de estudio se llamara inteligencia artificial, estaba interesado en diseñar un lenguaje artificial con el que programar aspectos tales como la auto-rreferencia, lo que dio lugar años después al lenguaje de programación LISP[2]. Shannon estaba interesado en la aplicabilidad de su teoría de la información para modelizar el cerebro, pero después de la reunión de Dartmouth dejó de interesarse por la IA. Por último, Minsky planteó una máquina capaz de construir un modelo abstracto de su entorno, de forma que a la hora de resolver cualquier problema primero intentara solucionarlo usando dicho modelo abstracto interno y, si eso no resultaba, intentara planificar experimentos externos para resolverlo.

Los primeros programas

Como hemos mencionado, Allen Newell y Herbert Simón fueron a la reunión de Dartmouth no solo para discutir ideas más o menos abstractas, sino con un programa ya operativo (Feigenbaum y Feldman, 1963) llamado Logic Theorist (LT), capaz de demostrar algunos teoremas sobre lógica proposicional contenidos en el libro *Principia Mathematica* de Bertrand Russell —concretamente la mayoría de los teoremas del capítulo 2—. Partiendo de cinco axiomas y tres reglas de inferencia, entre ellas la regla del *modus ponens*, LT pudo demostrar los primeros teoremas que se basaban únicamente en los axiomas de partida. A continuación, LT usaba los axiomas y, de forma similar a como lo haría un matemático, también los teoremas ya

demostrados para expresar otros nuevos. A pesar de que no lo pudo hacer con teoremas bastante obvios, LT abrió la puerta al desarrollo de demostradores de teoremas. Algunos de estos han mostrado teoremas de mucha complejidad, como veremos en el capítulo 5. LT fue la primera constatación de que el procesamiento simbólico y el uso de *heurísticas*[3] eran fundamentales para la resolución de problemas. Efectivamente, LT representaba los axiomas y los teoremas mediante estructuras simbólicas que eran transformadas, mediante las reglas de inferencia, hasta llegar a una estructura que coincidiera con la del teorema a demostrar de forma que la secuencia de transformaciones aplicadas constituyera la demostración. Dado que existía un número muy elevado de secuencias posibles, el problema era encontrar las que conducían al teorema a demostrar mediante heurísticas que seleccionaban las transformaciones más prometedoras.

Poco tiempo después, en una conferencia celebrada en París en 1959, Herb Gelernter (Feigenbaum y Feldman, 1963) presentó un programa capaz de demostrar teoremas de geometría. La idea principal era una estrategia conocida ahora por "divide y vencerás", consistente en descomponer el objetivo inicial —teorema a demostrar— en uno o más subobjetivos que podían a su vez ser divididos en subsubobjetivos, y así sucesivamente, hasta llegar a cuestiones triviales cuya solución conjunta implicaba inmediatamente la demostración del objetivo inicial. Por ejemplo, para demostrar que dos ángulos son iguales, el programa comprobaba que dichos ángulos correspondían a ángulos de dos triángulos congruentes; por lo tanto, el nuevo subobjetivo era demostrar que los triángulos son congruentes y para ello el programa disponía de varios métodos para trivialmente comprobar congruencias entre triángulos. En demostraciones complejas, este proceso de división en subobjetivos puede dar lugar a un número elevado de posibilidades (representadas mediante un árbol cuyo tronco principal es el objetivo inicial y las ramas los subobjetivos, subsubobjetivos, etc.), por lo que el programa de Gelernter usaba heurísticas para elegir el orden de demostración de los subobjetivos y descartar lo antes posible aquellas alternativas que no podían conducir a la solución del problema, debido a que alguno de los subobjetivos implicados era falso. Esta estrategia de resolución de problemas ha sido, y sigue siendo, muy utilizada en IA, ya que es efectiva para tratar problemas que se caracterizan por lo que se conoce como *explosión combinatoria*[4].

Otro conocido programa de resolución de problemas, también de finales de los años cincuenta y presentado por primera vez en la conferencia de París de 1959, es el *solucionador general de problemas* (General Problem Solver, GPS) (Feigenbaum y Feldman, 1963). El adjetivo "general" en el nombre del programa es coherente con las ideas sobre la IA fuerte de los pioneros de la IA. De hecho, GPS no era solamente un programa de ordenador, sino una teoría de cómo los humanos resolvemos problemas. Para Newell y Simon, igual que para la mayoría de los pioneros de la IA, el objetivo principal no era construir máquinas inteligentes, sino explicar el comportamiento inteligente humano. De hecho, la construcción de máquinas

inteligentes, más que un objetivo en sí, era el medio para explicar la inteligencia humana; por este motivo, el GPS era presentado como una teoría sobre la manera en que los humanos resolvemos problemas.

Igual que el LT, el GPS procesaba estructuras simbólicas, ya que, de acuerdo con la hipótesis SSF explicada en la introducción, los seres humanos somos procesadores de símbolos. El GPS incorporaba además una estrategia de dividir problemas en subproblemas, llamada *means-ends analysis*, similar a la utilizada por Gelernter. Esta estrategia consistía en calcular la diferencia entre la representación simbólica del problema a resolver y la representación simbólica del estado actual del problema. Identificada la diferencia, el programa consultaba una *tabla de diferencias* donde a cada una le correspondía un operador que, aplicado sobre la estructura simbólica que representaba el estado actual, obtenía otra estructura simbólica más próxima —menos diferente— a la del problema a resolver. Para poder aplicar un operador de transformación, el programa tenía que verificar que se satisficieran unas condiciones de aplicabilidad, lo cual daba lugar a tener que o solver un subproblema, de ahí la similitud con la estrategia usada por Gelernter. En estas transformaciones de estructuras simbólicas hay también similitudes con el LT. Además, igual que el LT y el programa de Gelernter, en problemas complejos, el GPS también tiene el problema de la explosión combinatoria, por lo que usaba heurísticas relativas al orden de aplicación de los operadores con el fin de podar el árbol de posibilidades. La solución del GPS para un problema dado consistía en la secuencia de operadores de trasformación que conducían del estado inicial al estado final.

Veamos un sencillo ejemplo del funcionamiento del GPS que consiste en demostrar que del estado inicial $A \wedge (\neg B \rightarrow C)$ se puede deducir el objetivo $(B \vee C) \wedge A$. El GPS compara sus estructuras simbólicas y la primera diferencia que tiene en cuenta es que la A está en lados diferentes del conectivo conjuntivo $\wedge$. A continuación, accede a la tabla de diferencias $\vee$ encuentra el operador $(X \wedge Y) \equiv (Y \wedge X)$. GPS lo aplica y obtiene $(\neg B \rightarrow C) \wedge A$, con lo que ya ha reducido en algo la diferencia entre la estructura simbólica del problema y la del estado actual, ya que ahora A está a la derecha en ambas expresiones. A continuación, detecta que existe una diferencia entre los conectivos $\rightarrow$ y $\vee$, por lo que vuelve a consultar la tabla de diferencias y encuentra el operador $(\neg X \rightarrow Y) \equiv (X \vee Y)$. Lo aplica y obtiene $(B \vee C) \wedge A$ con lo cual queda resuelto el problema, pues ya no hay diferencia entre las estructuras simbólicas del estado actual y del objetivo.

Newell y Simon afirmaron que su GPS podía resolver una amplia variedad de problemas si se proporcionaba en cada caso la tabla de diferencias con los operadores pertinentes para reducirlas. A pesar de que el GPS se usó para resolver distintos problemas, su aplicabilidad se limitaba a casos muy sencillos y por consiguiente poco representativos de la capacidad humana para resolverlos. Sin embargo, introdujo ideas importantes en resolución heurística de problemas que posteriormente se mejoraron y siguen estando en el origen de muchas ideas de la IA actual.

Es importante añadir que el GPS resuelve los subproblemas mediante recursividad. Es decir, que se llama a sí mismo cada vez que tiene que resolver un subproblema. En informática, y en particular en IA, actualmente es muy común que un programa se llame a sí mismo con la condición de que esta nueva versión de sí mismo resuelva un problema más sencillo que el inicial, evitando así entrar en un bucle infinito. De hecho, el concepto de recursividad es fundamental en los lenguajes y programas de IA.

Los primeros programas capaces de aprender

Una clase de problemas estudiados desde los primeros años de la IA son los de juegos de tablero y en particular el ajedrez. En este capítulo, tanto en la sesión sobre *learning machines* como en la Summer Session de Dartmout, varios participantes estaban trabajando en programas de ajedrez. Incluso Turing, ya en 1948, escribió un programa para jugar al ajedrez, cuya ejecución tuvo que simular manualmente. El ajedrez ha jugado un papel tan importante en el desarrollo de la IA que algunos se han referido a este juego como la "*Drosophila* de la IA". Es una metáfora adecuada para las investigaciones sobre búsqueda heurística, ya que muchos de los resultados más importantes se han logrado gracias al ajedrez. En el capítulo 5 hablaremos de nuevo de este juego.

En 1959, Arthur Samuel desarrolló el primer programa capaz de jugar a un juego de tablero a un nivel avanzado. Se trataba del juego de *checkers*, muy similar a las damas. La gran contribución de Samuel fue que mejoraba su nivel de juego ajustando los pesos de las variables de una función heurística de evaluación de los movimientos, que tenía en cuenta el valor de cada pieza y la posición de las piezas en el tablero, en particular la movilidad de las piezas o el control del centro del tablero. El programa representaba todos los posibles movimientos (así como todas las posibles respuestas del contrincante y sus respuestas a las respuestas, y así sucesivamente) mediante un árbol de búsqueda y calculaba, para cada posible movimiento, su valor heurístico y seleccionaba el movimiento de más valor. Dado que hay del orden de 5×10^{20} posiciones posibles, era completamente imposible evaluar todo el árbol de búsqueda. El programa tenía que limitar la profundidad del árbol[5], por lo que cada movimiento seleccionado podía no ser óptimo. Mediante aprendizaje, la calidad de la función heurística mejoraba aunque nunca llegaba a ser perfecta. En 1962, el programa de Samuel llegó a vencer a algunos maestros de *checkers* y tuvo una importante repercusión en los medios de comunicación.

Otra clase de dificultades de aquella época, que motivaron los primeros programas capaces de aprender, eran los de reconocimiento y clasificación de patrones. Ya hemos aludido en este capítulo al trabajo de Clark y Farley sobre el reconocimiento de patrones mediante una red neuronal artificial y también a los trabajos de Dinneen y Selfridge que se inspiraron en redes neuronales. Muchos

participantes en Dartmouth estaban interesados en modelos de aprendizaje basados en redes neuronales. En particular, Marvin Minsky, cuya tesis doctoral versaba sobre redes neuronales analógicas como posible modelo teórico del cerebro. Minsky diseñó y construyó una red neuronal analógica de 40 neuronas usando 3.000 tubos de vacío y otros componentes electrónicos, y la aplicó para simular cómo un ratón encuentra la salida en un laberinto. Los modelos de aprendizaje basados en redes neuronales era la aproximación dominante en los primeros programas capaces de aprender. Los trabajos posteriores de Oliver Selfridge, en reconocimiento de patrones con su sistema Pandemonium, también estaban basados en redes neuronales, aunque los procesos computacionales de Pandemonium, denominados *demons*, pueden adoptar tanto la forma de neuronas artificiales como de elementos de procesamiento simbólico para modelizar funciones cognitivas de más alto nivel. Se trataba pues de una arquitectura multinivel donde los procesos se ejecutarían en paralelo. Las ideas expuestas por Selfridge en Pandemonium eran avanzadas para su tiempo y no tuvieron el reconocimiento que merecían. Más tarde, el estudio de arquitecturas multinivel integrando procesamiento simbólico y subsimbólico fue objeto de atención en IA.

De mediados de los años cincuenta a mediados de los sesenta hubo mucha actividad en sistemas capaces de aprender a reconocer patrones mediante redes neuronales. Reconocían patrones bidimensionales como, por ejemplo, caracteres alfa-numéricos o fotografías, que se podían representar mediante matrices de números (posteriormente llamados píxeles). El reconocimiento óptico de caracteres alfanuméricos atrajo la atención del sector bancario debido a su aplicación a la lectura automática de los números de cuenta impresos en los cheques. Frank Rosenblatt desarrolló un tipo de red neuronal, llamada *perceptron*, basada en el modelo de McCulloch y Pitts. De nuevo, la aplicación en la que estaba interesado Rosenblatt era el reconocimiento de caracteres alfanuméricos. Igual que Minsky, Rosenblatt construyó sus redes neuronales en *hardware*, ya que el *software* de aquella época era demasiado lento para las aplicaciones en las que estaba interesado Rosenblatt, que incluían además el reconocimiento automático del habla. Otra contribución pionera al aprendizaje mediante modelos neuronales, también realizada en *hardware*, fueron los dispositivos Adaline de Bernard Widrow, que se aplicaron para desarrollar controladores adaptativos en el campo de la automática.

En los años sesenta, en el Stanford Research Institute, Duda y Hart desarrollaron un sistema basado en redes neuronales capaz de aprender a reconocer instrucciones manuscritas del lenguaje de programación Fortran. En aquellos tiempos, los programadores escribían a mano el programa en tarjetas perforadas que el ordenador leía ópticamente. El sistema cometía solamente un 2% de errores, por lo que tenía una tasa de acierto muy alta para la época.

Hemos visto que los primeros programas capaces de aprender eran de tipo subsimbólico y se basaban todos en la técnica de ajustar los valores de coeficientes

numéricos, ya fuera en redes neuronales artificiales o en funciones heurísticas. Más adelante veremos que hubo un giro radical hacia sistemas simbólicos en la investigación en aprendizaje artificial.

Los primeros intentos de procesar el lenguaje natural

En el apartado anterior hemos hablado del reconocimiento de caracteres alfanuméricos. El paso siguiente es entender el significado de los términos y frases que se forman a partir de dichos caracteres. Este es el objetivo de lo que en IA se conoce por "procesamiento del lenguaje natural", en contraposición de los lenguajes artificiales de programación. Los primeros intentos de procesar el lenguaje natural estuvieron motivados por la necesidad, en plena guerra fría, de llevar a cabo traducciones automáticas entre el ruso y el inglés. La primera conferencia sobre traducción *mecánica* tuvo lugar en 1952 en MIT. Sus organizadores eran muy optimistas acerca de la factibilidad de llevar a cabo traducciones completamente automatizadas basadas con diccionarios en formato electrónico y análisis sintáctico. Se obtuvieron buenos resultados con algunos programas traduciendo del ruso al inglés con un vocabulario limitado y usando reglas gramaticales sencillas. Sin embargo, no tardaron en darse cuenta de que la traducción de alta calidad, sin restricciones limitativas sobre vocabulario y gramática, era mucho más compleja. Algunos pioneros llegaron incluso a afirmar que la traducción automática no era posible ni a corto ni a largo plazo. El argumento principal era que las máquinas no poseen los conocimientos de sentido común[6] necesarios para comprender el significado del lenguaje.

Bar-Hillel ejemplificó esta imposibilidad con la frase en inglés: "*Little John was looking for his toy box. Finally he found it. The box was in the pen. John was very happy*". La dificultad para traducir este texto a otro idioma radica en la frase "*the box was in the pen*", ya que *pen* en inglés significa *pluma* (instrumento de escritura) y también *parque infantil* (pequeño recinto donde juegan los niños). Incluso proporcionado al programa ambas definiciones no es posible que pueda determinar cuál de los dos significados es el correcto, pues para ello tendría que tener conocimientos de sentido común sobre tamaños relativos de objetos en general y en este caso concreto de los tamaños de *caja*, *pluma* y *parque infantil*. Bar-Hillel tenía mucha razón: el problema de dotar de conocimientos de sentido común a las máquinas sigue siendo la principal dificultad en IA. A mediados de los años sesenta, los principales organismos financiadores de la actividad investigadora en Estados Unidos dejaron de apoyar los proyectos de traducción automática y los orientaron hacia el desarrollo de herramientas de ayuda al traductor humano. A pesar de ello, la actividad investigadora en procesamiento del lenguaje natural continuó y sigue siendo un área muy importante en IA, como veremos más adelante.

Además de la traducción automática, en los años sesenta también se iniciaron las investigaciones en sistemas *pregunta-respuesta*. Uno de los primeros programas de este tipo fue Baseball (Feigenbaum y Feldman, 1963), capaz de responder a preguntas gramaticalmente sencillas sobre partidos de balonmano referidos a un determinado año; por ejemplo, dónde jugó un equipo concreto en una fecha dada o cuál fue el resultado de un partido concreto. Básicamente se limitaba a preguntas sobre lugares, fechas, nombres de equipos y resultados. Otro programa similar, SAD SAM (Feigenbaum y Feldman, 1963), admitía preguntas sobre relaciones de parentesco entre personas. Para ello, previamente representaba los nombres de las personas y sus relaciones familiares mediante un árbol y a continuación podía responder a preguntas del estilo: "¿Quién es el padre de Pedro?". Este sistema aceptaba una variedad de preguntas en inglés básico y llegó a manejar un vocabulario de cerca de 1.000 palabras.

En su tesis doctoral de 1964, Daniel Bobrow, estudiante de Marvin Minsky, desarrolló el programa Student, capaz de resolver problemas de álgebra enunciados en inglés sencillo, del estilo: "La distancia entre Barcelona y Madrid es de 600 km. Si la velocidad media de un coche es de 100 por hora, ¿cuánto tiempo tardará en ir de Barcelona a Madrid?". Student analizaba el enunciado y lo transformaba en ecuaciones que a continuación resolvía. También en 1974, Bertram Raphael, otro estudiante de Marvin Minsky, desarrolló en su tesis doctoral otro sistema de pregunta-respuesta llamado Semantic Information Retrieval (SIR). El objetivo de Raphael era poder introducir conocimientos en el sistema y representarlos mediante un lenguaje de representación basado en lógica matemática para responder preguntas que requirieran algún tipo de deducción. El modelo que usaba SIR consistía en conectar términos mediante relaciones: por ejemplo, MANO es parte de BRAZO; NIÑO es subconjunto de PERSONA; DEDO es parte de mano; PERSONA tiene 2 MANOS, y asociar una lista de propiedades y valores a cada término. Por ejemplo, al término GATO le asocia, entre otras, las propiedades y valores: PATAS (4) COLOR BLANCO, NEGRO…, SONIDO MAÚLLA. Los conocimientos que Raphael introdujo en SIR corresponden a lo que entendemos por sentido común. Usando un mecanismo de deducción automática, SIR podía deducir que un niño llamado Juan tiene 10 dedos en las manos a partir de los conocimientos siguientes: JUAN es un NIÑO; NIÑO es subconjunto de PERSONA; PERSONA tiene 2 MANOS; DEDO es parte de MANO; MANO tiene 5 DEDOS. Otra característica de SIR es que podía representar excepciones. Por ejemplo, si una persona concreta tiene un solo brazo, esta información de carácter excepcional le permitiría responder que dicha persona tiene 5 dedos en lugar de 10. Estas situaciones excepcionales, que no cumplen las reglas generales, juegan un papel muy importante en IA y en particular en lo que se conoce como "redes semánticas" (véase capítulo 2). De hecho, SIR se puede considerar un precursor de las redes semánticas en tanto que formalismo de representación del conocimiento.

Todos estos sistemas funcionaban con vocabularios muy reducidos y en temas muy específicos. Además, el tratamiento semántico es casi inexistente, excepto en el caso de SIR, donde vemos un primer intento de representación semántica relacional. Sin embargo, el caso más claro de sistema de diálogo persona-máquina que no comprende ni una palabra del diálogo que lleva a cabo es el famoso sistema Eliza, desarrollado por Weizenbaum en 1966. El sistema se basa en una técnica sencilla de *reconocimiento de patrones (pattern matching)* que busca palabras clave en las respuestas del usuario, de tal forma que estas palabras clave a continuación se insertan en una frase prefabricada para generar la siguiente pregunta. Cuando la palabra clave no está entre las que tiene predefinidas, cambia bruscamente de tema formulando una pregunta no relacionada con la respuesta del usuario. Por ejemplo, Eliza puede empezar preguntando algo muy típico como: "¿Hola, que tal está usted?" y supongamos que el usuario responde: "Bien, pero paso demasiado tiempo frente a un ordenador"; en este punto Eliza puede detectar la palabra clave *ordenador* y seleccionar al azar alguna frase de la base de datos asociada a *ordenador* como, por ejemplo, "¿Por qué menciona usted los ordenadores?", y así sucesivamente. Los actuales *chatbots* que compiten anualmente por el Premio Loebner, inspirado en el test de Turing, se basan en la misma idea y, sorprendentemente, no son realmente mejores que Eliza, lo cual dice mucho de la inutilidad de esta absurda competición.

Los primeros lenguajes de programación

Recordemos que un SSF consiste en un conjunto de entidades, denominadas *símbolos* que pueden combinarse, mediante relaciones, dando lugar a estructuras más grandes y pueden transformarse aplicando un conjunto de procesos. En coherencia con ello, Newell y Simón vieron la necesidad de disponer de lenguajes de programación capaces de procesar símbolos de manera eficiente y diseñaron una serie de lenguajes de programación que llamaron Information Processing Language (IPL). IPL-I era una especificación de las funcionalidades que debían poseer este tipo de lenguaje. A continuación, IPL-II, IPL-III e IPL-IV fueron ya la serie de lenguajes capaces de llevar a cabo un procesamiento de símbolos según una estructura simple llamada *lista de símbolos*, mediante la cual se pueden construir estructuras más complejas tales como listas de listas de símbolos, listas de listas de listas, y así sucesivamente. Los primeros programas heurísticos LT y GPS se programaron mediante estos lenguajes. Poco después, John McCarthy diseñó el lenguaje LISt Processing (LISP), basado también en una estructura de listas de símbolos, con operadores para extraer símbolos de una lista, añadir símbolos a una lista, verificar si un símbolo pertenece o no a una lista, copiar una lista, etc. Combinando estos operadores se pueden llevar a cabo complejos procesamientos simbólicos. La funcionalidad más importante de LISP es que los programas se representan a su vez mediante listas y, por consiguiente, pueden procesarse dentro de otro programa del cual son subprogramas. De hecho, un

programa puede incluso tener como subprograma una versión de sí mismo, lo que se conoce por *recursividad*. Esta es posiblemente una de las características más importantes en la programación de IA. Prácticamente todos los primeros programas de procesamiento del lenguaje natural mencionados anteriormente se programaron usando LISP. La comunidad de investigadores en IA todavía usa extensiones mejoradas de LISP.

A mediados de los años sesenta, en la Universidad de Edimburgo se desarrolló un lenguaje de programación también para procesar listas llamado POP-2. El uso de este lenguaje quedó limitado al entorno de Reino Unido, ya que LISP se impuso como el lenguaje principal en IA a nivel mundial. En la Universidad de Edimburgo el investigador Robert Kowalski sentó las bases para que Alain Colmerauer y Philippe Roussel, de la Universidad de Aix-Marsella, desarrollaran el lenguaje Prolog (PROgrammation en LOGique) a principios de los años setenta. Un programa en Prolog consiste en una secuencia ordenada de expresiones lógicas relacionales. El modelo teórico de Kowalski se basa en hacer inferencias, mediante refutación, en un subconjunto de la lógica de primer orden que se conoce como cláusulas de Horn (Kowalski, 1980). También incorpora la negación por fallo que supone que un hecho es falso si no se puede deducir. La facilidad para programar sistemas de procesamiento de lenguaje natural, así como para desarrollar sistemas expertos[7], fue el motivo del éxito de Prolog, que llegó a rivalizar con LISP. En Japón, los responsables del proyecto Quinta Generación de Sistemas Computacionales lo adoptaron como lenguaje máquina de una hipotética futura generación de ordenadores inteligentes basados en la capacidad para realizar inferencias lógicas en paralelo y a gran velocidad. El proyecto japonés terminó a mediados de los años noventa sin alcanzar sus ambiciosos objetivos, pero fue el detonante de varias iniciativas como el programa Esprit en la Unión Europea —que está en el origen de los actuales Programas Marco de Investigación—, el programa Alvey en Reino Unido y la creación de la MCC Corporation en Estados Unidos. Por otra parte, las ideas sobre programación lógica siguieron extendiéndose; actualmente existe una numerosa y activa comunidad en informática que trabaja en este paradigma computacional.

Capítulo 2
Los primeros grandes avances

En este capítulo vamos a hablar de los primeros grandes éxitos de la IA, concretamente en los temas de representación del conocimiento, búsqueda heurística, planificación, sistemas expertos, aprendizaje automático, visión por computador, robots móviles y procesamiento del lenguaje escrito y hablado. En todos estos temas veremos que hubo avances muy significativos que permitieron que la IA se estableciera definitivamente como una disciplina científica, a pesar de que, como veremos en el capítulo 4, no tardaron en aparecer las primeras dificultades tanto de naturaleza técnica como filosófica y ética.

Representación del conocimiento

Una persona capaz de hacer inferencias no obvias sobre una situación la calificaríamos de *inteligente*. Quisiéramos replicar esa capacidad en un programa de ordenador, de forma que partir de "Juan es un hombre" pudiera deducir que "Juan es mortal", o que "Juan tiene sangre caliente" o que "Juan es varón". Para realizar ese tipo de deducciones, ese programa debe manejar una serie de conocimientos que nosotros hemos adquirido a través de la educación, en la escuela, etc., de forma que somos capaces de realizar esas inferencias sin esfuerzo. Por ejemplo, para deducir que "Juan es mortal", hacemos uso del conocimiento "todos los hombres son mortales". Para que un programa haga una deducción semejante, ese conocimiento ha de estar representado de alguna forma en su memoria. Algo similar sucede con la sangre de Juan: el programa ha de razonar sobre los tipos de sangre en los seres vivos, basándose en informaciones existentes en su memoria (que junto con el hecho de que "Juan es un mamífero" y que "todos los mamíferos tienen sangre caliente" permite deducir que "Juan tiene sangre caliente"). Para inferir que "Juan es un varón", el programa ha de ser capaz de razonar sobre los nombres que tienen los humanos, que dichos nombres están mayormente separados por sexo, y que el nombre "Juan" corresponde al sexo masculino.

Para representar este conocimiento es útil la *lógica de predicados*, ya que permite expresar conocimiento mediante fórmulas. Por ejemplo:

$$\forall x, \textit{hombre}\,(x) \Rightarrow \textit{mortal}\,(x)$$

en lógica de predicados se puede leer “para todo objeto x, si x es hombre entonces x es mortal”. Dicha lógica, si conoce el predicado:

hombre(Juan)

es capaz de generar el nuevo predicado:

mortal(Juan)

mediante la regla de inferencia del *modus ponens*. Este mecanismo es adecuado para representar reglas universales. Esta propiedad motivó que se considerara la lógica de predicados (o lógica de primer orden) como un serio candidato a lenguaje de representación del conocimiento (Feigenbaum y Feldman, 1963), especialmente tras la formulación de la resolución como método de prueba automatizable. Esta idea implicaba que el problema a resolver se debía traducir a una fórmula Φ que se debía derivar como consecuencia lógica de los axiomas. La lógica presentaba muchas propiedades positivas, pero también tenía algunos inconvenientes. Por un lado, era ineficaz, ya que se podían construir una enorme cantidad de deducciones correctas, pero completamente inútiles para el problema que se quería resolver, lo cual llevó a la cuestión del problema del control para orientar las deducciones. Otro un inconveniente era la semidecidibilidad: si Φ no se derivaba de los axiomas, el procedimiento de prueba podía no terminar. La búsqueda de fragmentos de la lógica que permitan un razonamiento eficiente ha sido un objetivo perseguido desde hace tiempo. Por ejemplo, la lógica de predicados reducida a cláusulas de Horn sin funciones (Datalog) es decidible (existe un algoritmo que termina siempre y responde si una fórmula se deriva de un conjunto de fórmulas de esta lógica). Considerando la resolución como regla de inferencia, permite deducciones generales (Kowalski, 1980). La dificultad para incluir conocimiento procedimental (la lógica es esencialmente declarativa) se veía como desventaja por parte de ciertos investigadores, y también la dificultad para manejar excepciones. Por ejemplo, uno puede pensar que la siguiente regla es razonable para expresar que los pájaros vuelan,

$$\forall x, \textit{pájaro}\ (x) \Rightarrow \textit{vuela}\ (x)$$

pero no se puede aplicar a un pingüino o a una golondrina con un ala rota. Son excepciones que no cumplen la norma. Y expresar excepciones en lógica de predicados es difícil.

Estos escollos y la necesidad de resolver problemas en un tiempo comparable al utilizado por los humanos para las mismas tareas hizo que se exploraran otros formalismos alternativos (Bobrow y Collins, 1975). Volviendo al ejemplo de “Juan es un hombre”, el programa demostraría cierta inteligencia si fuera capaz de mencionar la edad de Juan como el período de tiempo transcurrido entre su nacimiento y el

tiempo presente. De hecho, este concepto es común a todos los seres vivos y se podría utilizar una taxonomía en árbol con una estructura de clases de forma que las clases inferiores heredaran las características de las superiores. Esta es una forma de representación muy usada en IA, con la herencia como potente mecanismo deductivo. De nuevo, aparecen problemas para clasificar excepciones. Considerando una taxonomía de animales, el ornitorrinco es un mamífero con sangre caliente que pone huevos, con membranas entre los dedos y con un pico parecido al del pato. ¿De qué clase debe heredar, de los mamíferos o de las aves? Si hereda de las dos clases, ¿qué características de qué clase tienen prioridad?

Siguiendo con nuestro ejemplo, el programa inteligente podría hacer *suposiciones razonables*, en el sentido de preguntarse por el trabajo de Juan (si Juan no es un niño o un anciano), aunque Juan podría ser muy rico y no trabajar o estar parado. O también podría preguntarse por el cónyuge de Juan, aunque Juan puede que esté soltero. O por las redes sociales que frecuenta, aunque Juan puede que no visite ninguna. Todas estas suposiciones representan *razonamientos por defecto*, inferencias razonables para la mayor parte de los individuos, pero que han de contemplar excepciones. Son formas de razonamiento no monótono, es decir, que el conjunto de deducciones válidas no crece monótonamente con el número de hechos conocidos. Los humanos somos muy hábiles haciendo y deshaciendo inferencias, en el sentido de ser capaces de invalidar deducciones cuando conocemos algún dato nuevo que no las permite. Esa habilidad no está disponible en los ordenadores, que se muestran mucho más rígidos a la hora de alterar las deducciones. Se han desarrollado sistemas que mantienen las inferencias válidas basándose en suposiciones, denominados *assumption-based truth maintenance systems*, pero exigen un conocimiento demasiado exhaustivo, lo que en ocasiones los hace poco prácticos.

Los humanos disponemos de un amplio repertorio de conocimientos sobre nuestro mundo, sobre objetos comunes y sobre sus propiedades físicas básicas. Eso no quiere decir que manejemos mentalmente las ecuaciones que definen su un movimiento o su composición. Por ejemplo, sabemos que prácticamente todos los cuerpos caen (aunque no calculemos su velocidad o el tiempo de caída), que hay sólidos, líquidos y gases, que los materiales sólidos son duros o blandos, y que para ciertas acciones mecánicas (clavar un clavo, hacer una palanca) es mejor un sólido de material duro que uno de material blando. Estos conocimientos vienen de nuestra experiencia en el mundo, se suelen formar en la infancia y son el resultado de cientos y cientos de *experimentos* que los niños y niñas pequeños hacen para conocer y comprender cómo funciona el mundo; nos permiten comportarnos *inteligentemente* en situaciones muy diversas. Por ejemplo, si queremos rallar un objeto, sabemos que hemos de utilizar un objeto de un material más duro que el que pretendemos rallar, lo cual nos permite eliminar un conjunto de materiales. Esa estrategia evita comportamientos inútiles, somos capaces de anticipar el resultado de la acción. Como

ejercicio mental, podríamos imaginar a una persona intentando rallar un coche con un jersey de lana, ¿qué calificativo le daríamos?

Ese conocimiento que todos los humanos tenemos sobre nuestro mundo se denomina de *sentido común (common sense)* y es imprescindible para alcanzar un comportamiento inteligente de tipo general tal como aumentaremos en el capítulo 6. Sin embargo, es un tipo de conocimiento que es difícil de proporcionar a un ordenador. Es un conocimiento muy vasto, que requiere diferentes tipos de inferencia para su uso y que se resiste a la formalización, aunque ha habido intentos. A lo largo de este libro insistiremos en repetidas ocasiones sobre la crucial importancia de dotar a las máquinas de conocimientos de sentido común.

Desde mi punto de vista técnico, se puede afirmar que el papel del conocimiento es reducir los problemas de IA a problemas de búsqueda que se puedan resolver en tiempo razonable (Ginsberg, 1993). Como veremos en el próximo apartado, es relativamente sencillo expresar un problema de IA como una búsqueda en espacio de estados, pero esto no significa que hayamos resuelto el problema: esos espacios suelen ser extraordinariamente grandes, por lo que la búsqueda tiene muchas probabilidades de ser infructuosa (es como buscar “una aguja en un pajar”). El papel del conocimiento es reducir esos espacios de forma que la búsqueda sea efectiva en un tiempo razonable.

Dada la importancia de representar adecuadamente el conocimiento, se han propuesto distintas formas de representación. Posiblemente las más extendidas sean las representaciones basadas en la lógica —como ya hemos indicado al comienzo de este apartado— o en subconjuntos computables de esta (por ejemplo, cláusulas de Horn en lógica de predicados). También habría que mencionar las *reglas de producción*, el método de representar conocimientos más común en los sistemas expertos[8], aunque planteadas mucho antes (Newell y Simón, 1972), las *redes semánticas* propuestas por Woods, grafos dirigidos etiquetados en donde la inferencia se realiza como recorridos en el grafo (Bobrow y Collins, 1975) y la representación basada en *frames* propuesta por Minsky (Winston, 1975), orientada a mantener propiedades que no cambian con la realización de una acción. Todos estas ideas forman un conjunto de herramientas que pueden utilizarse en la constricción de programas inteligentes.

El periodo de máximo interés en la IA por la representación del conocimiento es en torno a la década de 1980, cuando se desarrollaron lenguajes de programación que ya proporcionaban de forma nativa algunas de estas representaciones. Actualmente, el objetivo de encontrar una forma general de representación del conocimiento que sea aplicable a cualquier tarea y que permita diversos tipos de inferencia no recibe tanta atención. Tras el año 2000, la comunidad de IA se está centrando en resolver problemas con un gran rendimiento (véase el capítulo 5) más que en la generalidad de los métodos que desarrolla. La facilidad humana para manejar conocimientos y

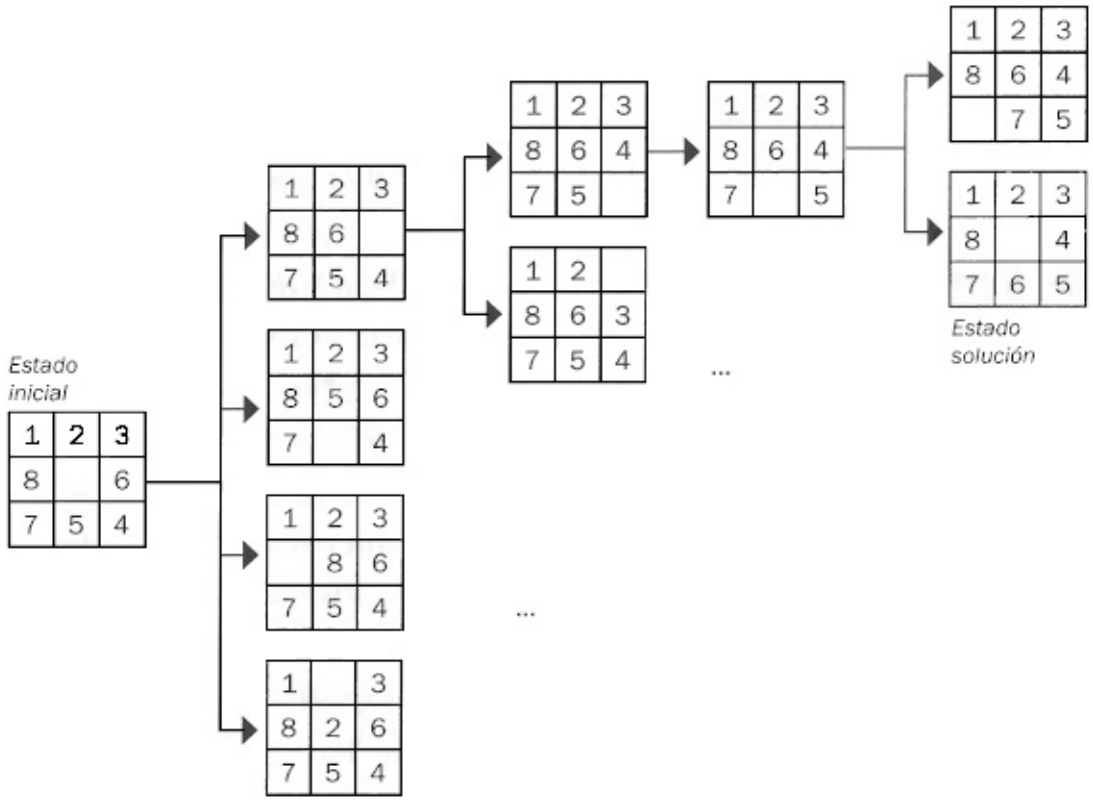
realizar diversas inferencias sobre ellos —incluso contradictorias en ocasiones— continúa siendo un reto muy difícil de emular mediante computación[9].

Búsqueda heurística

Una de las estrategias de IA más aplicada desde sus inicios ha sido la de búsqueda. Dado un problema, formado por unos constituyentes elementales y unos operadores, se puede formalizar como un problema de búsqueda: la idea básica consiste en considerar una configuración concreta del problema como un estado. Aplicando todos los operadores del problema a ese estado, se construyen todos los estados sucesores. Aplicando de nuevo todos los operadores a esos sucesores, generamos los sucesores de los sucesores, etc. De esta manera, se construye un enorme grafo dirigido, en donde sus nodos forman el conjunto de todos los estados posibles del problema, también conocido como su *espacio de estados*. Este grafo es prohibitivamente grande y nunca se desarrolla en la memoria del ordenador. En ocasiones se denomina *grafo implícito*, que subraya su carácter conceptual. A lo más que se llega es a desarrollar alguna pequeña parte de él, denominada *subgrafo explícito*. En muchos casos, resolver un problema es equivalente a encontrar un camino en ese grafo dirigido, entre el nodo inicial —que representa el estado inicial del sistema— y un nodo objetivo —que representa al estado deseado— (Newell y Simon, 1972).

Un ejemplo clásico es el puzle de 8, un tablero 3 × 3 que tiene 8 piezas movibles numeradas del 1 al 8 más un hueco. Dichas piezas se pueden deslizar en horizontal o en vertical si el hueco está en su misma fila o columna. Hay cuatro operadores: mover una pieza arriba, abajo, a la izquierda y a la derecha. El problema es encontrar la secuencia de movimientos más corta que pasa de una configuración de piezas a otra (figura 1).

Figura 1
Árbol de búsqueda de puzzle de 8. En cada nodo se han evitado los operadores que regeneran el nodo anterior. El camino ‘solución’ comprende cuatro pasos.



El espacio de estados se puede explorar mediante un *árbol de búsqueda* —que se construye a partir del estado inicial considerando todos los estados que se pueden alcanzar por la aplicación de un operador, de dos operadores, de tres, etc.— con las estrategias de recorrido en profundidad o en anchura. Estas estrategias se denominan de *búsqueda ciega* (o también de *fuerza bruta*) porque no utilizan ninguna función que oriente la búsqueda, simplemente recorren el árbol hasta encontrar un nodo solución. En problemas no triviales el espacio de estados es muy grande y la búsqueda ciega no alcanza soluciones en tiempos aceptables. En *búsqueda heurística*, esta considerable dificultad se ha intentado eludir —con más o menos éxito— mediante el uso de heurísticas: se trata de criterios para la resolución de problemas que suelen dar buen resultado en la mayoría de los casos. Desde el punto de vista cuantitativo, se han desarrollado algoritmos que utilizan una función heurística para estimar la distancia de un nodo a la solución, como el algoritmo A*. Este algoritmo evalúa cada nodo x visitado con una función heurística $g+h$, en donde $g(x)$ es el coste desde el nodo inicial hasta x y h(x) es una estimación del coste desde el nodo x hasta la solución más cercana. Esta estimación ha de ser admisible, es decir, no sobrepasar el coste real. A* expande los nodos más prometedores, según esta función heurística, entre los nodos visitados que aún no han sido expandidos, hasta encontrar una solución. A* es completo y termina al seleccionar para expandir una solución (Russell y Norvig, 2010).

Planificación

Se ha trabajado sobre el tema de planificación prácticamente desde los comienzos de la IA. Dados un conjunto de objetos y uno (o varios) agentes que pueden realizar acciones predefinidas, se trata de encontrar la secuencia de acciones que permite alcanzar un determinado objetivo (una determinada configuración de los objetos). Por ejemplo, si tenemos tres bloques sobre la superficie de una mesa (objetos), un brazo robot (el agente) capaz de cogerlos, levantarlos, apilarlos y desapilarlos (acciones) y queremos formar una torre con dichos bloques (objetivo), hay que sintetizar la secuencia de acciones que, aplicada sobre los objetos, nos lleve al objetivo deseado. Todo esto se hace en la *mente* del agente antes de ejecutar el plan. La capacidad de planificación es claramente necesaria para que un agente alcance una inteligencia de propósito general (de hecho, un indicio de inteligencia en mamíferos superiores es su capacidad para anticipar el resultado de las acciones y encadenarlas: por ejemplo, un chimpancé que mueve una banqueta justo debajo de un racimo de plátanos para, a continuación, subirse a ella y alcanzarlos).

La cuestión de la representación está muy presente en planificación. Se ha de representar el estado del mundo (en el ejemplo anterior, los bloques sobre la mesa), el objetivo (la torre de bloques) y las acciones que pueden realizar los agentes. Un estado del mundo se suele describir por medio de predicados. Por ejemplo, para decir que los objetos *A*, *B* y *C* están sobre el objeto *mesa*, se suele utilizar el predicado *sobre* (*X, Y*), que se evalúa a cierto si y solo si el objeto *X* está sobre el objeto *Y*. En nuestro caso, el estado inicial:

$$sobre\ (A, mesa) \wedge sobre\ (B, mesa) \wedge sobre\ (C, mesa)$$

y el estado final es:

$$sobre\ (A, mesa) \wedge sobre\ (B, A) \wedge sobre\ (C, B)$$

A principio de los años setenta se desarrolló el planificador STRIPS con una descripción de las acciones que se ha convertido en un estándar en la literatura de planificación. Una acción tiene tres elementos clave: las precondiciones, los predicados a añadir y los predicados a borrar (preconditions, add y delete lists). Para que una acción pueda ser aplicada en un estado, este ha de satisfacer sus precondiciones. Cuando se realiza la acción, este estado se transforma en un nuevo estado añadiendo los predicados que están en la lista “añadir” y borrando los que parecen en la lista “borrar”. Una acción se incluye en el plan cuando contiene en la lista “añadir” un objetivo del problema, y sus precondiciones se vuelven subobjetivos a cumplir. Este esquema de sustitución de unos objetivos por otros, idealmente más sencillos, es lo que se conoce como subgoaling. Esta estrategia de sustitución de objetivos por subobjetivos ya la vimos en algunos de los primeros programas de IA descritos en el capítulo 1. A menudo una acción específica lo que cambia en un estado, pero se ha de entender que los predicados que no cambian permanecen y se

siguen cumpliendo. Esta es la idea de los frame axioms, que simplemente especifican que los predicados no alterados por las acciones que forman el plan se siguen cumpliendo en estados posteriores al estado inicial. Todas estas nociones se formalizaron lógicamente mediante el denominado situation calculus (Russell y Norvig, 2010).

FIGURA 2
Anomalía de Sussman.
(a) Estado inicial; (b) estado objetivo.

STRIPS es un ejemplo de planificador lineal, es decir, construye el plan como una secuencia de acciones totalmente ordenada. Un problema que se encontró STRIPS fue la interacción entre subobjetivos, la denominada *anomalía de Sussman*, detectada en su tesis doctoral. Supongamos que tenemos el estado inicial indicado en la figura 2 (a) y el estado objetivo en la figura 2(b). Este estado objetivo se compone de dos subobjetivos: *sobre (A, B)* ∧ *sobre (B, C)*. Si empezamos por el primer subobjetivo, alcanzamos el estado de la figura 3(a), pero para alcanzar el segundo subobjetivo hemos de deshacer el ya alcanzado (porque no se puede mover un bloque con algo encima). Si empezamos por el segundo subobjetivo, alcanzamos el estado de la figura 3(b), pero de nuevo lo perdemos si queremos alcanzar el otro subobjetivo (hay que deshacer la torre para sacar el bloque *A* que está en la base).

Figura 3
Interacción entre subobjetivos.

En ambos casos no se alcanza el estado objetivo porque se deshace el primer subobjetivo al intentar alcanzar el segundo.

La solución a este problema vino de la mano de la planificación no lineal, en donde el plan se iba construyendo como un *conjunto de acciones parcialmente ordenado* (POP, *partial order planning*). En este nuevo esquema se mantiene la justificación de por qué se introdujeron acciones en el plan para alcanzar subobjetivos o precondiciones no satisfechos (*causal links*). Cuando se añade una nueva acción al plan, se analiza si amenaza a otras acciones (es decir, si elimina precondiciones requeridas por esas acciones) o si es amenazada por acciones ya presentes en el plan. En ambos casos, se establecen restricciones de ordenación entre las acciones que se amenazan, de forma que no afecten a sus precondiciones. La linearización del conjunto final de acciones las secuencia en un orden total, compatible con el orden parcial construido. La planificación de orden parcial es sólida (los planes que genera son correctos) y completa (si para un problema concreto existe un plan, POP lo encuentra). POP es un ejemplo de *mínimo compromiso* (*least commitment*), según el cual en la construcción de una solución solo se toman las decisiones imprescindibles con respecto a los elementos incluidos, retrasando las decisiones no imprescindibles (Ghallab *et al*., 2004).

Sistemas expertos

Después del GPS, Newell y Simon siguieron investigando la manera en que un ser humano resuelve problemas e idearon un modelo basado en lo que llamaron *sistema de producción* (Newell y Simon, 1972), que consistía en representar conocimientos sobre cómo resolver un problema mediante reglas de producción. Estas son del tipo *SI-ENTONCES*, donde la parte SI contenía alguna premisa que debía cumplirse con el fin de poder aplicar la parte *ENTONCES*, que consistía en una acción. Las reglas de producción se almacenarían en lo que llamaron *memoria a largo plazo*, que procesaría datos sobre tareas concretas a resolver, contenidos en otra memoria a corto

plazo, de forma que cuando algún dato cumpliera la condición expresada en la premisa de alguna regla esta se activaría y se llevaría a cabo la acción indicada en dicha regla. Habría acciones internas consistentes en añadir o borrar algún dato en la memoria a corto plazo. También podría haber acciones externas que cambiarían el entorno del sistema, por ejemplo, mediante un actuador conectado al sistema de producción. En ambos casos el estado de la memoria a corto plazo se modificaría y por lo tanto se podrían aplicar otras reglas de producción que a su vez volverían a modificar los datos de la memoria a corto plazo y así sucesivamente hasta, eventualmente, resolver la tarea. Cuando los datos pudieran activar más de una regla de producción, un *solucionador de conflictos* decidiría cuál aplicar. Newell y Simon no idearon esta arquitectura con la intención de programarla en un ordenador, sino como un posible modelo de cómo los seres humanos podríamos resolver problemas. Es decir, que fue una contribución al estudio de la cognición humana y no una contribución a la IA. Sin embargo, poco después, la idea de representar conocimientos mediante reglas del tipo *SI-ENTONCES* dio lugar a los *sistemas expertos*, posiblemente una de las técnicas más importantes que ha dado la IA.

El primer sistema experto propiamente dicho fue Heuristic Dendral, desarrollado durante 10 años desde principios de los años setenta en la Universidad de Stanford, en el ámbito de la química orgánica. Concretamente era capaz de hipotetizar la estructura topológica de un compuesto químico (la disposición espacial de sus átomos) para interpretar su espectrograma de masas, usando conocimientos de químicos expertos en analizar la información proporcionada por el espectrómetro. Estos conocimientos se representaron mediante reglas *SI-ENTONCES*. Un ejemplo de una de las muchas reglas de Heuristic Dendral es:

SI el espectrograma tiene dos picos *A* y *B* tales que $A + B = M + 28$ (siendo *M* la masa de la molécula), y $X - 28$ es un pico alto, $eY - 28$ es un pico alto, y por lo menos uno de los picos *A* o *B* es alto, *ENTONCES* la molécula contiene un grupo = *C* = O (cetona).

La capacidad de Heuristic Dendral para resolver problemas que requieren conocimientos muy especializados enseguida convenció a muchos investigadores en IA que esta metodología podría aplicarse a prácticamente cualquier campo del saber. Poco después, también en la Universidad de Stanford, Shortliffe, en colaboración con Buchanan y Cohen, desarrolló otro famoso sistema experto, llamado Mycin (Shortliffe, 1975), en el contexto de su tesis doctoral en medicina, usando también reglas del tipo SI-ENTONCES para representar el conocimiento de médicos expertos en enfermedades infecciosas. La parte *si* de una regla típica de Mycin describía posibles síntomas y la parte *entonces* expresaba una posible causa de dichos síntomas. Además de conocimientos que permitían diagnosticar el agente causante de la infección, Mycin también contenía conocimientos acerca del tratamiento a

administrar. Un aspecto muy importante de Mycin era tenían asociados *factores de certeza* que medían la confianza del experto en que los síntomas expresados en la premisa de regla fueran consecuencia de la causa expresada en la conclusión de dicha regla. Los factores de certeza podían tomar valores entre −1 y 1, por lo que no eran propiamente probabilidades, a pesar de que tenían su origen en los conceptos llamados *medida de credibilidad* y *medida de incredulidad*, basados a su vez en la teoría de la probabilidad. Un ejemplo de regla en Mycin es:

Si el organismo es GRAMNEGATIVO, y su morfología es de tipo BASTONCILLO y el organismo es ANAERÓBICO, entonces es un organismo BACTEROIDE con una certeza de *0,6*.

Mycin aplicaba las reglas mediante *encadenamiento hacia atrás*. Por ejemplo, la regla anterior se aplicaba cuando el sistema tenía como objetivo determinar si el organismo era una bacteria, para lo cual debía determinar el grado de certeza de las tres premisas expresadas en la parte *si*. Si estas premisas eran a su vez deducibles a partir de otras reglas, Mycin aplicaba dichas reglas comprobando el grado de certeza de sus premisas, y así sucesivamente hasta llegar a premisas cuya certeza no era deducible a partir de más reglas y, por consiguiente, era necesario preguntar al médico usuario acerca de su certeza. Mycin era un sistema de ayuda a la toma de decisiones para médicos.

Otra característica importante de Mycin era que, contrariamente a Heuristic Dendral, el mecanismo inferencial, que llevaba a cabo el encadenamiento hacia atrás de las reglas, era un programa llamado *motor de inferencia*, separado e independiente de las reglas que componían lo que se denomina la *base de conocimientos*. Esto permite reutilizar el mismo motor de inferencia para otras aplicaciones simplemente cambiando la base de conocimientos, es decir, el conjunto de reglas. De hecho, Mycin dio lugar al entorno de desarrollo de sistemas expertos Emycin, que se utilizó para aplicaciones distintas a la medicina. A partir de aquel momento, proliferaron por todo el mundo las aplicaciones de sistemas expertos, así como los entornos de desarrollo similares a Emycin, y se crearon empresas dedicadas al desarrollo de aplicaciones mediante la metodología de los sistemas expertos.

Otro sistema experto pionero, también desarrollado en Stanford, fue Prospector en el campo de la geología, concretamente como consulta sobre posibles depósitos de minerales. Prospector-ll, una versión ampliada y mejorada de Prospector, sigue todavía en uso. Contiene conocimientos, representados por reglas organizadas mediante una red semántica, sobre 140 tipos de minerales y en cuestión de minutos cualquier geólogo puede introducir datos geológicos sobre el subsuelo de una zona dada y recibir sugerencias sobre la existencia de posibles minerales en ese lugar. Prospector usa la teoría de la probabilidad, concretamente relaciones de

verosimilitud, e inferencia bayesiana para modelizar la incertidumbre de los conocimientos que contiene y razonar a partir de estos.

A mediados de los ochenta se estima que ya había centenares de sistemas expertos desarrollados en todo el mundo en distintos dominios de aplicación. Los espectaculares éxitos alcanzados con los sistemas expertos cambiaron significativamente las prioridades en la investigación en IA, ya que, en lugar de focalizar esfuerzos en desarrollar sistemas generales de resolución de problemas, la mayoría de los investigadores se dedicaron a resolver problemas muy específicos, es decir, a desarrollar aplicaciones en ámbitos muy delimitados donde los expertos humanos usaban conocimientos muy concretos para resolver los problemas a los que se enfrentaban. Se popularizó así la hipótesis de que *el conocimiento es poder*. Sin embargo, al cabo de poco tiempo afloraron importantes limitaciones de los sistemas expertos, en particular el elevado coste de los procesos de adquisición de conocimientos y la dificultad de su actualización, así como su fragilidad.

Los sistemas expertos son frágiles porque únicamente pueden resolver aquellas situaciones específicas que están representadas en sus bases de reglas. Si un problema a resolver requiere algún conocimiento que no está presente en sus reglas, entonces, a pesar de que se trate de una situación dentro de su área de especialización, fracasan estrepitosamente. Por ejemplo, si un sistema experto en medicina no contiene el conocimiento de que únicamente las mujeres pueden tener embarazos, dicho sistema podría preguntar por los resultados de una prueba prenatal para pacientes varones. Esta fragilidad está claramente relacionada con la ausencia de conocimientos de sentido común en los sistemas expertos.

Aprendizaje automático

En los años sesenta y setenta el aprendizaje automático se limitaba casi exclusivamente al uso de redes neuronales artificiales para aprender a clasificar datos y no ocupaba una posición central en la investigación en IA. Sin embargo, a partir de los ochenta, se empezaron a desarrollar nuevos métodos de aprendizaje que lo convirtieron en una de las áreas más activas de la IA. Posiblemente el método de aprendizaje más conocido a partir de la década de los ochenta es el aprendizaje inductivo de *árboles de decisión*. La mayoría de métodos de aprendizaje automático infieren hipótesis a partir de datos. Dichas inferencias son inductivas, contrariamente a las inferencias deductivas típicas de la lógica matemática. La diferencia es que las inferencias deductivas son consecuencia lógica de un conjunto de premisas y por lo tanto son conclusiones no falseables, mientras que las inferencias inductivas son solamente hipótesis susceptibles de ser falseadas según datos adicionales. Un ejemplo típico es que si un conjunto grande de datos sobre pájaros solo contiene información sobre pájaros blancos, entonces la hipótesis a la que llegaría un algoritmo de aprendizaje inductivo es que todos los pájaros son blancos. Si más adelante se

proporcionan datos adicionales que contienen información sobre pájaros negros, entonces la hipótesis anterior queda falseada.

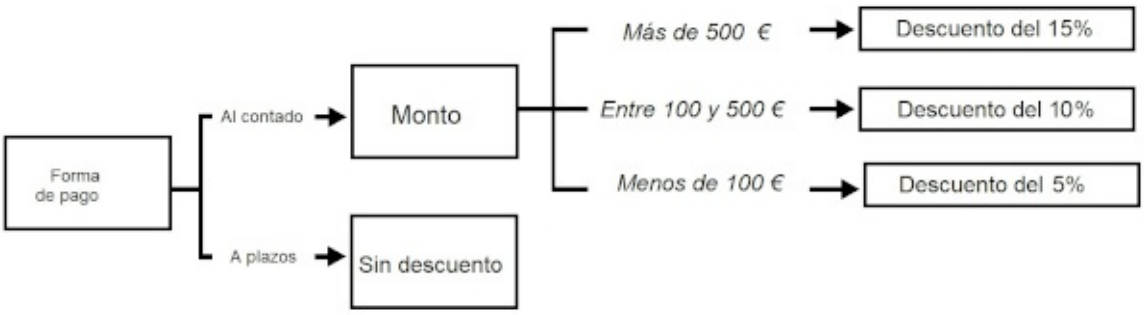
La construcción automática de árboles de decisión es sin duda el método de aprendizaje más conocido y aplicado. Los árboles de decisión consisten en una secuencia de decisiones con el objetivo de determinar una categoría o asignar un valor numérico a un atributo objetivo según los valores numéricos o categorías conocidas de otros atributos. Por ejemplo, el árbol de decisión de la figura 4 es útil para decidir entre tres posibles niveles de descuento a aplicar según los atributos *forma de pago* (con dos posibles valores) y *monto* (tres posibles valores).

Posiblemente, el primer sistema de construcción de árboles de decisión fue el que desarrolló Edward Feigenbaum a finales de los años cincuenta en su tesis doctoral, donde estudió los procesos de aprendizaje verbal de los seres humanos. Un experimento clásico en psicología para comprobar la habilidad para el aprendizaje verbal consistía en mostrar a los sujetos parejas de sílabas carentes de sentido como, por ejemplo, *tuc-rad* o *nam-jut*, donde la primera sílaba de cada par es el *estimulo* y la segunda la *respuestal* Después de ver un cierto número de tales pares de forma repetida, los sujetos recibían solamente una sílaba estímulo y debían dar la respuesta correcta. Por ejemplo, si recibían *nam* debían responder *jut*. El programa que desarrolló Feigenbaum, llamado EPAM (Elementary Pereeiver and Memorizer) hacía lo mismo, es decir, recibía, durante la fase de entrenamiento, pares de sílabas con las cuales construía un árbol de decisión (de hecho él lo llamó *red discriminante*) cuyos nodos interiores eran letras individuales y los nodos terminales contenían las respuestas, aprendiendo así a asociar los pares estímulo-respuesta. En la fase de test, EPAM recibía un árbol cuyo nodo terminal daba la respuesta correcta.

El algoritmo ID3 de aprendizaje de árboles de decisión

El sistema de aprendizaje inductivo de árboles de decisión más conocido es ID3 (por Iterative Dichotomizer), desarrollado por Quinlan. ID3 construye árboles de decisión mediante un proceso iterativo de partición del conjunto de datos de entrenamiento basado en los valores de los atributos que describen dichos datos.

Figura 4
Árbol decisión.



En la figura 4 aparece un árbol de decisión para clasificar pagos. El primer atributo, “forma de pago”, divide los datos en dos clases: una para aquellos casos en que el pago es al contado y otra para pago aplazado. En este último caso ya no se requiere volver a particionar el conjunto de datos, puesto que, en todos ellos, la decisión es pagar el monto total, es decir, “sin descuento”. Sin embargo, en el otro caso todavía se requiere otra interacción con base en el atributo “monto” para determinar el descuento aplicable mediante una nueva partición de los datos restantes de acuerdo a los tres valores posibles del atributo. El proceso termina aquí, ya que estos dos atributos son suficientes para determinar si se debe o no aplicar un descuento y, en caso positivo, de cuánto debe ser dicho descuento. ID3 decide el orden de aparición de los atributos en el árbol (en el ejemplo, primero la forma de pago y después el monto) mediante el cálculo de cuál de los atributos discrimina mejor los datos. Esta medida de discriminación se llama ganancia de información y se basa en la teoría de la información de Shannon. Versiones extendidas y mejoradas de ID3, como C4.5, C5.0 o ACLS, se han empleado extensivamente en múltiples aplicaciones de *minería de datos* (*data mining*) de alta complejidad, con cientos de atributos y decenas de miles de datos.

Visión por computador

Los seres humanos y otros animales adquirimos una gran cantidad de información mediante nuestra capacidad visual. El objetivo de la visión por computador es dotar de esta capacidad a las máquinas. Durante los años sesenta, la motivación principal para desarrollar sistemas de visión era poder proporcionar información para guiar brazos robóticos. Es lo que se conocía por *hand-eye research*. A lo largo de más de una década, una serie de estudiantes del MIT (Patrick Winston, Thomas Binford, Berthold Horn, Eugene Freuder, Adolfo Guzmán, David Waltz y David Hufman) desarrollaron sus tesis doctorales alrededor de lo que se conoce como *el mundo de los bloques*, consistente en escenas que contenían simples objetos geométricos tridimensionales similares a los ya mencionados anteriormente. Los primeros algoritmos que se desarrollaron encontraban los bordes de los objetos mediante *buscadores de líneas*. Con base en estas líneas, otros algoritmos identificaban los

distintos objetos presentes en la imagen y, una vez identificados, un brazo robótico ejecutaba un plan que consistía en montar y desmontar estructuras según dichos objetos (por ejemplo, agarrar una pirámide y colocarla encima de un cubo). La secuencia de acciones y movimientos necesarios para ejecutar el plan se planificaban mediante algoritmos de planificación. En 1972, Winston utilizó el mundo de los bloques para desarrollar algoritmos con el fin de aprender conceptos como *arco* (dos columnas separadas formadas por cubos unos encima de otros sosteniendo un objeto piramidal).

Simultáneamente, en Stanford, en el grupo de John McCarthy, también investigaban el problema del *hand-eye* usando *el mundo de los bloques*, apilando y desapilando torres formadas por cubos de distintos colores. A principios de los años setenta resolvieron el puzle llamado *instant sanity*, que consistía en apilar cuatro cubos cuyas caras estaban pintadas con cuatro colores distintos, de manera que cada uno de los cuatro lados de la torre tuviese los cuatro colores.

No toda la investigación pionera en visión se limitó al mundo de los bloques. También consiguieron, mediante una compleja secuencia de acciones, que un sistema de visión acoplado con una mano artificial electromecánica montara la bomba de agua de un coche Ford T, usando herramientas mecánicas que permitían atornillar unas piezas con otras de forma similar a como lo haría una persona. La importancia de la visión por computador para aplicaciones industriales (por ejemplo, el montaje de dispositivos mecánicos) hizo que florecieran rápidamente investigaciones pioneras similares en Japón, concretamente en los laboratorios de investigación de Hitachi, y también en Europa. La visión por computador sigue siendo una de las áreas más activas en IA. A pesar de los extraordinarios progresos alcanzados, sobre todo en reconocimiento de caras, problemas básicos como, por ejemplo, el reconocimiento general de objetos o el análisis interpretativo de escenas no están resueltos.

Robots móviles

A principios de los años sesenta, investigadores de la Universidad John Hopkins construyeron un robot móvil que, mediante sonar y fotocélulas, podía moverse a lo largo de pasillos con paredes blancas, buscando enchufes de color oscuro a los cuales se enchufaba para recargar sus baterías. Unos años más tarde, en el Stanford Research Institute, un grupo de investigadores liderado por Charles Rosen construyó uno de los robots móviles más famosos de la historia de la IA, el Shakey, que combinaba la capacidad de reconocer objetos sencillos mediante redes neuronales con un sistema de planificación de movimientos que le permitieron navegar en entornos conteniendo obstáculos simples (cajas con formas geométricas regulares). El nombre del robot se debe a que temblaba ostensiblemente cuando se detenía, para evitar los obstáculos, o al colisionar o al llegar a su destino. Shakey estaba equipado con una cámara de televisión para ver los obstáculos y un sensor láser para calcular las distancias a las

paredes y a los obstáculos. También disponía de sensores de contacto situados alrededor de la base que paraban inmediatamente los motores en caso de colisión. Se movía en varias habitaciones rectangulares separadas por paredes bajas (de forma que una persona pudiera observar todo el entorno desde el exterior), conectadas de forma que el robot podía ir de una habitación a otra sin tener que abrir puertas.

El *software* estaba organizado en tres capas. La capa interna contenía los programas que controlaban los motores y recibían la información de los sensores. La capa intermedia procesaba las imágenes de la cámara y contenía un *software* de navegación que controlaba las acciones básicas de giros y traslaciones que movían al robot. La capa externa contenía el *software* de planificación de toda la trayectoria que debía seguir el robot para desplazarse de una posición inicial a una final evitando los obstáculos. La presencia de estos imposibilitaba planificar una trayectoria en línea recta entre la posición inicial y final, es decir, el planificador debía encontrar una secuencia de posiciones intermedias por las que el robot debía pasar de forma que desde una posición a la siguiente la trayectoria fuese recta. Dichas posiciones intermedias se correspondían con esquinas de los obstáculos, pero a una cierta distancia para evitar colisiones. De entre las posibles trayectorias, el planificador seleccionaba la más corta. El planificador, denominado STRIPS**[10]**, que permitía llevar a cabo esta trayectoria óptima se basaba en el algoritmo de búsqueda heurística A***[11]**. Un plan típico de STRIPS consistía en una secuencia de acciones que se representaban internamente en el lenguaje de STRIPS mediante expresiones basadas en lógica de primer orden del estilo *GOTO ((X1, Y1) (X2, Y2))* para mover a Shakey de la posición con coordenadas (XI, Yl) a la posición *(X2, Y2)*; o *AT(ROBOT, XI, Yl)* para expresar que Shakey se encontraba en la posición (XI, Yl). Además de desplazarse de un lugar a otro, Shakey también ejecutaba acciones, como empujar un objeto. El modelo del entorno también se representaba mediante el mismo lenguaje lógico. Por ejemplo, *JOINSROOM(Dl, Rl, R2)* expresa que las habitaciones R1 y R2 compartían el umbral DI que las conectaba. Extensiones de STRIPS dieron lugar a planificadores cada vez más eficientes. Incluso todavía usamos la expresión *operadores* STRIPS para referirnos a precondiciones que se tienen que cumplir para ejecutar acciones de muchos planificadores modernos.

Procesamiento del lenguaje

Uno de los objetivos clásicos de la IA ha sido conseguir *procesar el lenguaje natural* (Natural Language Processing, NLP), es decir, el lenguaje de los humanos: español, inglés, etc. Se trata de un objetivo muy ambicioso y, en realidad, enmarca todo un tema de investigación que tiene diferentes objetivos concretos para los que se utilizan técnicas específicas. Por ejemplo, uno puede desear que un ordenador sea capaz de escribir en lenguaje natural (generación de lenguaje), que realice traducción de un idioma a otro (traducción automática) o que pueda responder a preguntas sobre un

texto (comprensión). En este apartado nos centramos en textos escritos en lenguaje natural que queremos procesar en alguna de las dimensiones anteriores (esto evita cuestiones asociadas al proceso del habla, como tratamiento de la señal, segmentación de palabras, etc.).

Tras el nacimiento de la IA en 1956 y durante la década de 1960, se realizaron diversos sistemas de NLP (capítulo 1) que mostraron la inmensidad y dificultad del tema; por ese motivo, funcionaban con vocabularios muy reducidos y en temas muy específicos. Poco después se desarrolló SHRDLU (Winograd, 1972), un sistema que procesaba el lenguaje en un entorno léxico también restringido, constituido por un mundo de bloques**[12]** sobre el cual se podían formular preguntas respecto a las configuraciones de los bloques al sistema de visión y también dar órdenes en lenguaje natural al planificador. En el sistema de pregunta-respuesta, Eliza simulaba una conversación, por escrito, con un léxico pretendidamente más amplio que SHRDLU, con un psicoterapeuta rogeriano sin comprender absolutamente nada del significado de las palabras.

Hasta los años setenta o bien el vocabulario era muy reducido con el fin de poder captar el contenido semántico de las frases o bien se obviaba completamente la semántica. Veremos ahora que posteriormente se empezó a enfocar el problema del lenguaje usando las herramientas que proporciona la lingüística, con el fin de tener en cuenta la sintaxis, la semántica e incluso la pragmática en lo que se conoce como la aproximación clásica al procesamiento del lenguaje natural. Un primer ejemplo de esta aproximación lo tenemos en los trabajos de Roger Schank en los llamados *grafos de dependencia conceptual*, inspirados en las redes semánticas, aunque más ricos en capacidad expresiva. Estos grafos se utilizaban para representar el significado independientemente del idioma en que estuviera expresada la frase de entrada. Una vez obtenida dicha representación, libre de ambigüedades, un módulo de generación de lenguaje producía una frase de salida sintácticamente correcta en otro idioma. Los grafos de dependencia conceptual fueron el componente básico de posteriores trabajos de Schank y sus alumnos sobre los *scripts* (Schank y Abelson, 1977) para representar conocimientos específicos y pragmáticos sobre situaciones típicas con las que nos encontramos frecuentemente como, por ejemplo, lo que típicamente uno hace cuando va a un restaurante o una tienda. Los *Scripts* permiten explicar los razonamientos que llevamos a cabo cuando oímos una historia cotidiana. Por ejemplo, si oímos que “Juan fue a un restaurante y pidió lasaña”, el *script* correspondiente a “restaurante” permite razonablemente suponer que el menú incluía lasaña, y si a continuación oímos que finalmente comió otra cosa distinta, entonces podemos inferir que la lasaña se había terminado. Adicionalmente, también podemos deducir que, entre otras muchas cosas, en el restaurante había camareros en lugar de ser un *self-service* y que pagó la cuenta antes de salir. Schank y sus colaboradores argumentan que las personas desde pequeñas adquirimos y memorizamos en nuestro

cerebro un gran número de *Scripts* que nos permiten comprender multitud de situaciones con las que nos encontramos.

En la aproximación clásica, es común traducir una frase en lenguaje natural a una representación interna, de forma que las ambigüedades, referencias, etc., estén resueltas. A partir de esa representación interna, que contiene todo el significado de la frase, se elaboran respuestas, traducción a otro idioma, etc. En este proceso se distinguen varias fases: el *análisis morfológico* (o léxico), el *análisis sintáctico*, el *análisis semántico* y el *análisis pragmático*. El morfológico se focaliza en cada palabra aislada para determinar todas las características (tipo, género, número, etc.) que serán necesarias en fases posteriores. Aquí ya nos encontramos con las primeras muestras de ambigüedad: palabras con varios significados (por ejemplo, banco como institución financiera y banco para sentarse). Alguna de estas ambigüedades se pueden resolver en las fases posteriores del análisis, aunque hay frases inherentemente ambiguas: por ejemplo, “hay un banco nuevo en la plaza”.

El *análisis sintáctico* (*parsing*) considera el análisis de la frase con respecto a una determinada gramática para determinar el papel de cada palabra. Idealmente, se espera construir un árbol de derivación a partir de la gramática y de la frase de entrada. Este proceso tiene dificultades: si limitamos la generalidad de la frase a tratar, el análisis sintáctico es realizable, pero cuando permitimos la completa generalidad del lenguaje (los lenguajes naturales están lleno de excepciones y casos particulares), la tarea se vuelve ardua y compleja. Varios tipos de gramáticas se han utilizado en esta fase, en particular las ATN (Winograd, I983). Una ATN se representa por un grafo dirigido, con etiquetas en los arcos. Estas etiquetas denotan categorías de palabras (sustantivo, verbo, etc.), saltos entre nodos, saltos a otra ATN o una etiqueta particular *done* que significa que el proceso se ha terminado con éxito. De nuevo, la ambigüedad aparece, ya que hay estructuras sintácticas con diferentes significados: por ejemplo, *ayudar a vencen al equipo X* (¿vence el equipo X o vence el equipo rival?).

El análisis semántico utiliza los significados de las palabras para extender y desambiguar el resultado del análisis sintáctico (Allen, 1995). El análisis semántico se basa en reglas que pretenden capturar el significado de clases de acciones. Por ejemplo, si analizamos frases con el verbo dar, una regla semántica exige que el agente que lleva a cabo la acción sea animado. De nuevo, pueden quedar frases con significado ambiguo: por ejemplo, “todos los traductores deben saber tres lenguas comunitarias” (¿tres cualesquiera o todos las mismas?).

Tras el análisis semántico, aún pueden quedar referentes sin resolver; entonces se realiza un análisis pragmático en donde se considera el conocimiento del dominio para completar la comprensión de la frase. En otras palabras, se interpreta la frase en un contexto amplio para resolver las ambigüedades que permanecen. Esta fase puede tener unos límites muy elásticos. Un ejemplo es la siguiente frase: “David necesitaba dinero desesperadamente. Fue a su mesa y sacó una pistola”. Tras leerla, nosotros

pensamos que va a robar, pero eso no está en la frase. Si la segunda frase fuera “Fue a la caja fuerte y sacó el anillo de diamantes”, entonces pensaríamos que va a venderlo o a empeñarlo. De nuevo, la acción futura está omitida y solo se puede obtener por los conocimientos culturales activos en el entorno donde se realiza la acción. Estos casos se conocen con el nombre genérico de *plan recognition* (Allen, 1995).

Existen diversas tareas concretas en el tratamiento del lenguaje natural. Una de ellas es clasificación de textos: dado un texto, decidir a qué clase pertenece de un conjunto predeterminado de clases. Es el caso típico de la clasificación de mensajes de correo electrónico en deseado y no deseado (o *spam*). Otra tarea es la conocida como *recuperación de información* (*information retrieval*): encontrar documentos que son relevantes para un usuario. Los buscadores de la web realizan este cometido bajo dos modelos predominantes: el de palabras clave y el de funciones de puntuación. Relacionado con esta tarea está la de *respuesta a preguntas* (*question answering*), cuyo objetivo está en responder a la pregunta del usuario no con una lista de documentos, sino con un texto o una frase. Otra tarea es la de *extracción de información* (*information extraction*), por ejemplo, extraer direcciones de páginas web o descripciones de tormentas en partes meteorológicos. Por último, mencionar la *traducción automática* (*machine translation*), la tarea de traducir un texto de un lenguaje natural a otro (a esta tarea se dedicaron muchos esfuerzos en los primeros años de la IA, hasta que se tomó conciencia de la dificultad del problema). La principal dificultad radica en que para traducir bien se necesita un conocimiento profundo del texto de entrada (incluyendo las connotaciones culturales del texto en cuestión). Los sistemas actuales de traducción automática no son perfectos y requieren la edición del resultado por un humano, aunque son más eficientes que la traducción pura (esta edición causa menos trabajo que traducir el texto de forma tradicional). A veces se prefiere editar el texto de entrada para adecuarlo conforme a las especificaciones del traductor automático.

Procesamiento del habla

El procesamiento del habla requiere procesar la señal acústica, codificada digitalmente (pero originalmente analógica) hasta convertirla en una secuencia de palabras y signos de puntuación (no hay que descuidar esos signos, ya que algunos conllevan un significado crucial: comparad “Coman niños” con “Coman, niños”).

El análisis de señal es un tema altamente técnico. Baste decir que las palabras que suenan igual, pero son diferentes (por ejemplo, en inglés *two* y *too*), ocasionan problemas. Una fase importante es la segmentación, determinar dónde acaba una palabra y empieza la siguiente. En este tema hemos de movernos en el mundo de la incertidumbre. Al final, el proceso propondrá la secuencia de palabras con más probabilidad, dada la señal de entrada.

Para calcular dicha probabilidad, hemos de estimar cuál es la probabilidad de que ocurra la secuencia de palabras candidata. Este es un cálculo complejo que se aproxima como el producto de probabilidades de que ocurran juntas cada pareja de palabras consecutivas, multiplicado por la probabilidad de la primera palabra. Para calcular esas probabilidades es necesario contar con un corpus representativo del lenguaje que queremos modelizar.

Además del modelo del lenguaje, necesitamos contar con uno acústico que indique cómo se pronuncian las palabras. Es clave disponer de un modelo que represente fielmente al hablante. De nuevo nos movemos en términos de modelos probabilísticos que emplean herramientas técnicas sofisticadas.

Los mejores sistemas actuales de procesamiento del habla reconocen entre el 80% y el 98% de palabras correctamente, dependiendo de la calidad de la señal, el idioma, la longitud de la entrada y la diversidad de hablantes. El reconocimiento del habla es fácil cuando hay un buen micrófono, un vocabulario corto, un buen modelo del idioma, pausas entre palabras y el sistema está específicamente entrenado para un único hablante (Russell y Norvig, 2010).

Capítulo 3
El invierno de la inteligencia artificial

A principios de los años setenta empezaron a aparecer voces críticas con la IA, algunas de carácter filosófico y otras de carácter ético, alrededor de la distinción entre mentes y máquinas. También en esa época, los investigadores en IA se dieron cuenta de que existían serias dificultades de carácter técnico relacionadas con el problema de la explosión combinatoria en los sistemas de IA, así como en las limitaciones de la representación del conocimiento y el razonamiento basados en la lógica matemática, y en la fragilidad de los sistemas basados en conocimientos debido a su excesiva especialización. En este capítulo discutiremos brevemente estos aspectos que, junto con las exageradas expectativas provocadas por algunos expertos en IA acerca de lo que se iba a lograr a corto plazo, contribuyeron a lo que ahora se conoce como el *invierno* de la IA, un periodo en el que cundió el desánimo entre muchos investigadores y frenó los progresos sobre todo en investigación básica. Este *invierno* se prolongó desde mediados de los setenta hasta más o menos mediados de los ochenta, momento en el que gracias a la proliferación de grandes y exitosas aplicaciones de los sistemas expertos y a su adopción por parte de grandes empresas volvió a resurgir el interés por la investigación en IA, incluida la básica. Hubo un cambio de paradigma: se abandonó en objetivo de lograr una IA de tipo general para centrarse en la IA especializada. Algunos autores hablan de un segundo *invierno* a finales de los años ochenta debido sobre todo a cierto desencanto con las limitaciones de los sistemas expertos a causa de su fragilidad (**capítulo 2**), pero, en nuestra opinión, lo que ocurrió no tuvo ni de lejos las consecuencias negativas del invierno real objeto de este capítulo.

Primeras críticas a la inteligencia artificial

En 1972, el Scientific Research Council del Reino Unido (el principal organismo de financiación de proyectos de investigación) encargó al profesor James Lighthill un informe evaluando los progresos en IA. El informe, titulado *Artificial Intelligence: A General Survey*, fue muy crítico con la mayoría de las investigaciones que se hacían en IA, argumentando que los métodos de resolución automática de problemas empleados solamente funcionaban con problemas de pequeño tamaño, los conocidos como *problemas juguete*, pero que no eran útiles para abordar problemas del mundo

real debido a la explosión combinatoria inherente a muchos problemas reales. Basándose en este informe, la financiación de la investigación en IA en Reino Unido sufrió drásticos recortes. Paralelamente, en Estados Unidos, DARPA, el principal organismo financiador de IA, también recortó significativamente la investigación básica y priorizó la financiación de proyectos con aplicaciones militares directas.

De los problemas juguete a los problemas reales: los conocimientos de sentido común

A medida que los investigadores en IA dejaban atrás los problemas juguete y abordaban problemas cada vez más complejos, reconocían las grandes dificultades que planteaba aplicar las técnicas de IA a problemas del mundo real. La explosión combinatoria no es, sin embargo, la única dificultad para resolverlos, sino que hay otro problema todavía más esencial: el de los conocimientos de sentido común. En el capítulo anterior ya nos hemos referido a este tipo de conocimientos acerca de la estructura y propiedades físicas, espaciales, temporales y sociales del entorno que nos rodea. Estos conocimientos nos permiten desenvolvernos cotidianamente en nuestro entorno sin grandes dificultades; son conocimientos generales que no adquirimos en el colegio ni mediante libros, sino más bien a través de nuestras vivencias, desde que nacemos hasta que somos adultos. Sin estos conocimientos no es posible una comprensión profunda del lenguaje ni una interpretación profunda de lo que capta un sistema de percepción visual.

Solamente una comprensión profunda de la frase “Napoleón murió en Santa Helena, Wellington sintió gran tristeza” nos permite, gracias al sentido común, inferir que “Wellington” y “Napoleón” eran personas, que “Santa Helena” es un lugar, que “Wellington” se enteró de la muerte de “Napoleón”, que “Wellington” apreciaba a “Napoleón”, que “Wellington” murió más tarde que “Napoleón”, etc. Hay millones de unidades de conocimientos de sentido común que implican relaciones temporales, relaciones de causa-efecto, etc., entre agentes, objetos y eventos que los seres humanos manejamos sin dificultad, pero que no sabemos cómo incorporarlos a una IA. En el capítulo anterior hemos mencionado más ejemplos concretos de estos conocimientos. También hemos dicho que la principal dificultad para diseñar IA de tipo general es cómo dotar de sentido común a las máquinas.

Searle y la habitación china

El filósofo John Searle, en 1980, publicó un artículo titulado “Minds, Brains, and Programs”, donde argumenta la imposibilidad de la IA (Searle, 1980). Su análisis radica en que las máquinas, contrariamente a los humanos, carecen de intencionalidad. La intencionalidad tiene que ver con dar significado a todo lo que nos rodea. Por ejemplo, un ordenador no puede saber que la secuencia de cinco letras

J-O-V-E-N en la expresión en lógica preposicional *JOVEN(JUAN)* se refiere a la propiedad de ser joven en el mundo real. Igualmente no sabe que la secuencia de cuatro letras *J-U-A-N* se refiere a una persona concreta en el mundo real. Es decir, que una máquina no conoce el significado de los símbolos que manipula.

Para apoyar sus críticas, Searle propuso en dicho artículo su famoso *experimento de la habitación china*. Searle esencialmente decía:

Supongamos que yo, que no sé absolutamente nada de chino, me encierro en una habitación donde tengo a mi disposición un conjunto muy completo de reglas, escritas en mi idioma, acerca de cómo manipular caracteres chinos y cómo generar otros caracteres chinos con base en estas manipulaciones. A continuación, desde el exterior, me proporcionan una serie de caracteres chinos y yo, aplicando las reglas, los transformo en otros caracteres chinos que devuelvo al exterior, de tal forma que estas respuestas son indistinguibles de las que daría una persona cuya lengua materna fuera el chino.

Searle llama *la habitación china* al sistema formado por él mismo, los símbolos chinos y las reglas de manipulación, y afirma que la habitación china no entiende chino porque lo único que está haciendo es una manipulación formal simbólica puramente sintáctica, es decir, sin consideraciones semánticas. Searle admite que la habitación china simplemente simula entender chino. Obviamente, la habitación china es una metáfora del funcionamiento de los ordenadores en tanto que procesadores de símbolos. En definitiva, su argumento es que la manipulación simbólica por parte de un ordenador no tiene intencionalidad y, por lo tanto, no tiene sentido.

Este artículo provocó encendidas reacciones por parte de los expertos en IA e incluso las sigue provocando. La reacción más común es que, aunque Searle no entienda chino, de hecho el sistema total consistente en la habitación entera sí. Una analogía de los que defienden esta postura es que en el cerebro cada una de las neuronas tampoco entiende el lenguaje, pero el todo, es decir, el cerebro y el cuerpo que lo contiene, sí que lo entiende.

Dreyfus y su crítica a la razón artificial no corpórea

En 1965, el filósofo Hubert Dreyfus publicó un artículo titulado “Alchemy and Artificial Intelligence”, donde afirmaba que el objetivo último de la IA, es decir, la IA fuerte de tipo general, era tan inalcanzable como el de los alquimistas del siglo XVII que pretendían transformar el plomo en oro. Dreyfus argumentaba que el cerebro procesa la información de forma global y continua, mientras que un ordenador usa un conjunto finito y discreto de operaciones deterministas aplicando reglas a un conjunto finito de datos. En este aspecto podemos ver un argumento similar al de Searle, pero

Dreyfus, en posteriores artículos y libros (Dreyfus, 1992), usó también otro argumento consistente en el hecho de que el cuerpo juega un papel crucial en la inteligencia.

Fue uno de los primeros en advocar la necesidad de que la inteligencia forme parte de un cuerpo con el que poder interaccionar con el mundo. La idea principal es que la inteligencia de los seres vivos deriva del hecho de estar situados en un entorno con el que pueden interactuar gracias a sus cuerpos. De hecho, esta necesidad de corporeidad está basada en la Fenomenología de Heidegger, que enfatiza la importancia del cuerpo con sus necesidades, deseos, placeres, penas, formas de moverse, de actuar, etc. Según Dreyfus, la IA debería modelar todos estos aspectos para alcanzar el objetivo último de la IA fuerte. Esto es, Dreyfus no niega completamente la posibilidad de la IA fuerte, pero afirma que no es posible con los métodos clásicos de la IA simbólica y no corpórea. Sin duda se trata de una idea interesante que hoy comparten muchos investigadores en IA y a la que nos volveremos a referir en el último capítulo.

Penrose o la necesidad de una nueva física

En 1989, el físico Roger Penrose publicó el libro *The Emperor's New Mind: Concerning Computers, Minds and the Laws of Physics*, en el que argumentaba que los ordenadores no podrían tener nunca consciencia ni alcanzar una inteligencia como la humana porque están limitados por los *teoremas de incompletitud* de Gödel. Dichos teoremas demuestran que entre los enunciados matemáticos no hay coincidencia exacta entre los verdaderos y los demostrables. Es decir, hay enunciados verdaderos que no son demostrables. Según Penrose, esta limitación se aplica a las máquinas o a todo procedimiento mecánico de demostración, pero no se aplica a los seres humanos, ya que no son máquinas y pueden intuir dichas verdades matemáticas. Penrose propone que para escapar de la limitación gödeliana se necesita una nueva física basada en lo que él llama *gravedad cuántica correcta*. Desafortunadamente, nadie sabe exactamente lo que es. La opinión generalizada de la comunidad de expertos en IA es que esta nueva física no es necesaria para lograr inteligencias artificiales como las humanas. Hay otras limitaciones, como las ya mencionadas de la adquisición de conocimientos de sentido común o la corporeidad, que son mucho más importantes y reales.

Weizenbaum o lo que no se debe hacer a pesar de que se pueda

Además de las críticas a lo que con la IA no se puede hacer, también tenemos críticas basadas en lo que, en cualquier caso, nunca se debería hacer. Muchas de estas críticas van dirigidas a aquellas tareas que son inherentemente humanas, como, por ejemplo, la toma de decisiones que requieran tener en cuenta valores humanos. Un

caso muy concreto son las armas autónomas. Una de las personas más activas en contra del uso socialmente irresponsable de la IA fue Joseph Weizenbaum. A mediados de los años sesenta, Weizenbaum programó el sistema Eliza[13] que, sin comprender nada en absoluto el lenguaje natural, era capaz de mantener una conversación en inglés simulando ser un psicoterapeuta. La conversación, sin embargo, pronto se volvía incoherente; aun así, muchos usuarios creían que estaban conversando con un psicoterapeuta humano. Ello sorprendió y escandalizó al propio Weizenbaum y provocó que se planteara cuestiones éticas sobre los peligros de la IA. De hecho, durante muchos años, hasta su muerte en 2008, fue muy crítico con la IA y publicó un interesante e influyente libro sobre ello (Weizenbaum, 1976) donde argumenta que, según diferencias fundamentales entre humanos y máquinas, hay tareas que los ordenadores no deberían hacer aunque técnicamente pudieran. Sus argumentos se basan en el papel que juega el medio social y cultural en el que un ser humano crece, vive y trabaja, adquiriendo experiencias que son distintas a las que podría llegar a adquirir una máquina[14].

La explosión combinatoria

Quizá la mejor forma de aproximarnos a esta idea sea con un ejemplo. Pensemos en el problema del ajedrez: queremos desarrollar un programa de ordenador que juegue bien al ajedrez. Además de conocer las reglas del ajedrez, de modo que solo haga movimientos legales, el programa ha de ser capaz de buscar —entre los millones y millones de secuencias de movimientos posibles— aquella que le lleva a una posición ganadora. El espacio de estados, es decir, de posibles configuraciones de este problema es enorme (del orden de 10^{40}). Formulado de esta manera, este problema sufre de explosión combinatoria porque el número de combinaciones que hay que explorar es gigantesco y no hay ordenadores que lo puedan realizar en un tiempo razonable.

El problema de la explosión combinatoria aparece muy frecuentemente en los problemas de IA, en especial cuando son formulados a partir de las combinaciones de los valores individuales que pueden tomar sus elementos, y se plantean como búsqueda en espacios de estados. Típicamente, esos espacios de estados son muy grandes y encontrar una solución en ellos puede exigir un tiempo muy grande (meses o años de cómputo). Este es el gran problema de la explosión combinatoria, que exige tiempos inasumibles. Incluso si el problema es semidecidible, el algoritmo de IA puede no acabar (no porque esté mal codificado, sino porque incluso haciendo lo correcto no hay forma de terminar; por ejemplo, deducción en lógica de predicados. Un problema es *decidible* cuando existe un algoritmo que *siempre* termina con una solución o, si esta no existe, retorna porque no hay solución. Para algunos problemas este algoritmo no existe —por ejemplo, el problema de la parada de una máquina de Turing—; son de especial interés los problemas *semidecidibles*; para los que existe un

algoritmo que termina cuando encuentra una solución, pero en caso contrario no termina —por ejemplo, comprobar si una fórmula en lógica de predicados se puede satisfacer—).

La explosión combinatoria está claramente relacionada con la complejidad exponencial que presentan muchos algoritmos de IA. Dicha complejidad indica que se ha de realizar un número exponencial (en el tamaño de la entrada) de operaciones para encontrar una solución. Cuando el tamaño de la entrada es pequeño, el número total de operaciones es realizable, pero cuando el tamaño de la entrada crece, el número de operaciones a realizar aumenta exponencialmente y no hay forma de ejecutarlas en un tiempo razonable. Por ejemplo, el problema SAT —encontrar una asignación a las variables de una fórmula proposicional para que la fórmula se evalúe a cierto o demostrar que no existe tal asignación— tiene una complejidad exponencial en el número de variables de la fórmula. Para fórmulas pequeñas (pocas variables) la exponencialidad no es un gran obstáculo y se pueden resolver fácilmente. Pero para fórmulas muy grandes, la exponencialidad es una barrera imposible de franquear, (aunque se han desarrollado algoritmos muy sofisticados para resolver el problema SAT y se han conseguido éxitos muy notables en el tamaño de los problemas que son capaces de resolver; desde los años sesenta hasta ahora, los algoritmos para SAT muestran una evolución drástica, con un aumento impresionante de su capacidad de resolución)**[15]**.

La explosión combinatoria está presente en la formulación exacta de muchos problemas de IA. A continuación discutimos tres estrategias para limitarla o controlarla: aproximaciones, heurísticas y el uso de poda.

Si la formulación exacta de un problema sufre de explosión combinatoria, una formulación (o resolución) aproximada puede que esté libre de ella —o la reduzca— y permita soluciones computacionales temporalmente factibles. Usar aproximaciones o no dependerá de la calidad de los resultados que estas aproximaciones consiguen.

En el capítulo anterior ya hemos hablado de que las heurísticas son criterios (tipo *probar en este orden* o *considerar solo los primeros tres valores*) que generalmente van bien y provocan mejoras en el rendimiento de los algoritmos, pero no hay garantías de que esto sea siempre así. Las heurísticas están justificadas por los resultados experimentales y suelen estar explicadas con argumentos de carácter estadístico. Se pueden aplicar tanto a formulaciones exactas como aproximadas.

Existen muchos ejemplos de heurísticas en IA; uno de ellos aparece en problemas de satisfacción de restricciones, en una formulación exacta cuando se resuelven dichos problemas mediante búsqueda exhaustiva en árbol. El orden de variables puede cambiar de rama a rama y la heurística de dominios mínimos para elegir la siguiente variable en la rama se ha mostrado muy eficaz en una amplia variedad de problemas.

La idea de poda consiste en eliminar zonas del espacio de búsqueda para las que podemos deducir que no contienen soluciones. Esta idea, ya presente en los primeros

programas de IA, se ha ido intensificando dado su impacto en la resolución. El término poda viene de las búsquedas en árbol, en donde ciertos subárboles pueden eliminarse sin perder soluciones. La poda se suele utilizar en formulaciones exactas. Como las heurísticas, esta técnica no es exclusiva de IA: aparece a menudo en otras disciplinas (como en investigación operativa, con sus estrategias *branch-and-prune* o en programación matemática al eliminar simetrías).

Las limitaciones de la representación y el razonamiento basados en la lógica matemática

Las dificultades de tipo técnico frustraron a muchos expertos en IA, lo cual motivó la búsqueda de paradigmas alternativos a los que se usaban en la IA simbólica, que era en aquellos momentos el modelo dominante de la IA. Uno de los pilares de la IA simbólica es la lógica matemática, pues se usa intensamente para representar conocimientos y razonar a partir de ellos de acuerdo con las reglas de inferencia de la lógica clásica, tanto proposicional como de primer grado. Uno de los primeros en cuestionar la lógica matemática en IA fue Drew McDermott, profesor en Yale. Sus críticas tuvieron un considerable impacto, ya que era un prestigioso experto en IA que había contribuido significativamente al desarrollo de la IA basada precisamente en la lógica. En un artículo publicado en 1987, “A critique of puré reason”, afirmó que la hipótesis de que el razonamiento humano puede ser modelado mediante inferencia deductiva, o incluso aproximadamente deductiva, era errónea. Concretamente afirmó que cuanta más deducción lógica busca uno en el razonamiento humano, menos deducción encuentra. En lugar de deducción, lo que uno encuentra es que muchas de las inferencias que inicialmente parecen deductivas de hecho contienen componentes no deductivos. Uno de los argumentos que da es el siguiente:

Piensa en la última vez que planificaste algo y pregúntate si serías capaz de haber demostrado que el plan trazado era correcto; pues bien, seguramente podrías fácilmente enumerar diez circunstancias plausibles que hubieran hecho fracasar el plan, sin embargo seguiste adelante con el plan previsto.

El artículo fue publicado junto con una serie de comentarios a favor y en contra que provocaron una interesante controversia acerca del rol de la lógica en la IA, controversia que motivó el desarrollo de nuevos paradigmas que extienden la lógica clásica, como, por ejemplo, las lógicas no monótonas, las lógicas capaces de representar la incertidumbre y la imprecisión, y las lógicas descriptivas (daremos detalles de las dos últimas en el próximo capítulo). También se desarrollaron otros paradigmas inspirados en la lógica clásica, como, por ejemplo, las redes semánticas (Bobrow y Collins, 1975). Finalmente, también se propusieron otros que se apartaban completamente de la lógica matemática, como, por ejemplo, los paradigmas conexionistas, evolutivos y corpóreos. De hecho, uno de los aspectos centrales de esta

controversia sobre la capacidad o no de la lógica matemática para modelar el razonamiento humano es, una vez más, el problema de representar conocimientos de sentido común y razonar a partir de ellos. En el caso de las lógicas no monótonas, se pretende poder retractarse de una conclusión previamente deducida a partir de un conjunto de hechos conocido, basándose en el conocimiento de un hecho adicional que contradiga el conocimiento previo. Esta capacidad, tan común en el razonamiento humano, se muestra difícil de mecanizar. En cuanto a la representación de la incertidumbre, la objeción a la lógica clásica es que los hechos representados tienen que ser o bien verdaderos o bien falsos, sin posibilidad de valores de verdad intermedios, mientras que la inmensa mayoría del conocimiento y razonamiento humano es de naturaleza incierta y, sin embargo, somos capaces de llegar a conclusiones y tomar decisiones. Este hecho se tuvo en cuenta desde muy pronto en el contexto de los sistemas expertos (por ejemplo, tanto Mycin como Prospector permitían representar conocimiento incierto)**[16]**.

A lo largo de toda la historia de la IA se ha seguido investigando en representaciones basadas en la lógica y actualmente, gracias a las extensiones y mejoras introducidas, sigue siendo uno de los campos más activos dentro del ámbito de la representación del conocimiento en IA, como veremos en el siguiente capítulo.

Capítulo 4
Una nueva primavera: nuevas aproximaciones a la inteligencia artificial

En el capítulo anterior hemos hablado de las dificultades tanto de carácter ético como filosófico y técnico a las que la IA se enfrentó. En este capítulo hablaremos del resurgimiento de la IA con la aparición de nuevas e importantes aproximaciones en todas sus áreas, desde la representación de conocimientos y el razonamiento hasta los sistemas multiagente, pasando por el aprendizaje, la visión y la robótica. En todas estas áreas hubo avances significativos que sentaron las bases que permitieron alcanzar los grandes éxitos de los que hablaremos en el capítulo 6.

La lógica difusa

Tal y como hemos mencionado en el capítulo anterior, una de las limitaciones de la lógica clásica es la dificultad para representar conocimientos inciertos e imprecisos, ya que las proposiciones en lógica clásica deben ser completamente ciertas o completamente falsas. Sin embargo, es un hecho que la mayoría de conocimientos que los seres humanos manejamos están plagados de incertidumbre e imprecisión. Cuando dudamos de la validez de un hecho se trata de incertidumbre y cuando tenemos dificultades para expresar un hecho con claridad se trata de imprecisión. A menudo los conocimientos que manejamos son simultáneamente inciertos e imprecisos. Veamos con un sencillo ejemplo esta diferencia. Supongamos que disponemos de la información imprecisa: “La temperatura del reactor es superior a 200 grados”; dado este conocimiento previo, la afirmación “la temperatura del reactor es de 300 grados” es precisa pero incierta. Por otra parte, si el conocimiento previo fuera “la temperatura es de 250 grados” entonces la afirmación de que “la temperatura es de 300” grados es claramente falsa. Es decir, que en este último caso no hay incertidumbre, ya que siempre podremos determinar si una afirmación acerca de la temperatura es falsa o verdadera. Sin embargo, si el conocimiento previo contiene términos con significado vago, como, por ejemplo, “la temperatura del reactor es alta”, entonces la afirmación de que “la temperatura es de 600 grados” es incierta y lo único que podremos decir es hasta qué punto esta temperatura de 600 grados es compatible con el conocimiento previo de que la temperatura es alta. Este concepto es similar al concepto de verdad en la teoría de correspondencia de la verdad de Russell y Wittgenstein (Newman, 2007). La teoría matemática que permite

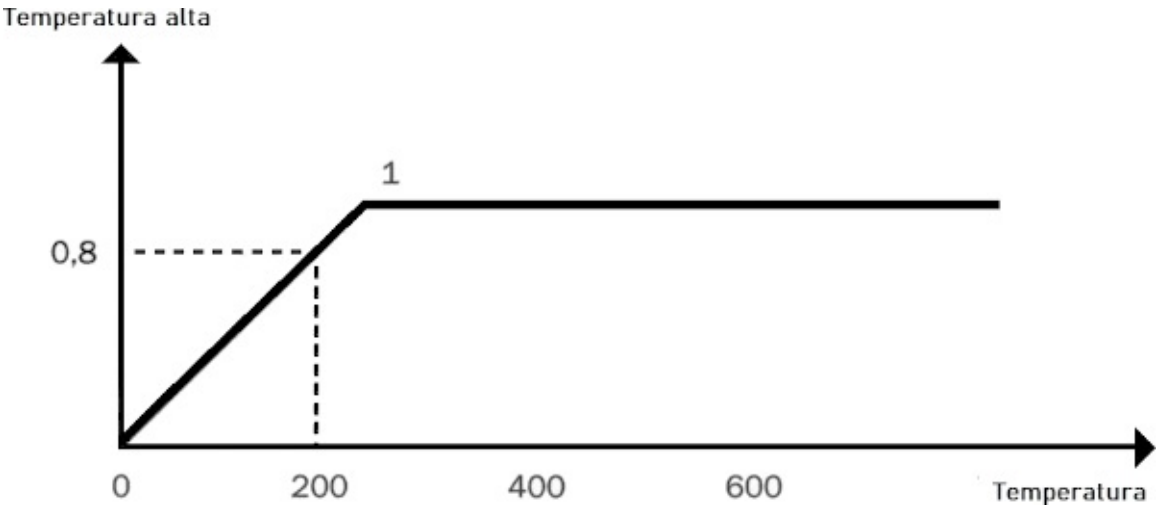
modelar tanto los conocimientos imprecisos como inciertos es la teoría de los *conjuntos difusos* (*fuzzy sets*) propuesta por Lotfi Zadeh en 1965 (Zadeh, 1965).

El concepto de conjunto difuso

Consideremos la proposición *X es A*, donde *X* es una variable (por ejemplo, *TEMPERATURA*) que toma valores en un universo *U* (por ejemplo, el intervalo de 0 a 1.000 grados) y *A* es un predicado difuso (por ejemplo, *ALTA*). Este predicado denota un subconjunto difuso de *U* representado mediante una función de pertenencia que a cada valor de temperatura le asigna su valor de pertenencia al conjunto difuso de temperaturas altas. En la figura 5 podemos ver una posible función de pertenencia del conjunto difuso *TEMPERATURA ALTA* de forma que una temperatura de 600 grados pertenece a dicho conjunto difuso con un grado igual a 0,8. La función de pertenencia es una extensión de la función característica de la teoría clásica de conjuntos. Un función característica asigna únicamente los valores 1 o 0 a cada uno de los elementos del universo *U* dependiendo de si pertenece o no al conjunto mientras que en los conjuntos difusos la función de pertenencia asigna cualquier valor dentro del intervalo [0, 1] a los elementos de *U*.

De forma similar a los conjuntos clásicos, en la teoría de conjuntos difusos se definen operaciones de complementación (o negación), intersección y unión (López de Mántaras, 1991) que extienden las correspondientes de los conjuntos clásicos. También se define el concepto de inclusión entre conjuntos difusos.

FIGURA 5
Función de pertenencia y grado de certeza.



De los conjuntos difusos a la lógica difusa

La lógica difusa proporciona una metodología para representar variables lingüísticas, es decir, variables cuyos valores no están limitados a ser numéricos, sino que pueden ser expresiones lingüísticas. Por ejemplo, la variable *ALTO*, en el contexto de las personas, además de numérica (tomando valores desde 0 a 230 cm) también puede ser vista como variable lingüística con los valores: *MUY BAJO, BAJO, MEDIANO, ALTO, MUY ALTO*. Cada uno de estos valores se puede interpretar como un conjunto difuso sobre el universo $U = [0, 230]$. Valores más complejos como *ni muy alto ni muy bajo* se pueden generar mediante las operaciones lógicas de conjunción, unión y negación que extienden los de la lógica clásica exactamente igual que los operadores sobre conjuntos difusos mencionados anteriormente. Veamos con un ejemplo la relación entre función de pertenencia en conjuntos difusos y valor de verdad en lógica difusa. Supongamos que Juan mide 175 cm y que a esta talla le corresponde un valor de pertenencia de 0,7 al conjunto de personas altas; pues bien, decimos que la proposición "Juan es alto" tiene un valor de verdad de 0,7. De la misma forma, en el ejemplo de la temperatura del reactor, 0,8 es el valor de verdad de la proposición "temperatura alta" cuando la temperatura es de 600 grados. Es decir, que el grado de pertenencia de un elemento *X* al conjunto difuso *A* es el valor de verdad de la proposición *X es A*.

Otra extensión importante de la lógica difusa en relación con la clásica es que en esta última solamente disponemos de dos cuantificadores (*EXISTE y PARA TODO*), mientras que en lógica difusa se puede definir un número amplio de cuantificadores difusos como, por ejemplo, *VARIOS, POCOS, LA MAYORÍA, ALGUNOS*, etc. Estos cuantificadores se representan mediante números difusos que corresponden a caracterizaciones imprecisas de la cardinalidad de un conjunto.

La inferencia en lógica difusa

En la lógica difusa se generaliza la noción fundamental de la lógica clásica para llevar a cabo inferencias mediante el concepto de *modus ponens generalizado* (MPG). Dada la proposición condicional (regla) *Si X es A entonces es B*, donde como antes *A* y *B* son conjuntos difusos definidos mediante sus respectivas funciones de pertenencia, y dada la proposición *X es A**, donde *A** es también un conjunto difuso definido mediante su función de pertenencia, entones el MPG permite concluir que *Y es B**, de tal forma que la función de pertenencia del conjunto difuso *B** se calcula en función de las funciones de pertenencia *A*, *B* y *A**, aplicando un operador de implicación que a su vez se define a partir de una combinación de operadores de negación, intersección y unión (López de Mántaras, 1991). Lo realmente interesante del MPG es que, contrariamente a la lógica clásica, la premisa *X es A* no* tiene por qué ser idéntica al antecedente *X es A* de la regla para poder aplicar el MPG. Cuanto

más próximo es *A** de *A* más próximo es *B** de *B*. Vejamos un ejemplo de inferencia usando el MPG.

Dado un universo de objetos, dada la regla *Si* PRECIO *es* BARATO *entonces* DURACIÓN *es* CORTA, dada la función de pertenencia representando, en el contexto de dichos objetos, al conjunto difuso BARATO y dada la función de pertenencia representando, en el mismo contexto, al conjunto difuso DURACIÓN de la vida útil del objeto, entonces mediante el MPG podemos calcular la función de pertenencia asociada a la duración imprecisa de un objeto partiendo de información también imprecisa sobre su valor. Por ejemplo, si la información es PRECIO *es* BASTANTE BARATO entonces con el MPG deduciríamos que DURACIÓN es *bastante corta*. Sin embargo, según las propiedades del MPG, no se puede deducir que la duración es MUY CORTA a partir de la información de que el objeto es MUY BARATO; es decir, que no es posible deducir nada que sea más preciso que lo indicado en la conclusión de la regla.

Existen diversas formas de definir los operadores de implicación, interacción y unión, y escogiendo una combinación correcta de operadores se obtiene que en el caso particular en que $A = A^*$ tendremos $B^* = B$, es decir, que el *modus ponens clásico es* un caso particular del MPG. En el ejemplo anterior tendríamos, pues, que si el objeto es BARATO entonces el MPG, igual que con el *modus ponens clásico*, concluye que la duración es CORTA.

La teoría de la posibilidad

La interpretación del concepto de función de pertenencia en tanto que grado de certeza da lugar a la teoría de la posibilidad. En el ejemplo de la temperatura del reactor diríamos que 0,8 es el grado de posibilidad de que la temperatura es de 600 grados sabiendo que la temperatura es alta. Es decir, sabiendo que la temperatura es ALTA, para cada una de las temperaturas *X* del universo *U* podemos obtener su correspondiente grado de posibilidad. Por lo tanto, dado un predicado difuso *A*, representado mediante una función de pertenencia, y dada una variable *X*, la distribución de posibilidad es el conjunto de posibles valores de *X* dado *A* y se define mediante la función de pertenencia. De hecho, la teoría de la posibilidad es un caso particular de la lógica difusa, ya que sirve para evaluar el grado de posibilidad de una proposición precisa (por ejemplo, *la temperatura es de 600 grados*) dada una información imprecisa (por ejemplo, *la temperatura es* ALTA). También permite definir una medida de certeza como el grado de imposibilidad de proposición contraria, es decir, en el ejemplo anterior, de la proposición *la temperatura no es 600 grados*. En este caso, el grado de certeza sería de $1 - 0{,}8 = 0{,}2$.

En el caso general de la lógica difusa, el resultado de evaluar valores de verdad de proposiciones imprecisas dadas informaciones previas vagas resulta en conjuntos difusos sobre [0, 1]. Estos conjuntos difusos se llaman *valores de verdad difusos* y

permiten integrar tanto la imprecisión como la incertidumbre dentro del mismo marco teórico.

Tanto la lógica difusa como la teoría de la posibilidad han sido extensivamente aplicadas a la IA y en particular al desarrollo de sistemas expertos en dominios en los que el conocimiento y la información son imprecisos e inciertos.

Lógicas de la descripción

Las *lógicas de la descripción* (*description logics*, DL) son formalismos de representación de conocimiento basados en lógica de predicados (Baader *et al*., 2003). Fueron motivadas por la necesidad de realizar razonamiento eficiente dentro de un sistema formal. Se pueden ver como fragmentos decidibles de la lógica de predicados. Históricamente, aparecieron para dotar de fundamentos lógicos a constructos tales como redes semánticas y *frames*, elementos que habían aparecido en el área de representación del conocimiento con un punto de vista exclusivamente procedural. En este sentido, el sistema KL-ONE se puede considerar como el primero basado en DL. A este sistema se sucedieron muchos otros, con aplicaciones a las ontologías y web semántica, como veremos más adelante. Hay diversos tipos de DL, en función de su expresividad y de las propiedades que permiten comprobar. DL diferencia entre conceptos (equivalentes a predicados unarios) y roles (equivalentes a predicados binarios), permite expresiones entre conceptos y entre roles, y dispone de conceptos especiales y axiomas. Todo ello posibilita describir los elementos y las relaciones existentes sobre una base de conocimientos de forma mucho más fácil y amigable que en lógica de predicados.

Un sistema expresado en DL se divide en dos partes: la A-box y la T-box. La A-box contiene los elementos del sistema, mientras que la T-box contiene las relaciones entre esos elementos. La inferencia se realiza aplicando las relaciones de la T-box sobre los contenidos de la A-box. Este formalismo proporciona razonamientos básicos, por ejemplo, encontrar todos los elementos individuales de un concepto, o todos los conceptos de un elemento, o si un concepto es más general o más específico que otro, etc. La combinación de una T-box con una A-box constituye una base de conocimientos (KB). El formalismo de DL se ha utilizado en el desarrollo de ontologías, por ejemplo, en el lenguaje OWL para realizar ontologías en la web. La existencia de razonadores para DL, como RACER o FaCT, permite ofrecer servicios para sistemas basados en estas ontologías.

Aunque se desarrollaron de forma independiente, existe una relación cercana entre DL y la lógica modal. En general, un concepto corresponde a una variable proposicional modal y un rol corresponde a un operador modal con oso rol como su relación de accesibilidad.

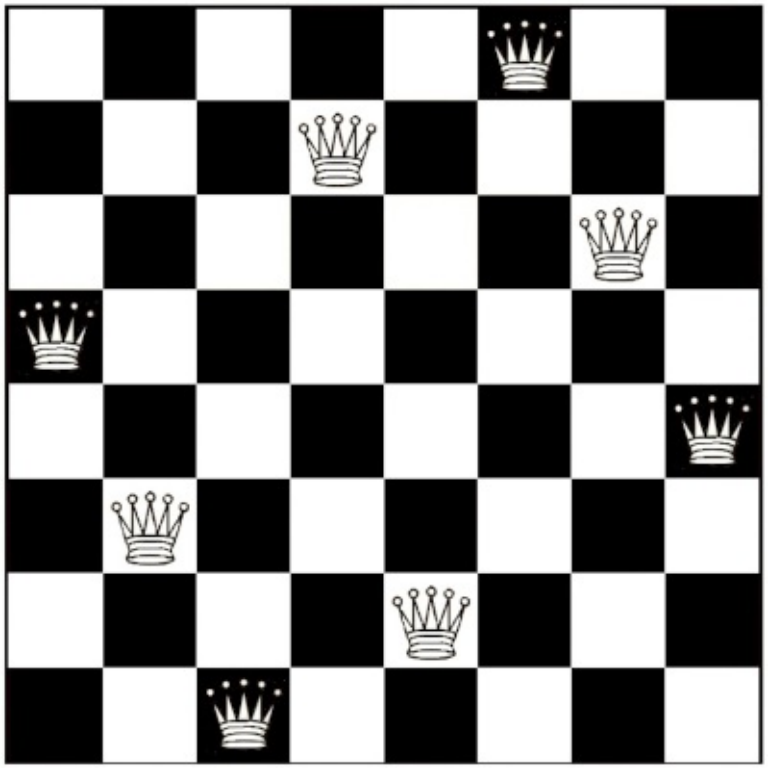
Las aplicaciones de las DL incluyen diversos campos. La web semántica es una de las áreas donde más aplicación tienen las técnicas de DL, en la parte de desarrollo

de ontologías y su explotación para realizar inferencias. También se han utilizado para aplicaciones de configuración, información sobre paquetes de *software* y bases de datos (Russell y Norvig, 2010).

Razonamiento basado en restricciones

Un modelo que ha tenido mucho éxito en la IA desde los años setenta ha sido el razonamiento basado en restricciones. La idea básica de un *problema de satisfacción de restricciones* (Constraint Satisfaction Problem, CSP) es plantear unas variables sobre conjuntos finitos de valores que han de satisfacer un conjunto de restricciones. Como el conjunto de variables es finito, el problema se puede resolver con un algoritmo de búsqueda en profundidad que recorre el árbol de búsqueda del problema, verificando que las restricciones se cumplen en cada nodo. Este algoritmo se denomina *backtracking* y es ampliamente conocido por la comunidad de IA. Se trata de un problema NP-Completo. Un ejemplo de este tipo de problemas es el famoso problema de las 8 reinas, que aparece en la figura 6.

Figura 6
El problema de las 8 reinas.



Un CSP se puede tratar con métodos que aumentan el nivel de consistencia local (consistencia de subproblemas formados por una, dos, tres… variables). Este

tratamiento origina que: 1) se eliminen valores que no aparecerán en ninguna solución global, y 2) si al eliminar valores una variable se queda sin valores posibles significa que el problema no tiene solución. Originalmente, se realizaban estos tratamientos como un preproceso anterior al *backtracking*. Pero se han encontrado formas de incluir estos métodos en un esquema de *backtracking* generando algoritmos que mantienen la consistencia local en el subproblema que consideran en cada momento. Estas estrategias han sido claves para lograr esquemas algorítmicos eficientes para resolver CSP. La *arco consistencia* es el grado de consistencia local comúnmente aceptado como el que proporciona una mejor relación coste-beneficio.

La modelización de este tipo de problemas ha sido un elemento crucial para la popularización de esta tecnología. En primer lugar, el uso de *restricciones globales* que vinculan a más de dos variables (en un contexto inicialmente dominado por la visión binaria) para las que se puede explotar su semántica de cara a implementar arco consistencia de forma eficiente. La utilización de varios modelos redundantes en la resolución de un problema, conectados por las llamadas *chanelling constraints*, han permitido podas extras de valores que no se obtendrían utilizando cada modelo por separado. En este tipo de problemas suelen existir *simetrías*. Una explotación adecuada de ellas permite simplificar la búsqueda, generando importantes ahorros computacionales.

De forma alternativa a la búsqueda, se han planteado métodos de resolución (inferencia completa) basados en operaciones sobre restricciones. Se conocen con el nombre genérico de *eliminación de variables* (Dechter, 2003). Aunque suelen ser superados por los métodos basados en búsqueda, sobre algunos problemas han dado excelentes resultados.

Una visión alternativa a la restricción *dura* —o de cumplimiento obligatorio— es la restricción *blanda* —que debe cumplirse siempre que sea posible, pero no invalida una solución si esta no la cumple—. Los problemas de satisfacción de restricciones devienen problemas de optimización —satisfacer todas las restricciones duras y optimizar el cumplimiento de las restricciones blandas— que se suelen atacar con procedimientos tipo *ramificación y poda (branch and bound)*. Se han desarrollado métodos de *consistencia local blanda* que, como en el caso clásico, permiten eliminar subconjuntos de valores que no estarán en ninguna solución. Se pueden combinar con los métodos de búsqueda.

Otra visión novedosa es la de *restricciones distribuidas*. Las aproximaciones anteriores suponen que el problema está centralizado en un ordenador, pero existen escenarios en donde el problema está distribuido entre varios agentes, y no se puede reunir en un único agente (en muchos casos por razones de confidencialidad). Se han desarrollado algoritmos para alcanzar la solución mediante paso de mensajes, garantizando la privacidad de los agentes.

El razonamiento con restricciones ha generado un estilo nuevo de programación declarativa. Utilizando un entorno de programación con restricciones —hay varios

disponibles en Internet—, se trata de declarar las variables, los posibles valores que pueden tomar y las restricciones que han de cumplir. Los elementos del lenguaje de programación (secuencia, condicionales, iteradores) se utilizan para expresar restricciones complejas. Las variables son asignadas por el resolvedor de restricciones, un código que interpreta/compila el programa que se le pasa como entrada.

El razonamiento con restriciones[17] tiene fuertes conexiones con la satisfactibilidad booleana (el problema SAT se puede ver como un caso particular de CSP), investigación operativa (en particular con programación entera), rutas dinámicas de vehículos, planificación, etc.

Nuevas estrategias de búsqueda

A pesar de las fuertes limitaciones creadas por la explosión combinatoria, la comunidad de IA ha seguido mejorando los métodos de búsqueda para poder resolver problemas de mayor tamaño. En este sentido, mencionamos una serie de ideas que han surgido en los últimos 30 años y han demostrado su utilidad en IA. Dividimos la sección en dos partes dedicadas a búsqueda sistemática y búsqueda local.

La búsqueda sistemática considera las estrategias que se basan en una enumeración sistemática del espacio de estados; produce algoritmos completos que obtienen soluciones óptimas. El ejemplo clásico es el algoritmo)A*[18]. Un esquema diferente pero aplicable a problemas con ramas no infinitas es *ramificación y poda*. Trabaja con cotas inferiores y superiores del coste de una solución y realiza poda cuando estas cotas se alcanzan.

Dentro del esquema de búsqueda de A*, se ha mejorado la calidad de las heurísticas admisibles con las denominadas *bases de datos de patrones*. Se trata de bases de datos que almacenan buenas estimaciones (en el límite exactas) de la heurística para configuraciones parciales de los estados. Ello mejora mucho el rendimiento de A*. Estas bases de datos se calculan en una fase de preproceso, fuera de la búsqueda propiamente dicha. Esta idea se aplica a problemas que se han de resolver de forma repetida, como juegos o puzzles, de forma que se amortiza el esfuerzo de calcularla en muchas búsquedas. Otra idea que mejora la efectividad de A* es la *búsqueda frontera*. Es un ingenioso método para no guardar la lista de nodos ya expandidos por A* que, en la práctica, llega a doblar la capacidad de memoria del algoritmo en los problemas que resuelve.

Dado el alto consumo de memoria del A*, se han desarrollado otros algoritmos que también alcanzan soluciones óptimas con un consumo lineal de memoria. Uno de ellos, denominado de *profundización iterativa,* realiza búsquedas en profundidad con profundidad acotada que se incrementa en cada iteración (la primera vez que se resolvió el cubo de Rubik por ordenador fue utilizando este algoritmo). Un esquema diferente pero aplicable a problemas con ramas no infinitas es *ramificación* y *poda*.

Trabaja con cotas inferiores y superiores del coste de una solución y realiza poda cuando estas cotas se alcanzan.

Una visión alternativa es la *búsqueda con discrepancias*. Consideramos un problema donde las soluciones están en un determinado nivel en el árbol, que se recorre con un esquema en profundidad. En este escenario, el papel de la heurística es disponer los nodos más prometedores a la izquierda (el recorrido del árbol es de izquierda a derecha y en cuanto se encuentra una solución, el algoritmo termina). Este nuevo esquema de búsqueda sugiere visitar las hojas del árbol por número de discrepancias creciente, en donde una discrepancia es no seguir la heurística en un nodo.

Todas estas aproximaciones se basan en el supuesto de que primero se calcula la solución y después esta se ejecuta, sin que ello pueda invalidar la solución. Sin embargo, en entornos no controlados, la ejecución de una solución puede ser inviable (por ejemplo, cuando los obstáculos cambian de posición en un problema de encontrar caminos en un laberinto). Cuando se tiene en cuenta la ejecución de la solución, nos encontramos ante un problema de *búsqueda en línea* (como contrapunto a la *búsqueda fuera de linea*, el tipo de búsqueda explicada anteriormente). En este caso, es de interés mencionar la *búsqueda en tiempo real* que realizan los algoritmos RTA* y LRTA*, y la *búsqueda incremental* mediante los algoritmos D* o D*-lite (alguna de estas técnicas se han empleado en robots de exploración espacial)**[19]**.

La búsqueda local toma decisiones en el entorno local del estado actual, decisiones que no necesariamente han de ser óptimas en una visión global del problema. Se basa en la optimización de una *función objetivo* —dependiente del problema— que se realiza en la *vecindad* del estado actual, moviéndose hacia mínimos locales que no necesariamente son mínimos globales. A menudo incluye elementos estocásticos o aleatorios. Las *condiciones de terminación* se basan en algún límite (tiempo, máximo número de iteraciones, ratio de mejora de la solución). Las estrategias de búsqueda local no son completas —puede existir una solución y no encontrarla—, aunque escalan mejor que la búsqueda sistemática y permiten atacar problemas de mayor tamaño**[20]**. A continuación, enumeramos brevemente unos pocos metaheurísticos populares en la comunidad de búsqueda:

- *Descenso por gradiente* (*hill-climbing* o *basic local search*). Consiste en moverse a un estado vecino con menor valor de la función objetivo que el estado actual. Esta técnica presenta el problema de convergencia a mínimos locales; buena parte del trabajo en metaheurísticas se ha centrado en cómo escapar de estos mínimos locales.
- *Enfriamiento simulado (simulated annealing).* Inspirado en la mecánica estadística, esta metaheurística está gobernada por un parámetro que juega el papel de la temperatura en los sistemas físicos (en donde la función a optimizar es la energía). Con probabilidad 1, acepta movimientos a estados

donde el valor de la función objetivo disminuye, pero acepta con probabilidad distinta de cero movimientos a estados donde la función objetivo aumenta, lo que le permite escapar de mínimos locales. Estos movimientos son más probables a temperatura alta que a temperatura baja. Se exige un esquema lento de bajada de temperatura hasta el valor cero, en donde no se aceptan cambios a estados con mayor valor de la función. En el límite, se demuestra que la solución tiende a un mínimo global.

- *Búsqueda tabú* (*tabu search*). Esta estrategia realiza un descenso por gradiente, en donde los estados visitados recientemente se guardan en la denominada lista tabú y están prohibidos. La vecindad del estado actual está limitada a los vecinos que no están en la lista tabú. Ello favorece la exploración de otras áreas del espacio. El mejor estado en cada iteración entra en la lista tabú, que libera al estado más antiguo. El tamaño de la lista tabú indica concentración en pequeñas zonas (tamaño corto) o diversificación en grandes áreas (tamaño largo) y puede variar durante el proceso de búsqueda.
- *Computación evolutiva (evolutionary computation, genetic algorithms).* Inspirados en la evolución de las especies biológicas, estos algoritmos trabajan con una población de soluciones. En cada iteración hay un proceso de selección, combinación y mutación entre estas soluciones, de forma que generan una nueva población. Solo las soluciones más aptas son elegidas para formar la siguiente generación. El proceso termina al alcanzar un cierto límite máximo de generaciones. Hay gran variedad de estrategias en la codificación de las soluciones, así como en la implementación de los distintos operadores.

Tanto en búsqueda sistemática como en búsqueda local, la cantidad de contribuciones relevantes es grande[21].

Redes bayesianas

A pesar del innegable interés de la lógica difusa como modelo de representación de la incertidumbre y la imprecisión, la mayoría de investigadores en IA se han inclinado más hacia la teoría de la probabilidad como modelo para representar dichos conocimientos y, en particular, los de naturaleza incierta. En este modelo, se usa la inferencia bayesiana como base para el razonamiento a partir de la representación probabilística de dichos conocimientos. Ya mencionamos en el capítulo 2 el uso de factores de certeza en Mycin y de relaciones de verosimilitud probabilística o inferencia bayesiana en Prospector para modelar el razonamiento aproximado en sistemas expertos. Sin embargo, estos modelos de razonamiento aproximado no tenían en cuenta las importantes interdependencias entre las probabilidades asociadas a aquellos conocimientos que están relacionados entre sí, con el fin de simplificar los cálculos y evitar la explosión combinatoria cuando el número de conocimientos en el

sistema es grande. El problema reside en que, dado un conjunto de conocimientos representados mediante proposiciones, no es suficiente tener en cuenta las probabilidades individuales de cada proposición, sino que también se necesita tener en cuenta las probabilidades conjuntas de las combinaciones de dichas proposiciones, lo cual conduce a una explosión combinatoria del número de probabilidades a tener en cuenta. Con el fin de evitar este problema, sin tener que llevar a cabo aproximaciones basadas en suposiciones de independencia no realistas, en los años ochenta se propuso un método para representar dependencias de forma eficiente. La clave consiste en que, en general, no es necesario considerar todas las posibles combinaciones en una gran probabilidad conjunta, ya que la mayoría de las proposiciones son condicionalmente independientes unas de otras, por lo que no es necesario considerar sus interacciones. Es suficiente representar de forma modular aquellas proposiciones que interactúan entre ellas. Estas dependencias se representan de forma gráfica mediante una red llamada *red bayesiana (bayesian network)* (Pearl, 1988). Como ejemplo, supongamos que tenemos los siguientes conocimientos representados por las proposiciones:

A: El motor de arranque funciona.

B: El motor de arranque hace girar el motor del coche.

C: El sistema de alimentación de gasolina funciona.

D: El coche arranca.

Si todas estas proposiciones representaran eventos independientes, entonces la probabilidad conjunta sería el producto de las cuatro probabilidades individuales:

$$P\ (A, B, G\ D) = P\ (A)\ P\ (B)\ P\ (C)\ P\ (D)$$

Con lo que es suficiente conocer el valor de solamente las cuatro probabilidades individuales. En el otro extremo en que todo interaccione con todo necesitaríamos conocer $2^4 = 16$ probabilidades distintas para poder modelar este sistema correspondiente a las 16 combinaciones:

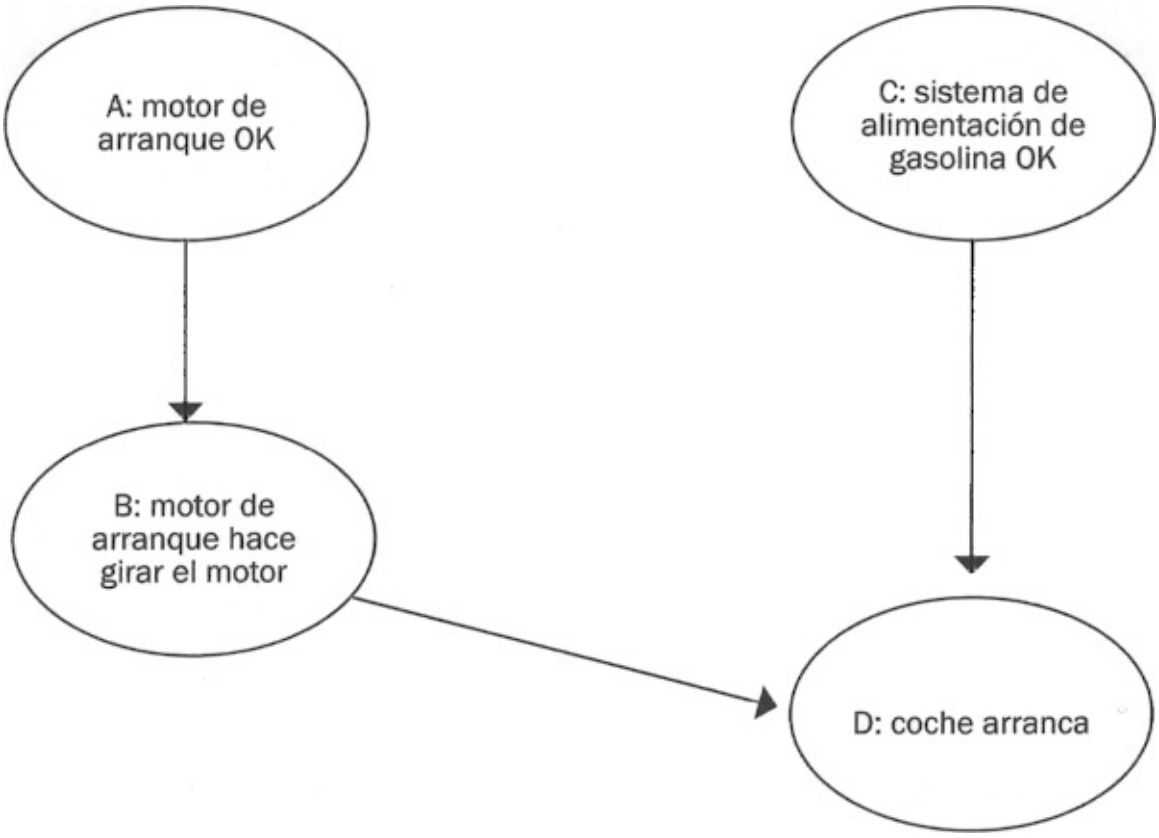
$$P\ (A, B, C, D),\ P\ (A, B, C\ \neg D),\ P\ (A, B, \neg C, D),\ P\ (A, B, \neg C, \neg D)\ldots\ y\ P\ (—A, \neg B, \neg C, \neg D)$$

Es decir, la probabilidad de que *A, B, C, D* sean los cuatro verdaderos, la probabilidad de que *A, B* y *C* sean verdaderos y *D* falso, la probabilidad de que *A* y *B* sean verdaderos y *C* y *D* falsos, etc. Sin embargo, no es necesario considerar todas estas combinaciones, ya que no es difícil observar que *D* depende de *A, B* y *C* y que *B* depende de *A*. Esto conduce a un *grafo de dependencias* (figura 7), donde los nodos representan a las proposiciones *A, B, C y D* y los arcos dirigidos representan las dependencias condicionales entre los nodos que unen, de forma que la dirección del arco va del nodo que condiciona al nodo condicionado. A cada uno de los arcos le corresponde un valor de probabilidad condicional y a los nodos a los que no llega

ningún arco les corresponde un valor de probabilidad incondicional. En la figura 7, las probabilidades necesarias son las probabilidades condicionales: *P (D/B, C), P (D/B, ¬C), P (D/¬B, C), P (D/¬B, −C), P (B/A), P (B/¬A),* y las probabilidades incondicionales: *P (A), P (C).* Es decir, necesitamos la mitad de las probabilidades en el caso extremo en el que no explotáramos el conocimiento de cuáles son las variables que dependen condicionalmente unas de otras. En este caso, donde solamente tenemos 4 proposiciones, el ahorro parece modesto, pero cuando el número de proposiciones aumenta, el número de combinaciones sin explotar las relaciones de dependencia condicional crece exponencialmente. Por ejemplo, en el caso de 20 variables hay 2^{20} combinaciones, es decir, más de un millón. Si de hecho únicamente unas pocas proposiciones tienen relaciones de dependencia condicional, entonces el factor de reducción puede ser de muchos órdenes de magnitud. Obviamente, para poder decidir qué proposiciones son condicionalmente dependientes unas de otras y cuáles son los valores de sus probabilidades es necesario un conocimiento experto sobre el dominio de aplicación que se intenta modelar, o bien aplicar métodos de aprendizaje**[22]**. Una vez el conocimiento está representado mediante un grafo (o red) de este tipo, la inferencia bayesiana se usa para poder calcular cómo las probabilidades de algunos de los nodos se ven afectadas por las probabilidades de otros nodos (Pearl, 1988). Por ejemplo, si estamos seguros de que tanto el motor de arranque como el sistema de alimentación de gasolina funcionan (es decir, que *P (A) = 1* y *P (C) = 1)* entonces podemos inferir el valor de probabilidad de que el coche arranque con base en el conocimiento de las probabilidades condicionales entre *A* y *B, B* y *D,* y *C* y *D*. Inversamente, a partir del conocimiento de que el coche no arranca *(P (D) = 0),* se pueden calcular las probabilidades *P (A)* de que el motor de arranque funcione y *P (C)* de que el sistema de alimentación de gasolina funcione.

Aquí es interesante observar que la probabilidad de que el coche arranque *(P (D)* no depende de la probabilidad de que el motor de arranque funcione *(P (A)* si de entrada ya se conoce la probabilidad de que el motor de arranque haga girar el motor del coche cuando accionemos la llave de contacto *(P (B))* y la probabilidad de que el sistema de alimentación de gasolina funcione (PfQ). Dicho de otra manera, el conocimiento de *P (A)* no nos dice nada adicional sobre *P (D)* si ya conocemos *P (B)* y *P (C).* Técnicamente decimos que *P (D)* es condicionalmente independiente de *P (A),* dados *P (B)* y *P (C).* Tener en cuenta estas independencias condicionales reduce la complejidad del razonamiento probabilístico. Este segundo tipo de inferencia de efectos hacia causas es el que se usa principalmente en sistemas de diagnóstico médico.

FIGURA 7
Red bayesiana.



A pesar de que este método puede reducir significativamente el número de probabilidades a considerar, el número restante puede ser todavía excesivamente grande para que el problema sea tratable computacionalmente. Afortunadamente, en la literatura existen métodos que permiten reducir todavía más el número de probabilidades necesarias a costa de sacrificar exactitud en la inferencia y realizarlas de manera aproximada. Estos métodos se basan en herramientas matemáticas sofisticadas (Russell y Norvig, 2010). De hecho existen redes bayesianas con centenares de nodos y miles de arcos que se han aplicado con éxito a la biología, la medicina, el tratamiento de imágenes, etc. Estas redes grandes no se construyen generalmente de forma manual, sino que se derivan automáticamente mediante datos usando algoritmos de aprendizaje[23].

Planificación

La planificación independiente del dominio se planteó desde sus comienzos como una búsqueda en el espacio de estados del problema (*space-based planning*) del que el sistema STRIPS sería un buen ejemplo[24]. Debido a los problemas que se generaban con la interacción entre subobjetivos, el paradigma dominante pasó a ser la

planificación de orden parcial (*partial-order planning*) que conseguía resolver estas interacciones. En este paradigma se buscaba en el espacio de los planes, hasta conseguir un plan sin amenazas que realizara todos los objetivos desde el estado inicial. A mediados de los años noventa, apareció el planificador Graphplan, que revolucionó el area. Tras él, vino una serie de planificadores con buenas prestaciones basados en búsqueda heurística en el espacio de estados del problema (es decir, bajo el primer paradigma). A continuación presentamos brevemente los planificadores Graphplan, FF, FD y LAMA, como muestra representativa del desarrollo de la planificación en los últimos 20 años.

El planificador Graphplan de Blum y Furst, en 1997, se basa en el análisis del problema relajado, obtenido a partir del problema original eliminando las listas de predicados a borrar (*delete lists* en la nomenclatura de STRIPS). A partir de una búsqueda hacia delante, construye un grafo en el que se alternan niveles de acciones con niveles de predicados, considerando en cada nivel de acciones todas las acciones que se pueden aplicar dados los predicados alcanzables en el nivel anterior. El primer nivel de predicados lo constituye el estado inicial. Con el concepto de *mutex* conceptualiza aquellas acciones o predicados que son incompatibles entre sí. Cuando todos los objetivos (estado final) aparecen en el mismo nivel y no son *mutex*) lanza una búsqueda hacia atrás, considerando aquellas acciones que satisfacen los objetivos finales, no son *mutex* ni sus precondiciones son *mutex*. A medida que esta búsqueda se realiza, las precondiciones de las acciones que entran en el plan se consideran nuevos subobjetivos. Se construye el plan cuando se alcanza el estado inicial, de forma que se satisfacen los objetivos finales. El algoritmo de Graphplan es correcto y completo: los planes retornados son auténticos planes y si existe un plan para un problema, lo encuentra. Pero el interés práctico es que obtenía resultados mucho mejores que los planificadores conocidos hasta la fecha.

El sistema FF (Fast Forward) de Hoffmann y Nebel se propuso en 2001 con un rendimiento claramente superior a Graphplan. Hace uso de la aproximación anterior para calcular la heurística que estima la distancia entre un estado y la solución. FF presenta un esquema original que permite calcularla con mucho menos esfuerzo (algo importante, ya que la heurística se calcula en cada nodo). A parlar de la heurística el sistema FF aplica métodos de búsqueda para calcular el plan de la forma siguiente. Con un esquema de búsqueda hacia delante, utiliza estrategias de descenso por gradiente para encontrar una solución que puede no ser óptima. Para escapar de mínimos locales utiliza profundización iterativa[25]. Este sistema ha superado a Graphplan claramente en las competiciones de planificación. Aunque este planificador no alcanza las garantías teóricas de Graphplan, su superior rendimiento en la práctica lo hacen preferible.

El planificador FD (Fast Downward), de Helmert en 2006, mejoró el rendimiento de FF También está basado en búsqueda heurística y usa una representación multivaluada de las tareas de planificación. La descomposición jerárquica de esas

tareas en esta representación alternativa permite calcular una nueva heurística h_{CG} denominada *causal graph heuristic*. El planificador aplica una búsqueda del estilo *primero el mejor (best first),* alternando las heurísticas h_{FF} y h_{CG}. Como en el caso anterior, este planificador no es completo.

El planificador LAMA, propuesto por Richter y Westphal en 2010, ha supuesto un nuevo paso adelante en el rendimiento. Está basado en el esquema de FD, es decir, es un sistema basado en búsqueda heurística hacia delante, que incluye diversos métodos heurísticos. Alternadamente, LAMA utiliza la heurística h_{FF} y también otra heurística basada en *landmarks*, que son fórmulas que se han de cumplir para cualquier plan que resuelva el problema. La generación de *landmarks* y su ordenación han sido esenciales para el alto rendimiento de este planificador.

Por último, mencionamos una aproximación diferente que sigue ofreciendo un alto rendimiento. Dado un problema de planificación, la aproximación SATPLAN considera la traducción de la pregunta "¿existe un plan de *k* pasos?" a una fórmula preposicional, de forma que existe el plan si y solo si la fórmula es SAT. Para resolver la fórmula se utilizan resolvedores SAT —que han alcanzado altos niveles de eficiencia—. Si se ha encontrado un modelo de la fórmula, este se retraduce en términos de planificación; en caso contrario, se incrementa *k* y se repite el proceso**[26]**.

Aprendizaje automático

En anteriores apartados hemos hablado de programas de aprendizaje automático cuyo denominador común era que aprendían a reconocer y clasificar patrones. Vimos que la técnica más común era el aprendizaje mediante redes neuronales. Posteriormente, el interés se focalizó hacia técnicas de aprendizaje simbólico y en particular el aprendizaje de árboles de decisión. Todas estas técnicas forman parte de lo que se denomina *aprendizaje supervisado,* es decir, que aprenden a clasificar basándose en un conjunto de datos de entrenamiento cuya clasificación ya es conocida. El modelo resultante (red neuronal, árbol de decisión…) se usa posteriormente para determinar a qué clase pertenecen los nuevos datos que se van a presentar al algoritmo. En esta sección vamos a describir brevemente otras técnicas de aprendizaje, en particular el *aprendizaje no supervisado* y otra a medio camino entre aprendizaje supervisado y no supervisado llamada *aprendizaje por refuerzo*. También describiremos la técnica del *aprendizaje basado en casos*, que consiste en guardar en memoria los datos de entrenamiento sin construir un modelo a partir de ellos. Finalmente, hablaremos de dos de las técnicas que recientemente están teniendo más impacto: una que consiste en aprender un modelo gráfico probabilístico, en particular una red bayesiana, y otra que se basa en redes neuronales con múltiples capas que se denomina *aprendizaje profundo*.

Aprendizaje no supervisado

En los algoritmos de aprendizaje supervisado se dispone de un conjunto de datos de entrenamiento que permiten aprender un modelo. En el caso de un algoritmo que aprenda a clasificar, el conjunto de entrenamiento consiste en datos de los que se conoce, a qué clase pertenecen y esta información permite aprender un modelo (red neuronal, árbol de decisión…) a partir del cual poder clasificar nuevos datos, cuya clase es inicialmente desconocida. Sin embargo, en algunos casos existe la posibilidad de agrupar datos en clases distintas sin conocer la clase a la que pertenecen los datos contenidos en el conjunto de entrenamiento. En este caso decimos que el aprendizaje es no supervisado. Por ejemplo, supongamos que los datos se puedan representar mediante puntos en un espacio bidimensional, de tal forma que las dos coordenadas de cada punto corresponden a dos características f_1 y f_2 de los datos. Podemos pensar que aquellos datos cuyas características tienen valores próximos pueden pertenecer al mismo grupo o clase. Los algoritmos capaces de identificar automáticamente estos agrupamientos y los límites entre ellos son los algoritmos de aprendizaje no supervisado. Hay una amplia variedad de algoritmos no supervisados; la mayoría consiste básicamente en repetir los siguientes pasos:

1. Iniciar el proceso situando en el espacio de características un número prefijado *k* de puntos del conjunto de entrenamiento, que llamaremos centroides, que pueden ser seleccionados aleatoriamente (cada uno de estos centroides representa inicialmente a una clase de las *k* clases y es el primer elemento de dicha clase).
2. Clasificar el resto de los puntos en función de sus distancias a los centroides, de forma que tengamos en la misma clase a aquellos puntos que estén más próximos a su centroide que al resto de los centroides (debemos disponer, pues, de una medida de distancia o similitud).
3. Calcular el nuevo centroide para cada una de las *k* clases (el nuevo centroide de cada clase podría ser, por ejemplo, la media de los puntos que pertenecen a ella).
4. Repetir los pasos 2 y 3 hasta llegar a la convergencia; es decir, hasta que no haya puntos que cambien de clase (observemos que al haber cambiado la posición del centroide en el paso 3 habrá puntos que cambiarán de clase antes de llegar a una situación estable).

Una vez alcanzada la convergencia, el algoritmo clasificará datos nuevos asignándolos a la clase cuyo centroide sea el más cercano. Al tratarse de un algoritmo heurístico, no hay garantías de que converja a un óptimo global y, además, el resultado depende de la inicialización, pero como es un algoritmo que en muchos casos es rápido (a pesar de que en el peor de los casos sea exponencial), se suele

ejecutar varias veces con inicializaciones distintas con el fin de intentar evitar la convergencia a óptimos locales. Otro aspecto importante es que el número de clases no es siempre conocido de antemano, por lo que suele ser necesario añadir un paso de ajuste del número de clases consistente en minimizar la distancia entre puntos contenidos en la misma clase y maximizar al mismo tiempo la distancia entre puntos de clases distintas.

El algoritmo descrito pertenece a la familia de algoritmos llamados *k-means*. Existen numerosas extensiones y variaciones de este método. Una de ellas consiste en una versión secuencial del algoritmo, que es muy útil cuando no disponemos de todos los datos de golpe, sino que nos llegan secuencialmente. En esta variación el primer dato pasa a ser el primer centroide (y primer y único elemento de la primera clase). El siguiente dato se compara con el primero y, si su distancia es inferior a un umbral dado, se coloca en la misma clase y se recalcula el nuevo centroide (que suele ser la media entre los dos puntos que, en este momento, la primera clase contiene). En caso contrario, pasa a ser el primer centroide de una nueva segunda clase. Cuando llega el tercer dato, se compara con los centroides de las dos clases existentes y se clasifica en la clase del centroide más cercano si, de nuevo, la distancia es inferior al umbral. De lo contrario, será el primer centroide de la tercera clase y así sucesivamente hasta que no lleguen más datos para ser clasificados. Otra variación es el algoritmo AutoClass de Peter Cheeseman, famoso por haber descubierto una nueva clase de estrellas, además de nuevas clases de proteínas en el secuenciación del ADN.

Aprendizaje por refuerzo

Los algoritmos de aprendizaje por refuerzo se puede considerar que se sitúan entre los supervisados y los no supervisados. Aunque las versiones más conocidas y completas de estos algoritmos son relativamente recientes (Sutton y Barto, 1998), de hecho es un tipo de aprendizaje que ya se estudió en el caso de animales a principios del siglo XX (Thorndike, 1911). Por otra parte, el algoritmo de Arthur Samuel que aprendía a jugar a las damas jugando contra una copia de sí mismo, descrito en el capítulo 2, también es un tipo de aprendizaje por refuerzo.

En el aprendizaje por refuerzo un agente aprende a partir de interactuar con su entorno. Concretamente, a partir de las consecuencias de acciones que selecciona, ya sea según su experiencia previa (fase de explotación de lo aprendido), o según una selección aleatoria de las acciones posibles en cada situación, mediante un proceso de *prueba y error*. El agente recibe un valor numérico llamado refuerzo (que puede ser positivo o negativo), que codifica el éxito o fracaso de sus acciones. El objetivo del agente es seleccionar aquellas acciones que maximizan el refuerzo acumulado. No es aprendizaje supervisado porque no se proporcionan ejemplos correctos de pares estado-acción, pero tampoco es completamente no supervisado, pues el agente recibe un valor de refuerzo que guía su aprendizaje.

Un ejemplo muy típico es el de un ratón que debe aprender a recorrer un laberinto. En la salida del laberinto la recompensa podría ser, por ejemplo, un trozo de queso. La posición inicial dentro del laberinto sería el estado inicial y la salida, el estado final. Dentro del laberinto también tiene que haber situaciones fácilmente identificables que corresponderían al resto de estados del problema. Por ejemplo, las esquinas, los cruces de pasillos y los extremos de pasillos sin salida serían estados de este problema. En cada estado hay una o más acciones posibles. Por ejemplo, en un cruce hay cuatro acciones posibles, pero en el extremo de un pasillo solamente cabe solo la acción de ir hacia atrás. El agente no conoce el resultado de las acciones que toma, es decir, que no conoce *a priori* cuál es el estado siguiente antes de llegar a él. Dicho de otra manera, no conoce el mapa del laberinto. Si lo conociera, entonces este se podría representar mediante un grafo y se podría aplicar un método de búsqueda heurística[27] del tipo *A** para encontrar el camino que atraviesa el laberinto. La alternativa es aprender dicho camino, es decir, aprender una *política* que asocie estados con acciones.

El aprendizaje por refuerzo consiste precisamente en aprender una política que asocie a los estados aquellas acciones que conduzcan al camino más corto para atravesar el laberinto. Para ello el agente empieza con una estrategia de prueba y error recorriendo el laberinto de forma aleatoria hasta alcanzar el estado final (la salida en el caso del laberinto), de forma que la acción que lo condujo desde el estado anterior al final (o penúltimo estado) recibe un valor de refuerzo elevado. Este valor se propaga a su vez a la acción que condujo desde el estado anterior al anterior al final —el antepenúltimo estado— hasta el penúltimo, y así sucesivamente. Después de un elevado número de intentos un algoritmo de aprendizaje por refuerzo converge a unos valores que siempre conducirán al agente hasta el estado objetivo de forma eficiente.

Hay una serie de variaciones y extensiones de algoritmos de aprendizaje por refuerzo. Una variación consiste en asociar un valor inicial a cada posible acción en cada estado y luego usar el aprendizaje por refuerzo para ajustar estos valores de forma que, tomando en cada estado la acción cuyo valor sea el máximo, los valores ajustados conduzcan al agente hasta el estado final. Este algoritmo se llama Q-Learning, propuesto por Watkins en 1989 en su tesis doctoral. Una extensión importante tiene en cuenta la posibilidad de que el agente tenga un conocimiento imperfecto sobre el estado en que se encuentra. Se trata de una situación muy realista, sobre todo en el caso de los robots móviles, ya que sus sistemas de percepción no son lo suficientemente precisos como para que el robot pueda en todo momento saber con seguridad en qué posición se encuentra. En este caso, el estado se dice que está *escondido* y el problema se debe modelar mediante lo que se conoce como Proceso de Decisión Markoviano Parcialmente Observable (Mitchell, 1997).

Otro aspecto importante del aprendizaje por refuerzo es el compromiso entre explotar una política aprendida o seguir explorando (aprendiendo) con la esperanza de encontrar una política mejor. Es una decisión que depende de cada problema y se

basa en un proceso de experimentación. Finalmente, el mayor problema del aprendizaje por refuerzo es su escalabilidad. Cuando el número de estados y acciones posibles es muy elevado, el aprendizaje es muy lento, por lo que es necesario extender los algoritmos (por ejemplo, mediante técnicas heurísticas) para acelerar el proceso de aprendizaje.

El aprendizaje por refuerzo es una técnica muy popular en IA que se ha combinado con éxito con otras. En particular, se ha combinado con redes neuronales para desarrollar programas que juegan al Backgammon a un nivel superior al de los mejores jugadores humanos. Esta misma combinación de técnicas también está en la base del programa AlphaGo, que recientemente ha batido al campeón mundial de Go.

Razonamiento basado en casos

Es una aproximación al aprendizaje y resolución de problemas que hace énfasis en el papel que juega la experiencia adquirida resolviendo problemas a la hora de solucionar futuros problemas. Es decir, nuevos problemas se resuelven mediante la reutilización y, si es necesario, la adaptación de las soluciones a problemas similares que se resolvieron en el pasado (Kolodner, 1993). Es también una técnica de IA que se ha aplicado con notable éxito a una amplia variedad de tareas y dominios.

A pesar de que mucha de la inspiración para el estudio del *razonamiento basado en casos (case based reasoning*, CBR) procedía de la investigación sobre la memoria humana llevada a cabo por Roger Schank, la metodología resultante ha demostrado ser útil en una amplia gama de aplicaciones. A diferencia de la gran mayoría de las metodologías de aprendizaje en LA que se basan en aprender un modelo siguiendo un conjunto de ejemplos de entrenamiento, el CBR no aprende un modelo general, sino que memoriza ejemplos concretos de pares problema-solución. La hipótesis fundamental del CBR es que problemas similares tienen soluciones similares (López de Mantaras *et al*., 2006). Esta hipótesis se ha demostrado cierta para escenarios simples y se ha validado empíricamente en muchos ámbitos del mundo real.

Resolver un problema mediante CBR consiste en obtener una descripción del problema, calcular su similitud con las descripciones de problemas anteriores (almacenados en una base de casos junto con sus respectivas soluciones), *recuperar* uno o más casos similares y tratar de *reutilizar* la solución de uno de los casos recuperados, posiblemente después de su *adaptación* para tener en cuenta las diferencias en las descripciones de los problemas. A continuación la solución propuesta por el sistema, si es necesario, es *revisada* después de ser aplicada al problema inicial y se evalúa su validez por un experto del dominio, tras lo cual, tanto la descripción del problema como su solución pueden ser *retenidas* como un nuevo caso, con lo que el sistema memoriza el nuevo problema aprendido.

El modelo clásico de Agnar Aamodt y Enric Plaza sobre el ciclo de resolución de problemas en CBR consiste en las cuatro etapas mencionadas: *recuperar reutilizar,*

revisar y *retener* y se conocen como el ciclo de las 4R. Debido al papel fundamental de la etapa de recuperación en el ciclo CBR, una considerable cantidad de investigación se ha centrado en la recuperación de casos y, por consiguiente, en la evaluación de la similitud entre descripciones de problemas. Los aspectos relacionados con la reutilización y retención, y en menor medida con la revisión, también han dado lugar a importantes investigaciones.

Aprendizaje de redes bayesianas

Las redes bayesianas[28] pueden ser construidas automáticamente a partir de grandes bases de datos mediante técnicas de aprendizaje automático (Mitchell, 1997). Aprender automáticamente una red bayesiana implica aprender tanto la estructura como las probabilidades condicionales correspondientes a las relaciones de dependencia entre nodos representada mediante los arcos direccionales que los unen. Veamos cómo se pueden aprender ambos con el sencillo ejemplo de red bayesiana, concretamente cómo aprender las probabilidades condicionales para el nodo *D* (el coche arranca) que tiene como nodos padres el nodo *B* (el motor de arranque hace girar el motor del coche) y el nodo *C* (el sistema de alimentación de gasolina funciona). Es decir, necesitamos aprender las cuatro probabilidades siguientes: *P (D/B, C), P (D/B, ¬C), P (D/¬B, C)* y *P (D/¬B, ¬C).* Supongamos que tenemos acceso a una base de datos que contiene ejemplos de situaciones en las que un coche a veces arranca y otras no; también contiene situaciones en las que a veces el motor de arranque hace girar el motor del coche y a veces no, y finalmente también contiene situaciones en las que a veces el sistema de alimentación de gasolina funciona y a veces no. Entonces, en esta base de datos se puede, por ejemplo, calcular el número de veces que, simultáneamente, el coche arrancó, el motor de arranque hizo girar el motor y el sistema de alimentación de gasolina funcionó (es decir, *P (D, B, C*)) y dividirlo por el número de veces en que, simultáneamente, el motor de arranque hizo girar el motor y el sistema de alimentación de gasolina funcionó (es decir, *P (B, C)* y dividir la primera cantidad por la segunda con lo que obtenemos una estimación de la probabilidad condicional *P (D/B, C).* De forma similar se calcularían las tres restantes probabilidades. Finalmente, procederíamos a estimar las probabilidades condicionales correspondientes a los otros nodos, es decir, *P (B/A), P (B/¬A)*. Con un número suficientemente grande de ejemplos en la base de datos, estas estimaciones serían suficientemente fiables. Por otra parte, la estructura de una red bayesiana desconocida se puede aprender con base en un algoritmo que, básicamente, consiste en empezar con una estructura que refleje algún conocimiento previo acerca de potenciales conexiones entre nodos. En caso de no disponer de este conocimiento, la estructura inicial se limitaría a los nodos sin ninguna conexión entre ellos. Después se van añadiendo incrementalmente arcos estimando las probabilidades condicionales entre

pares de nodos de acuerdo con el contenido de la base de datos siguiendo el proceso mencionado.

A continuación se define una medida de calidad de la red obtenida, basada en calcular el grado de exactitud con el que la red es capaz de inferir las probabilidades de las situaciones contenidas en la base de datos con las probabilidades condicionales estimadas. Finalmente, se afina la red obtenida llevando a cabo un proceso de *hill climbing*[29] basado en pequeños cambios en su estructura (añadiendo y eliminando arcos e intercambiando pares de nodos), estimando de nuevo las probabilidades condicionales y midiendo el grado de calidad de la red modificada. El proceso se detiene cuando ya no se producen mejoras significativas en la calidad o aplicando algún otro criterio de parada previamente establecido, que puede estar basado en imponer restricciones relativas a parámetros relacionados con el tamaño de la red. Es un algoritmo computacionalmente costoso aunque, aplicando restricciones, se pueden tener algoritmos razonablemente eficientes. De hecho, existen miles de aplicaciones en campos tan diversos como, por ejemplo, la genómica, la predicción de las condiciones del tráfico rodado o el diagnóstico de enfermedades, y muchas de ellas han requerido el aprendizaje de redes con miles de nodos y arcos, y decenas de miles de probabilidades.

Aprendizaje profundo

Se conoce como *aprendizaje profundo* (*deep learning*) a una familia de métodos que aprenden representaciones de datos con múltiples niveles de abstracción (LeCun *et al*., 2015). Estos algoritmos aplican en cascada un conjunto de transformaciones no lineales, de forma que cada nivel recibe en entrada la salida del nivel anterior y lleva a cabo una abstracción aprendiendo así representaciones cada vez más complejas. Por ejemplo, para aprender a reconocer sillas basándose en imágenes de muchas sillas como datos de entrenamiento, el primer nivel de un sistema de aprendizaje profundo recibirá como entrada una matriz conteniendo los valores de los pixeles de una imagen y lo que aprende son características de bajo nivel, como, por ejemplo, líneas rectas con distintas orientaciones. Estas líneas constituyen los datos de entrada del segundo nivel que, a su vez, aprende una representación de características un poco más abstracta, como, por ejemplo, una serie de líneas formando contornos que delimitan regiones de la imagen. Estos contornos alimentan al tercer nivel que aprende otra representación de características todavía más abstracta, como, por ejemplo, partes de la silla, y así hasta el último nivel, en el que el sistema reconocerá la silla completa. El aspecto clave del aprendizaje profundo es que estas características cada vez más abstractas no están diseñadas manualmente por el programador del sistema, sino que son automáticamente extraídas mediante aprendizaje.

Uno de los métodos de aprendizaje profundo más conocidos son las redes neuronales profundas. En este caso el algoritmo básico de aprendizaje que se usa en cada uno de los niveles es, de hecho, el mismo que en las redes neuronales clásicas (es decir, no profundas) que contienen únicamente dos niveles. Este algoritmo, propuesto por Rumelhart, Hinton y Williams en 1986, se llama *backpropagation* y consiste en llevar a cabo un descenso por gradiente, actualizando los pesos de las conexiones entre neuronas, de forma que se minimice el error de clasificación de acuerdo a los datos de entrenamiento. Aunque las redes neuronales con múltiples capas no es un concepto nuevo, hasta ahora no se había podido usar debido a que requieren un número enorme de datos de entrenamiento para que sean capaces de aprender, además del uso de *hardware* de muy altas prestaciones. Recientemente, gracias a la disponibilidad de cientos de miles, incluso de millones, de datos de entrenamiento, como, por ejemplo, imágenes en Internet de toda clase de objetos, animales, etc., y gracias al acceso a ordenadores de altas prestaciones ha sido posible aplicar el método del aprendizaje profundo con éxito para reconocer patrones o incluso jugar a Go a nivel de los mejores jugadores del mundo. Actualmente so están desarrollando muchas aplicaciones del aprendizaje profundo, entre ollas al diagnóstico médico basado en imágenes y al reconocimiento visual de objetos en robótica.

Visión por computador

En el capítulo 2 vimos que los primeros éxitos en visión por computador a principios de los años setenta se basaban en la detección de líneas con el fin de identificar los bordes de los objetos para poder ser manipulados mediante brazos robóticos. Muy pronto se extendieron estos trabajos iniciales con el fin de extraer información tridimensional acerca de los objetos presentes en una escena. Por ejemplo, el conocimiento acerca de la física y geometría de la luz que reflejan las superficies de los objetos puede utilizarse para obtener propiedades tridimensionales a partir de una única imagen bidimensional. Uno de los primeros trabajos en esta línea fue el de Berthold Horn, en su tesis doctoral de 1970 en el MIT, que permitía determinar la forma tridimensional de un objeto basándose en la cantidad de luz reflejada por las distintas caras o superficies del objeto, ya que esta cantidad depende de los ángulos de inclinación de dichas caras respecto a la fuente de luz y a la posición de la cámara que capta la imagen del objeto. Otros investigadores extendieron estas ideas desarrollando técnicas que detectaban la forma de los objetos según otras aproximaciones, como, por ejemplo, mediante el uso de múltiples cámaras (estereovisión) o según la textura del objeto y también según el movimiento (Faugeras, 1993).

A finales de los años setenta y principios de los ochenta se empezó a desarrollar la *visión basada en modelos*. La idea consiste en representar los objetos

descomponiéndolos en sus partes constituyentes y especificando las relaciones espaciales que dichas partes deben satisfacer. Por ejemplo, la forma de un cuerpo humano se puede representar mediante una jerarquía de cilindros (Marr, 1982), de tal forma que, en una primera aproximación, todo el cuerpo correspondería a un cilindro que lo envuelve completamente; en el siguiente nivel de la jerarquía habría seis cilindros interconectados espacialmente de forma que cada una de las cuatro extremidades se aproxima por un cilindro, y el tronco y la cabeza por otros dos cilindros. En el siguiente nivel, cada extremidad se descompone en dos cilindros interconectados, uno para el brazo y el otro para el antebrazo y la mano; a continuación, el antebrazo y la mano se descomponen en un cilindro para cada uno y así sucesivamente los dedos, las falanges, etc., hasta el nivel de resolución que se requiera. Las relaciones o conexiones entre los cilindros deben satisfacer las restricciones correspondientes a las posiciones de las partes correspondientes de un cuerpo humano. Según este modelo, un sistema de visión sería capaz de detectar un cuerpo humano en una imagen si encuentra objetos en la imagen que se ajustan al modelo. Este modelo basado en cilindros se puede generalizar para detectar otros tipos de objetos en imágenes. Esta aproximación a la visión por computador basada en modelos está motivada por el problema del análisis de escenas, cuyo objetivo principal es de hecho construir un modelo tridimensional de los objetos presentes en una escena por su imagen bidimensional.

En vez de construir un modelo completo de las escenas visuales, existe otra aproximación que consiste en modelar únicamente aquello que sea necesario para guiar aquellas acciones que un robot debe llevar a cabo para ejecutar una tarea dada como, por ejemplo, navegar y manipular objetos. Esta aproximación se conoce como *visión activa* y está basada en la idea de que el propósito de la visión es la acción (es decir, que no se trata de desarrollar sistemas de visión de propósito general). Una ventaja de esta aproximación es que requiere menos recursos computacionales que los que se necesitan para modelar completamente una escena. Es interesante notar que entre los investigadores en visión bioinspirada también existen estas dos aproximaciones. Por una parte tenemos, por ejemplo, los trabajos de David Marr sobre la modelación de los procesos visuales humanos (Marr, 1982), que adoptan la aproximación del análisis completo de escenas. Por otra parte, otros investigadores que han investigado detalladamente la percepción visual en animales, como, por ejemplo, las ranas, concluyeron que se trata más bien de visión activa dirigida por el propósito de, en el caso de las ranas, atrapar insectos. Analizando la literatura científica existente podemos observar que ambas aproximaciones son importantes en IA. Por ejemplo, en el caso de los coches autónomos lo importante es detectar la carretera, así como la presencia de otros vehículos y obstáculos en y al lado de la carretera, pero no es importante detectar casas, árboles, montañas, etc., situados lejos de la trayectoria del vehículo, aunque formen parte de la escena visible desde el coche. Sin embargo, en otros contextos y aplicaciones es importante analizar

completamente toda la escena; por ejemplo, los sistemas que analizan fotografías con el fin de construir modelos tridimensionales de todos los elementos presentes en la imagen.

Últimamente gracias a la disponibilidad de grandes cantidades de imágenes en Internet y al acceso generalizado a la computación de altas prestaciones junto con recientes progresos en aprendizaje automático profundo se ha renovado el interés por resolver los problemas asociados al análisis de escenas completas. Los modelos más prometedores que se están usando son modelos jerárquicos. Existen múltiples propuestas en la literatura especializada, pero todas tienen en común que aplican secuencialmente un conjunto de transformaciones no lineales de forma que cada nivel de la jerarquía recibe en entrada la salida del nivel anterior y lleva a cabo una abstracción, aprendiendo así representaciones cada vez más complejas. Por ejemplo, el primer nivel agrega los píxeles de la imagen formando agrupamientos que corresponden a pequeños bordes, esquinas u otros componentes de los elementos presentes en la imagen; en el siguiente nivel se agrupan estos componentes obteniendo contornos completos y así sucesivamente hasta obtener objetos reconocibles[30]. Esta aproximación basada en el procesamiento estadístico no supervisado de grandes cantidades de imágenes de entrenamiento está produciendo resultados espectaculares en reconocimiento de imágenes, pero es una aproximación frágil, ya que no lleva a cabo un análisis profundo del significado de la escena. Por ejemplo, al analizar la imagen de un niño sujetando un cepillo de dientes con el cepillo en posición vertical y en primer plano, y por consiguiente con un tamaño aparentemente más grande, la respuesta del sistema es que se trata de un niño sujetando un bate de béisbol. La explicación de este error es que hay millones de fotografías de niños sujetando bates de béisbol accesibles en Internet y relativamente pocas imágenes de niños sujetando cepillos de dientes, por lo que el sistema llega a una conclusión errónea debido a la enorme diferencia entre el número de imágenes de ambas situaciones. Si el algoritmo llevara a cabo un reconocimiento profundo de las características que distinguen un cepillo de dientes de un bate de béisbol, no cometería un error tan importante. Actualmente se intenta mejorar el aprendizaje basado en el análisis masivo de datos, añadiendo componentes semánticos basados en conocimiento de sentido común para proporcionar más robustez a los sistemas.

Procesamiento del lenguaje natural con métodos estadísticos

La aproximación tradicional al tratamiento del lenguaje natural basada en análisis gramatical es rígida: una frase es correcta o no lo es, no caben situaciones intermedias. Se han hecho muchos esfuerzos para extender este tratamiento a las excepciones, ambigüedades, etc., sin conseguir el proceso ideal. Frente a esta aproximación, aparece la idea de procesar el lenguaje con métodos estadísticos (Manning y Schütze, 1999). Una frase se puede ver como una transmisión de

información (el contenido de la frase) a través de un canal con ruido (el texto). Podemos asociar una probabilidad a cada mensaje (significado), dada la salida observada (la frase textual). Esta visión asigna probabilidades a las distintas interpretaciones que se pueden dar en tareas del lenguaje natural. Para trabajar con probabilidades, es necesario disponer de un gran conjunto de ejemplos anotados con las características que nos interesan. Las matemáticas que están detrás de esta visión son especializadas y no entraremos en ellas.

La aproximación estadística al lenguaje natural es factible actualmente dado que existen muchos textos disponibles *online* (por ejemplo, en Internet hay textos legales traducidos a varios idiomas que pueden servir como ejemplo para traducción automática). Existe una relación cercana entre estos métodos para lenguaje natural y aprendizaje estadístico. La robustez de los modelos estadísticos para producir respuestas adecuadas a entradas que no se han visto antes —o que son erróneas— y la capacidad de mejorar su rendimiento simplemente procesando más datos de entrada son consideradas como buenas propiedades de estos modelos. Sin embargo, la estricta inferencia estadística es frágil y puede fracasar estrepitosamente en ciertos casos, dado que en estos sistemas no hay una *comprensión* profunda del mensaje.

Robótica[31]

Desde los finales de los años sesenta con robots como Shakey, los investigadores en IA han visto en la robótica móvil una excelente oportunidad para investigar aspectos fundamentales de la IA de forma integrada, es decir, teniendo en cuenta todos los componentes que conforman un sistema inteligente, desde la percepción hasta la acción, pasando por el razonamiento, el aprendizaje y la comunicación (Siciliano y Khatib, 2016). En la introducción ya se discutió la importancia que debería jugar el cuerpo en el desarrollo de la IA general, y los robots autónomos, y en particular los robots humanoides, constituyen una plataforma ideal para dicho desarrollo. Un ejemplo relativamente reciente de este tipo de investigación es la realizada en Stanford por Andrew Ng con el robot STAIR a finales del primer decenio de este siglo. So trata de un robot autónomo capaz de llevar a cabo una amplia variedad de tareas incluyendo la navegación en entornos domésticos y oficinas, recoger objetos y herramientas e interaccionar con ellos, y también mantener una conversación con humanos en estos entornos. STAIR incluso os capaz de aprender a agarrar objetos que no había visto anteriormente. Para ello, mediante su sistema de visión, so entrena su algoritmo de aprendizaje a partir de una base de datos que contiene miles de imágenes de múltiples objetos comunes en entornos de oficinas y en nuestras casas como, por ejemplo, tazas, vasos, platos, cubiertos, grapadoras, libros, teléfonos móviles, llaves, destornilladores, cepillos de dientes, etc. Cada imagen de entrenamiento para cada objeto contiene información sobre cómo agarrarlo. Además, para cada objeto so tomaron imágenes bajo condiciones distintas de luminosidad,

desde puntos de vista diferentes y con orientaciones distintas. Una voz entrenado, STAIR os capaz de determinar cuál es el mejor punto de agarre para objetos nuevos no vistos anteriormente y desarrollar un plan de acciones para mover su brazo y agarrar cada objeto. El porcentaje de éxito en el agarre está próximo al 90% (teniendo en cuenta que, para que se considere un agarre con éxito, el robot debe levantar el objeto a una altura de aproximadamente 30 centímetros y sujetarlo por lo menos durante 30 segundos).

Existen numerosos proyectos de investigación sobre robots autónomos similares no solamente en Estados Unidos, sino también en Europa, con el programa de innovación en robótica SPARC con una inversión de 700 millones de euros entre 2014 y 2020, y también en Japón. Este último es posiblemente el país donde más esfuerzos so dedican al desarrollo de robots humanoides. En 2015, el Gobierno japonés anunció la Japan's Robot Strategy con una inversión de unos 1.000 millones de dólares durante cinco años para innovar en robótica y, en particular, en robots autónomos con un énfasis en robots humanoides de servicio. Todos estos proyectos requieren integrar diversas técnicas de IA tales como aprendizaje automático, visión artificial, navegación, planificación, razonamiento, manipulación de objetos y procesamiento del lenguaje. Estas investigaciones son ejemplos muy interesantes de lo que se conoce como *sistema integrado*. La integración de todos estos componentes es una etapa imprescindible en el camino hacia las inteligencias artificiales de tipo general, aunque, por ahora, la gran mayoría tienen objetivos menos ambiciosos que el robot de propósito completamente general y se esfuerzan en conseguir robots capaces de realizar una variedad relativamente amplia de tareas dentro de entornos limitados, como, por ejemplo, nuestras casas y oficinas. Algunos ejemplos de estas tareas son coger y entregar objetos, recoger y limpiar, usar el lavaplatos, preparar comidas, montar y desmontar muebles, etc. Y a mayor largo plazo incluso asistir a seres humanos y, en particular, a personas mayores.

Los robots capaces de llevar a cabo este tipo de tareas son *robots de servicio*. El hecho de que estos robots deberán trabajar en entornos humanos e incluso colaborativamente con humanos plantea mayores dificultades que las que plantearon los robots industriales. Una de las más importantes es garantizar la seguridad para las personas con las que van a compartir espacios. Otro aspecto importante es que deberá ser posible enseñarles a realizar las tareas necesarias sin tener conocimientos de programación. Una posible solución es que sean capaces de aprender observando a una persona realizando dichas tareas. También deberán ser capaces de manipular con destreza objetos deformables, como, por ejemplo, ropa. Además, tendrán que tener una elevada capacidad de adaptación para realizar sus tareas de forma robusta en entornos no controlados, dinámicos e inciertos. Otra posible solución para que aprendan es mediante *la nube (cloud robotics)*. La ventaja reside en que cada robot podría, en principio, aprender de las experiencias del resto de robots a los que esté interconectado a través de la nube, lo cual debería permitir un progreso rápido de

aprendizaje y mejora de las capacidades de cada uno de los robots. Cuantos más robots formen parte de la nube, más rápido deberían aprender y mejorar. Una técnica de aprendizaje apropiada para este enfoque podría ser el aprendizaje profundo, ya que los robots deberán aprender basándose en grandes cantidades de dato[32].

Otro aspecto muy importante de los robots de servicios es que van a tener que ser sociales, es decir, que tendrán que ser capaces de colaborar de forma proactiva con personas. Ello está planteando una serie de complejos problemas técnicos que tienen que ver con la capacidad de comunicar de forma eficiente, tanto mediante el uso de lenguaje natural como lenguaje gestual, e incluso adivinar las intenciones del usuario observando dicha gestualidad. Por otra parte, también se plantean problemas de seguridad, por lo que estos robots tendrían que estar construidos con materiales blandos para evitar daños importantes en caso de que se produzcan colisiones o golpes accidentalmente al interaccionar con personas. Existe una variedad de aproximaciones a este problema. Además de las mencionadas (Torras, 2016), basadas en sensores o en propiocepción, se está también empezando a investigar en *robótica blanda* (*soft robotics*). Un robot blando está fabricado con materiales blandos, incluyendo los sensores y los actuadores con los que debe ser equipado; ello plantea enormes problemas técnicos como, por ejemplo, el control de estructuras deformables, pues la teoría clásica de control no es suficiente para manejar este tipo de estructuras.

Finalmente, también se plantean problemas de aceptabilidad por parte de las personas. Un estudio, llevado a cabo en la Universidad de Darmstad en 2016, afirma que más del 60% de un total de 600 empleados de Alemania y Estados Unidos aceptarían trabajar colaborativamente con robots siempre que estos se limiten a efectuar tareas repetitivas o poco agradables como buscar y archivar documentos, organizar las agendas o hacer de mensajeros. La mayoría no quiere que los robots expresen emociones en el trabajo. Además, la inmensa mayoría de los trabajadores no aceptaría que su superior jerárquico fuera un robot humanoide. Entre las razones expresadas para justificar este rechazo está el hecho de que los robots carecen de empatía, que no pueden juzgar a las personas o que no pueden ser un modelo a seguir. La cuestión de si es conveniente o no que un robot aparente tener emociones que realmente no tiene es controvertida, aunque hay argumentos a favor: el más importante es que la expresión de emociones, tanto faciales como corporales, y también la expresión oral de emociones mediante inflexión facilitaría la comunicación.

Hay varios grupos trabajando en robots humanoides con el objetivo de que se parezcan lo máximo posible a seres humanos, intentando incluso confundir a las personas haciéndoles creer que son humanos gracias al uso de materiales muy parecidos a la piel y el pelo. Los *geminoides* de Hiroshi Ishiguro son una muestra de ello, así como las cabezas robóticas, desarrolladas por la empresa Hanson Robotics, recubiertas de piel flexible y cuyas caras son capaces de generar expresiones faciales

y de mantener el contacto visual. Estas cabezas robotizadas parecen reales gracias a un novedoso polímero sintético que imita la piel humana, denominado *frubber*. Internamente, tienen un complejo sistema mecánico con 62 grados de libertad, es decir, pueden producir hasta 62 movimientos diferentes de cara y cuello. Además, van equipadas con una cámara en cada ojo y un sofisticado sistema de reconocimiento de las expresiones faciales y de la voz de sus posibles interlocutores. Los desarrolladores de este tipo de robots creen que el intercambio de emociones faciales facilitará la comunicación con personas e incluso puede tener aplicaciones terapéuticas en el caso de niños autistas o personas mayores. Sin embargo, otros piensan que este tipo de robot con apariencia tan humana puede producir un efecto de rechazo. En cualquier caso, creemos que estos plantean dudas de naturaleza ética que deben ser consideradas detenidamente. Un robot es un objeto mientras que un ser humano es un sujeto, por lo tanto: ¿es ético pretender que una máquina es un humano?

Sistemas multiagente

Los agentes inteligentes tienen que interactuar con otros agentes, ya sean humanos o máquinas. Estas interacciones pueden ser competitivas o colaborativas; en cualquier caso, es necesario modelarlas computacionalmente y la subárea de la IA que estudia estas interacciones entre agentes son los *sistemas multiagente* (*multiagent systems*, MAS). Estos han adquirido recientemente una gran importancia en IA, aunque sus orígenes se remontan a principios de los años ochenta en lo que por aquel entonces se conocía como *inteligencia artificial distribuida*. Los trabajos de Víctor Lesser sobre una versión distribuida de la arquitectura de HEARSAY-II, llamada DISTRIBUTED HEARSAY-II, son el origen de los SMA. HEARSAY-II es un sistema de reconocimiento del habla capaz de responder a preguntas y recuperar documentos de una base de datos con resúmenes de artículos de IA usando un vocabulario limitado aproximadamente a 1.000 palabras. Lo realmente innovador de HEARSAY-II es que se basa en una estructura multinivel llamada *pizarra* (*blackboard*), que permite representar el problema a resolver de forma jerárquica; por ejemplo, en el nivel inferior de la pizarra se representan los fonemas junto con su probabilidad de aparición en las palabras; en el siguiente nivel, se usa conocimiento, llamado *knowledge source* (KS), acerca de cómo se construyen las sílabas a partir de los fonemas para estimar la probabilidad de la presencia de ciertas sílabas; otras KS con conocimiento sobre la formación de palabras a partir de sílabas estiman la probabilidad de que palabras han sido pronunciadas por el interlocutor. Cada KS accede a la pizarra para leer en ella los datos proporcionados por los otros KS y poder también añadir información a la pizarra y todo ello de forma asincrona, sin depender de en qué momento las restantes KS escriben en, o leen de, la pizarra. Esto permite también que algunas KS con conocimiento sobre probabilidades de secuencias de

palabras puedan predecir la probabilidad de la siguiente palabra en función de las palabras ya presentes en la pizarra. La versión distribuida de HEARSAY-II consiste en una combinación de pizarras, cada una con sus KS, de forma que trabajando colaborativamente sobre el mismo problema son más eficientes a la hora de gestionar la incertidumbre de señales conteniendo ruido, mejorando así las prestaciones de HEARSAY-II en reconocimiento del habla. Otras aplicaciones realizadas con DISTRIBUTED HEARSAY-II incluyen el control distribuido de tráfico mediante redes de sensores para hacer el seguimiento de vehículos, y también los sistemas basados en multirrobots para llevar a cabo tareas que requieren tanto la cooperación como la competición, igual que en el caso de los robots jugadores de fútbol.

Puesto que en los MAS los distintos agentes deben cooperar o competir, un requisito básico es que cada agente debe tener un modelo de cada uno de los otros agentes, es decir, debe tener conocimiento acerca de lo que los otros agentes creen y saben, y sacar conclusiones y aprender a través de estas creencias y conocimientos. Por ejemplo, un agente *A* debería poder estimar a qué conclusiones puede llegar otro agente *B* conociendo las capacidades de razonamiento de *B*. Una de las herramientas que permiten representar formalmente estos modelos es la *lógica epistémica*. Para poder razonar y aprender a partir de lo que un agente sabe sobre otros agentes es necesario que los agentes se comuniquen. Las acciones de comunicación se denominan *actos de habla* (*speech acts*). Entre las distintas categorías de actos de habla tenemos las *aserciones*, para transmitir hechos de un agente a otro, las *directivas*, para transmitir órdenes, o los *compromisos*, para expresar que un agente se compromete a realizar una acción concreta. Una de la implementaciones más conocidas de la teoría de los actos de habla es el lenguaje KQML, que incluye predicados que, mediante expresiones en cálculo de la lógica de primer orden, representan acciones comunicativas tales como *pregunta-si*, *responde*, *informa*, etc. Actualmente existen diversas plataformas y lenguajes para implementar sistemas multiagente complejos, la mayoría basados en lógica matemática.

Uno de los temas actualmente más importantes en MAS son los modelos computacionales de *argumentación*. Esta permite modelar la comunicación entre agentes de forma más realista, ya que cada agente, en lugar de tener que creer por defecto lo que otros agentes le dicen, puede formular críticas a través de argumentos y contraargumentos, pedir clarificaciones e incluso persuadir a otros agentes de sus errores.

El método formal más conocido y usado para modelar arquitecturas de agentes es el de *creencias*, *deseos* e *intenciones* (*beliefs*, *desires*, *intentions*). Las creencias no son otra cosa que los conocimientos que un agente posee de sí mismo y de los otros agentes. Los deseos son los objetivos que un agente quiere alcanzar en algún momento y las intenciones representan los deseos concretos que, en un momento dado, un agente está intentando satisfacer mediante la ejecución de un plan de acciones. En el caso de un equipo de robots jugando al fútbol, las creencias de cada

uno de los agentes serían los conocimientos que tiene sobre el entorno en todo momento (su posición y las posiciones de sus compañeros de equipo y de los contrincantes, situación del balón, situación de la portería contraria) y conocimientos sobre cómo planificar jugadas. Los deseos serían en este caso conseguir marcar goles y defenderse para evitar goles del equipo contrario, y las intenciones serían las acciones concretas que en una determinada situación realizan siguiendo un plan, es decir, jugadas como avanzar con el balón, pasar el balón a un compañero, recibir un pase, etc.

Además de los sistemas multirrobot, hay muchas otras aplicaciones de los MAS, entre las que destacan el comercio electrónico y las redes de sensores. En estas, y en otras aplicaciones, se usan modelos computacionales de *negociación* para alcanzar acuerdos colectivos, así como modelos de *confianza* y *reputación* para que cada agente pueda evaluar su confianza en otros agentes y su reputación. También se usan modelos basados en la *teoría de juegos* y la *teoría de la elección social* con el fin de modelar situaciones en las que lo que elige hacer un agente depende de las elecciones de otros agentes con el objetivo de alcanzar una decisión colectiva que satisfaga a todos los agentes implicados (Wooldridge, 2009).

Capítulo 5
Grandes éxitos de la inteligencia artificial

En este capítulo haremos un repaso de algunas de las aplicaciones más exitosas de la IA focalizando nuestra atención en los últimos 20 años. Varias de estas aplicaciones han sido muy mediáticas, entre ellas están Deep Blue, AlphaGo, Watson, vehículos autónomos, robots en Marte, videojuegos, etc. En todos los casos describimos cuáles han sido las técnicas de IA aplicadas. Creemos que las aplicaciones seleccionadas son claramente representativas de las capacidades de la IA en el segundo decenio del siglo XXI.

Juegos por ordenador

Los juegos de ordenador han sido un objetivo de la IA desde sus inicios (Russell y Norvig, 2010). En particular, el juego del ajedrez ha sido durante décadas un objetivo permanente; dentro de la IA también se han estudiado otros juegos como las damas, Othello o Go. Estos son juegos denominados de *información perfecta*, porque no hay elementos ocultos ni estocásticos: toda la información disponible para un jugador también está disponible para el otro. Estos juegos se denominan *de suma cero* porque si un jugador obtiene una utilidad x, el otro recibe $-x$. Otros juegos, como el póquer o el *bridge*.) son de información imperfecta porque hay elementos del juego que son conocidos por unos jugadores, pero no por otros. En otros casos hay elementos estocásticos, como el lanzamiento de dados. Consideramos el escenario en donde un jugador es humano y el ordenador es el otro jugador; lo restringimos a dos jugadores, aunque hay extensiones a esquemas multijugador. En lo que sigue, llamaremos a nuestros jugadores A y B.

La aproximación clásica de la IA a este tipo de juegos es la denominada *minimax* (Ginsberg, 1993; Russell y Norvig, 2010). Se representa internamente mediante un *árbol de juegos*. Supongamos que le toca jugar a A: la raíz del árbol es el estado actual, con tantos sucesores como jugadas distintas posibles puede hacer A. Para cada una de esas jugadas, consideramos todas las posibles respuestas de B; a su vez, para cada una de estas respuestas, consideramos todas las posibles respuestas de A, etc. En el árbol de búsqueda los jugadores están alternados por niveles, y crece multiplicativamente en cada nivel.

Si pudiéramos desarrollar el árbol de juegos hasta posiciones ganadoras o perdedoras, que etiquetaríamos con +1 (ganadora) o −1 (perdedora) —con 0 para posiciones de empate—, y propagáramos hacia arriba —desde las hojas hasta la raíz — el mejor movimiento posible para cada jugador (están alternados por niveles, el mejor movimiento para A es el máximo de sus sucesores, mientras que el mejor movimiento para B es el mínimo de sus sucesores), al final el jugador A conocería la mejor jugada para él en esa tirada. El recorrido del árbol propagando hacia arriba los valores de las hojas del árbol lo realiza el conocido algoritmo minimax en su versión de búsqueda en profundidad. Si somos capaces de, mediante heurísticas, ordenar los nodos sucesores de un estado, de forma que las posiciones ganadoras se agrupen a la izquierda del árbol, la exploración mediante el algoritmo *alfa-beta* permite podar zonas ramas del árbol que representan zonas del espacio de búsqueda en donde no hay soluciones mejores que las que ya hemos obtenido. En la práctica, alfa-beta produce mejoras sustanciales con respecto al rendimiento de minimax.

La realidad es que no se puede desarrollar completamente el árbol de juegos porque es demasiado grande (un simple cálculo es útil aquí: el factor de ramificación del ajedrez es en torno a 35 y las partidas a menudo alcanzan 50 movimientos por jugador; el árbol de juegos tiene en torno a 35^{100} nodos, imposible de ser recorrido); solo se puede buscar hasta cierta profundidad, en donde no hay posiciones ganadoras o perdedoras. Los nodos a esa profundidad se han de evaluar mediante una *función de evaluación*, que retorna un número en el intervalo [+1,-1] y se propaga hacia la raíz mediante los algoritmos mencionados anteriormente. La profundidad a la que se explora el árbol de búsqueda no ha de ser necesariamente fija, puede variar en función de la zona. Un ejemplo es la *no quiescencia*, cuando el valor de la función de evaluación cambia rápidamente entre nodos a diferentes niveles (en ajedrez sucede cuando se capturan piezas). En ese caso, se recomienda continuar la búsqueda hasta que se recupera la quiescencia, y la función de evaluación vuelve a cambiar suavemente entre niveles.

El *efecto horizonte* aparece precisamente como resultado de buscar hasta determinada profundidad. Si una jugada aparentemente buena esconde un mal final detrás del horizonte de búsqueda (la profundidad a la que se busca), estamos indefensos frente a esto (que únicamente tiene un remedio: buscar más profundo, pero ¿dónde?). Una forma de enfrentar el efecto horizonte son las *extensiones singulares*. Si el algoritmo elige una jugada que solo está soportada por un nodo con una clara diferencia con sus nodos hermanos, se recomienda buscar más profundo bajo ese nodo. Se trata de confirmar la evaluación de esa jugada.

Hay juegos en los que la idea de función de evaluación, imprescindible en juegos de gran tamaño, no funciona (en el sentido de que no se conoce como construirla). Para esos juegos, una estrategia alternativa a la búsqueda minimax es la MCTS (Monte Carlo Tree Search) (Browne *et al.*, 2012).

MCTS desarrolla el árbol de juegos muy parcialmente (solo se memorizarán nodos que estén cerca de la raíz), suponiendo que la bondad de un nodo se puede estimar basándose en simulaciones del juego que parten de ese nodo hasta llegar a un nodo terminal (posición ganadora o perdedora). MCTS realiza un número de iteraciones hasta que se alcanza un máximo (en tiempo, número de iteraciones, consumo de memoria). Una iteración tiene cuatro fases:

1. *Selección*: partiendo de la raíz (estado actual) se selecciona un nodo hijo siguiendo una *política de árbol* (*tree policy*; este proceso se itera hasta encontrar un nodo no terminal con descendientes no visitados (nodo expandible).
2. *Expansión*: se añade un nuevo descendiente de ese nodo expandible, representa la ejecución de una acción.
3. *Simulación*: a partir de ese nuevo nodo, se eligen acciones alternativamente para los jugadores A y B hasta llegar a un nodo terminal (se simula la realización del juego entre A y B). La selección de acciones sigue la *política por defecto* (*default policy*).
4. *Actualización*: el resultado se registra en los nodos que conectan el nodo terminal con la raíz, actualizando sus estadísticas.

Este proceso depende de esas dos políticas mencionadas:

1. *Política de árbol*: se trata de seleccionar el nodo hijo del nodo actual, hasta llegar a nodos expandibles. Una política que ha dado buenos resultados es la UCB1, que elige la acción con más éxito hasta ahora, combinada con un elemento de exploración.
2. *Política por defecto*: en el paso 3, simulación, se trata de seleccionar acciones para los jugadores A y B hasta llegar a un nodo terminal. El caso más sencillo es una política aleatoria: escoger las acciones al azar.

MCTS desarrolla el árbol de juegos de forma asimétrica (las mejores acciones se seleccionan con más frecuencia). En algunos casos, se han desarrollado estrategias de poda. Cuando se alcanza el límite, se selecciona la acción con más éxito.

Otro conjunto de juegos incluyen elementos estocásticos —como tirar un dado— que determinan la evolución posterior del mismo. Un ejemplo es el Backgammon, donde las jugadas posibles de cada jugador dependen del resultado del lanzamiento de dos dados. El concepto de árbol de juegos permanece, aunque se modifica su estructura: a cada posible jugada de A, se consideran todos los posibles resultados de tirar los dados y, para cada resultado, todos los posibles movimientos de B. El árbol de juegos tiene niveles intermedios entre las posibles jugadas de A y las de B, que representan los distintos resultados de los dados. Este árbol de juegos se recorre con un algoritmo alfa-beta para propagar hacia arriba los valores de los nodos terminales

con el siguiente criterio: si el nodo corresponde a A, se propaga el máximo de los sucesores; si el nodo corresponde a B, se propaga el mínimo de los sucesores, y si el nodo corresponde a una tirada de dados, se propaga el valor esperado de minimax, que es la suma de los valores de cada sucesor multiplicado por su probabilidad. El estado actual de juegos por ordenador aparece reflejado en Russell y Norvig (2010). A continuación, presentamos tres juegos que han atraído mucha atención por parte de los investigadores de IA.

Sobre el ajedrez, en 1997 un equipo de IBM compuesto por seis personas (Feng-Hsiung Hsu, Joe Hoane, Chung-Jen Tan, Joel Benjamin, Jerry Brody y Murray Campbell), que habían trabajado tres años desarrollando el programa Deep Blue de ajedrez por ordenador, ganó al campeón del mundo en ese momento, Gari Kasparov, en un torneo celebrado a seis partidas según condiciones internacionales. Deep Blue era capaz de explorar 200 millones de posiciones por segundo, realizaba un alfa-beta paralelo, utilizaba una extensa biblioteca de aperturas y finales (4.000 aperturas, todos los finales con 5 piezas y muchos con 6 piezas), e incluía novedades como las extensiones singulares. Disponía de una extensa base de datos con 700.000 partidas a nivel de gran maestro para extraer recomendaciones. Realizaba de forma habitual búsquedas a profundidad 14, aunque en algún caso llegó a profundidad 40. La función de evaluación incluía 8.000 variables. Kasparov, poco feliz con el resultado del torneo, tuvo que reconocer que “la cantidad se había vuelto calidad”.

Sobre el juego de damas, el actual campeón (en su versión inglesa) es el programa CHINOOK, que implementa también una búsqueda alfa-beta y ganó al campeón mundial en 1990. Desde 2007, es capaz de jugar perfectamente, es decir, que nunca puede perder, combinando alfa-beta con una base de datos de millones de posiciones finales.

El juego del Go es, sin duda, el más complicado por el gran número de acciones posibles en cada jugada (el clásico árbol de juegos tiene un factor de ramificación en torno a 250), con partidas muy largas (hasta 150 movimientos). Para este juego, el esquema minimax no funciona (no se conocen buenas funciones de evaluación, el factor de ramificación es muy grande). Sin embargo, el esquema MCTS sí es útil. Con políticas genéricas, hay programas que alcanzan un nivel de aficionado débil. Cuando ostas políticas se entrenan con movimientos de expertos humanos, el nivel de juego se incrementa hasta un aficionado experto. En 2015 se desarrolló el programa AlphaGo, que ganó por 5 a 0 al humano campeón mundial *oficioso* de este juego. AlphaGo combina MCTS con dos redes neuronales profundas que permiten mejorar las políticas que se aplican; el resultado de MCTS se combina con el de una red neuronal. Los resultados obtenidos son excelentes.

Robots

En esta sección hablaremos de algunos éxitos indiscutibles de aplicaciones de la robótica. No vamos a incluir ninguna aplicación a la robótica que no tenga que ver claramente con el uso de la IA[33]. De entre los numerosos éxitos, nos limitaremos a los que consideramos más representativos. De hecho, tres de los robots descritos en esta sección (Spirit, Opportunity y Stanley), junto con Shakey y Robby, el robot de ficción de la película de 1956 *El planeta prohibido* constituyen, según la revista *Wire*; los cinco mejores robots de la historia.

Robots en el espacio

El primer robot que la NASA envió a Marte en 1997, Sojourner, medía solamente 65 cm de largo por 48 de ancho y 30 de alto, y pesaba unos 10 kg. Durante los aproximadamente 83 días que estuvo operativo (correspondientes a 85 días de la Tierra), recorrió aproximadamente 100 metros, a una velocidad de un centímetro por segundo, y nunca se alejó más de unos 12 metros de la estación Pathfinder. Envió 555 fotografías y tomó muestras del suelo y de rocas en 16 lugares distintos para poder analizar sus propiedades químicas mediante un instrumento que usaba espectrometría por rayos X. Aunque también estaba equipado con dos cámaras para captar posibles obstáculos durante sus limitados desplazamientos, de hecho no planificaba autónomamente su navegación, sino que desde la NASA se enviaban las instrucciones paso a paso al sistema que controlaba los giros y desplazamientos del robot; por consiguiente, no estaba equipado con ningún sistema de IA, pero sirvió para abrir el camino para futuros robots que, como veremos a continuación, ya incorporaron la IA en mayor o menor grado.

En 2004, la NASA envió a Marte los robots Spirit y Opportunity. Ambos unas cinco veces más grandes que Sojourner y con un peso de unos 180 kg. En principio debían estar operativos durante 90 días, pero ambos han excedido con creces estas expectativas. De hecho, Spirit estuvo operativo hasta el año 2009, cuando quedó bloqueado contra un obstáculo mientras navegaba. Sin embargo, Opportunity continuaba estando parcialmente operativo a finales de 2016, ha tomado más de 200.000 fotografías y ha recorrido más de 40 km durante estos 12 años, aunque sus espectrómetros dejaron de funcionar. El sistema de navegación de Spirit y Opportunity, aunque no es completamente autónomo, es mucho más sofisticado que el de Sojourner, que funciona de la siguiente manera: desde la NASA se envían las coordenadas precisas de localizaciones muy cercanas a la posición del robot, que forman una secuencia hacia el objetivo final a alcanzar. El sistema de navegación del robot avanza hacia dichas localizaciones, evitando posibles obstáculos no detectados por el teleoperador. Cada vez que el robot alcanza una de estas localizaciones, el teleoperador envía las coordenadas de la siguiente y así sucesivamente hasta alcanzar la posición objetivo final. Se trata todavía de una IA limitada, como lo demuestra el hecho que Spirit quedó bloqueado para siempre por un obstáculo que no pudo evitar.

La experiencia adquirida con estos robots permitieron a la NASA enviar de nuevo un robot a Marte en 2012 que a día de hoy sigue funcionando plenamente. El robot, llamado Curiosity, pesa unos 900 kg y en lugar de paneles solares se alimenta de energía nuclear. Se encuentra en una zona en las antípodas del robot Opportunity. Su misión también consiste en analizar suelo y rocas y encontrar indicios de presencia de agua en el pasado. Desde el punto de vista de la IA, Curiosity es mucho más sofisticado que sus antecesores en Marte. Por una parte, su *software* de navegación y su sistema de visión estereoscópica, equipado con un sofisticado *software* de análisis de imágenes basado en detección de contornos y otras características tales como brillo, tamaño, forma, orientación, etc., le permiten desplazamientos autónomos mucho más largos (del orden de decenas de metros) que los de Spirit y Opportunity. Además, a mediados de 2016, la NASA incorporó remotamente un nuevo *software* denominado AEGIS. Este *software*, desarrollado en el Jet Propulsion Laboratory, permite a Curiosity tomar decisiones sobre los objetivos científicos a los que dirigirse para llevar a cabo medidas usando la instrumentación con que está equipado. AEGIS permite apuntar el láser, que efectúa la medición obteniendo datos geoquímicos, sobre objetivos muy pequeños con una precisión superior a la que puede lograr un operador humano teleoperando desde una sala de control de la NASA. Además, es capaz de priorizar sus objetivos gracias a informaciones previamente proporcionadas por los científicos sobre las características que deben tener las rocas a analizar. Esta capacidad de decisión autónoma es muy importante, ya que reduce muy significativamente el tiempo necesario para llevar a cabo las mediciones y experimentos científicos, debido a que Curiosity no tiene que estar horas esperado a que lleguen las instrucciones transmitidas desde la NASA.

Otra presencia de IA en el espacio fue la sonda espacial llamada *Deep Space 1* equipada con el *software* llamado Agente Remoto (Remote Agent, RA), que la pilotaba. *Deep Space* fue lanzada en 1998 con el único objetivo de verificar el comportamiento de una serie de tecnologías en el espacio, entre ellas el agente remoto RA. Este *software* era un agente inteligente que planificaba y ejecutaba las acciones de la nave. El planificador recibía instrucciones del Gestor de Misiones (Mission Manager, MM) y usaba búsqueda heurística para generar los planes que a continuación se ejecutaban, mediante un sistema en tiempo real. También disponía de un módulo que monitorizaba la ejecución de las acciones del plan con el fin de identificar posibles fallos e intentar corregirlos. Algunos ejemplos de acciones eran conectar y desconectar los sistemas de propulsión que permitían orientar la nave, posicionar la cámara, etc. Un aspecto interesante de este sistema es que los operadores desde una sala de control en la NASA podían enviar instrucciones relativas a objetivos de alto nivel al MM como, por ejemplo: “Durante las próximas 48 horas toma fotografías de los asteroides X, Y, Z, utilizando un 90% del empuje del motor durante todo ese tiempo”, en lugar de tener que proporcionar una secuencia detallada de acciones distribuidas en el tiempo a lo largo de dichas 48 horas. Gracias

a este objetivo de alto nivel, era el planificador quien generaba las acciones detalladas para cumplir el objetivo.

Vehículos autónomos

A mediados de los años ochenta, la agencia americana DARPA lanzó su primer programa de I+D sobre vehículos terrestres autónomos con fines principalmente militares. En este programa participaron diversas universidades de primer nivel en Estados Unidos bajo la coordinación de la empresa Martín Marietta y del ejército estadounidense. Ello dio como resultado un vehículo (ALV) todoterreno, de 8 toneladas de peso, equipado con una cámara y un escáner láser que proporcionaban información sobre las posiciones de los bordes de la carretera que servían para generar un modelo de lo que había frente al vehículo. Un módulo de navegación y de búsqueda de objetivos visuales llevaba a cabo un proceso de razonamiento basado en reglas y planificaba las acciones que el piloto automático ejecutaba para guiar el vehículo, procurando no salirse de la carretera y evitando obstáculos. Dada la limitada capacidad de cálculo de los procesadores de aquella época y las limitaciones de los sistemas de visión artificial, este vehículo únicamente se podía desplazar a velocidades por debajo de los 10 km/h y los desplazamientos eran de muy pocos kilómetros de distancia. Además, los obstáculos eran artificiales y se colocaban a no menos de 100 metros los unos de los otros. Los obstáculos más pequeños que ALV podía detectar medían cerca de 1 metro de alto. La visión artificial era el problema principal, ya que en aquellos años la velocidad de procesamiento disponible estaba varios órdenes de magnitud por debajo de la necesaria para detectar robustamente en tiempo real la gran diversidad de obstáculos naturales y la carretera. De hecho, el 85% de la carga total de procesamiento lo usaba el sistema de visión y aun así era muy poco robusto, ya que era muy sensible a las variaciones de las condiciones de iluminación debidas a la posición del sol, la presencia de sombras, etc. En testeos sucesivos llevados a cabo con pocas horas de diferencia los resultados podían ser completamente distintos, no solamente detectando obstáculos, sino también detectando los bordes de la carretera. En 1988, desarrollar coches autónomos. Actualmente, no solo todos los grandes fabricantes de coches, sino también empresas de otros sectores como Google, Apple, Amazon o Huber están desarrollando tecnología para coches autónomos. Según ha anunciado recientemente el Departamento de Transporte de Estados Unidos, durante 2017 se destinarán 4.000 millones de dólares del presupuesto oficial para el desarrollo de automóviles autónomos. El objetivo a corto plazo es que 2.500 de estos vehículos puedan circular dentro de 2 años. El futuro que está en juego no es únicamente el del automóvil, sino el del transporte en general. Hay especialmente tres tecnologías que están jugando un importante papel en los automóviles: el reconocimiento y síntesis de voz, la cartografía digital avanzada y los sensores y cámaras. Actualmente, las compañías

automovilísticas abordan el desafío de adopción de estas tecnologías con estrategias distintas: mientras que Ford prefiere asociarse con fabricantes específicos de componentes, Toyota ha optado por crear una *spin-off* de IA, en colaboración con las universidades de Stanford y MIT, en la que ya ha invertido mil millones de dólares contratando a 200 personas. El objetivo principal es crear un coche incapaz de causar accidentes, pero el gran problema a resolver para tener coches autónomos es la robustez de las tecnologías empleadas, desde el *software* a los sensores, la electrónica y las comunicaciones. En todos esos casos estamos muy lejos de tener tecnologías robustas y, en particular, en el caso del *software*, ya que debe ser *safety critical* y por lo tanto deberá tener garantías absolutas de fiabilidad y de protección ante ciberataques, siendo necesario ir mucho más allá del estado actual de la ingeniería del *software*.

Teniendo en cuenta que las decisiones que dicho *software* va a tener que tomar tendrán que tomarse en milésimas de segundo, es mucho más complejo que, por ejemplo, el *software* que actualmente controla el pilotaje automático de un avión, ya que en el espacio aéreo el tiempo de reacción es por lo menos tres órdenes de magnitud más lento que en el tráfico rodado gracias a que el sistema de control de tráfico aéreo asegura prácticamente que los aviones más cercanos siempre estarán a cientos de metros de distancia, suposición que no se cumple con el tráfico rodado. Por consiguiente, la autonomía completa, sin ninguna intervención por parte del humano (incluso eliminando el volante, el acelerador y el freno), plantea enormes problemas tecnológicos. Por otra parte, también plantea serios problemas legales, éticos y económicos, además de posibles problemas de aceptación por parte de los usuarios.

Centrándonos en los aspectos técnicos, Google en 2016 dio cifras de cerca de 400 fallos que obligaron a desconectar el piloto automático y, de estos, unos 300 fueron fallos técnicos (fallos de sensores, pérdida de comunicación con Internet, problemas de dirección, problemas de frenado, etc.). Ese mismo año, Nissan confiesa que desconectó el piloto automático también en más de 400 ocasiones, y Mercedes más de 1.000 veces en solamente unos 3.000 km, unas 500 por fallo técnico y el resto por decisión del conductor que expresó “no sentirse cómodo”. En decenas de estos casos se evitaron accidentes gracias a la intervención del conductor humano. A corto e incluso a medio plazo, los humanos seguiremos jugando un papel fundamental en la conducción del vehículo, actuando como copilotos y supervisores, pasando el control del vehículo al piloto automático cuando ello sea posible, pero no siempre. A más largo plazo, cuando la tecnología haya resuelto las limitaciones actuales, posiblemente llegue el momento de plantearse seriamente la conducción totalmente autónoma. Otra fuente de problemas es debida a que las personas somos impredecibles en nuestro comportamiento y, en muchos casos, insensatos a la hora de tomar decisiones que pueden ser peligrosas. En estos casos tenemos que tener un *software* suficientemente inteligente como para tomar decisiones que eviten estos

peligros. En IA existe un área de investigación llamada *planificación tolerante a fallos*, que puede gestionar situaciones en las que el comportamiento del conductor se desvía del plan correcto. Pero para aplicarlo es necesario hacer estudios sobre aquellos elementos sutiles del comportamiento humano que deben de tenerse en cuenta para programar un robot (o un coche) para que pueda trabajar semiautónomamente y colaborativamente con un humano. Una vez estos elementos estén suficientemente identificados se deben incorporar al *software* de control, de forma que planifique sus acciones y sea capaz de crear un plan alternativo en situaciones de emergencia. Por ejemplo, cuando gracias a los sensores que monitorizan el grado de atención y fatiga del conductor el coche detecta que el conductor está cansado, el plan alternativo consistiría en comunicar al conductor que debe ceder el control y, si es necesario, proceder a cambiar la ruta inicialmente planificada y circular por carreteras donde es más fácil y seguro conducir de forma automática. En caso de no recibir respuesta, el coche debe tener la capacidad de, autónomamente, tomar la decisión de disminuir la velocidad, situarse en el arcén y parar el coche.

Entonces, ¿cuál es el camino a recorrer hacia el coche autónomo? Se habla de 5 niveles de automatización; los coches más avanzados actualmente se situarían en el nivel 3, lo que significa que el coche puede hacerse con el volante y los pedales, estar atento al entorno, pero depende por completo de la supervisión humana. A partir de 2020 se cree que se logrará el nivel 4, en el que será el coche quien lleve la voz cantante, aunque en ocasiones específicas requerirá intervención humana. El último nivel implicaría que el humano nunca tendrá que hacer nada. Para alcanzar el nivel 4 queda mucho trabajo por hacer. Uno es tener sistemas de localización mucho más precisos que ahora. Para ello se necesitará cartografía digital de muy alta precisión, que ubique objetos y elementos fijos muy concretos tales como los árboles a los lados de una carretera. Estos mapas, además, tendrán que ser en 3D con una resolución inferior a un metro y, para evitar la obsolescencia de las cartografías digitales actuales, será el coche el que los cree sobre la marcha mediante lo que en IA se conoce por SLAM (Simultaneous Localization and Mapping), un algoritmo que se usa extensivamente en robotica que permite que un robot explore un entorno desconocido, haga un mapa del entorno y, simultáneamente, calcule su localización en dicho mapa, todo ello mediante una aproximación probabilística que tiene en cuenta la incertidumbre inherente en los sensores, los motores y la dinámica del entorno.

En cuanto a los sensores para construir mapas del entorno y detectar obstáculos en movimiento, hay varias tecnologías que se están considerando, como, por ejemplo, las cámaras de muy alta resolución (posiblemente combinadas con nuevas generaciones de faros capaces de iluminar mucho más lejos que ahora sin deslumbrar a otros conductores), los sensores basados en láser (Ford está desarrollando estos sensores en colaboración con la empresa Velodyne, con un alcance de 200 m frente a

los 70 de anteriores modelos), y los infrarrojos (muy indicados para detectar personas y animales, actualmente hasta unos 40 m de distancia, quizá insuficiente todavía). Lo que es seguro es que serán necesarios todos estíos distintos tipos de sensores.

Por último, en los coches completamente autónomos, otra dificultad de naturaleza no técnica vendrá por aspectos relacionados con la responsabilidad legal. En caso de accidentes, ¿quién será el responsable? La respuesta no es evidente y quizá será necesario equipar los coches con *cajas negras* similares a las de los aviones con el fin de poder dilucidar responsabilidades (¿falló algún sensor?, ¿falló el *software*?, etc.). También será imprescindible reflexionar sobre aspectos éticos. Por ejemplo, en el *software* de un coche completamente autónomo se tendrá que haber previsto qué hacer ante alternativas como, por ejemplo, ¿salvar la vida de los pasajeros o de otras personas?

A la vista de tan numerosos y complejos problemas es lógico preguntarse si realmente algún día tendremos coches completamente autónomos. En cualquier caso, aunque así fuera no es de esperar que a medio plazo los veamos circulando en cualquier tipo de entorno; sí los veremos a corto/medio plazo en zonas controladas, ciertos tramos de autopista, campus universitarios, parques industriales, áreas comerciales, etc. Un ejemplo es la iniciativa británica de que circulen unos 40 pequeños vehículos de dos ocupantes a 8 km/h por los 250 km de calles peatonales y carriles para bicicletas de la ciudad de Milton Keynes. Estos vehículos harán las funciones de taxi al que se llamará mediante una *app* en el móvil. Pruebas similares se van a llevar a cabo en otras ciudades británicas y en otras partes del mundo como Singapur y Pittsburgh. Además, estamos viendo que ya se van incorporando funcionalidades cada vez más sofisticadas de ayuda a la conducción, como, por ejemplo, mantener automáticamente una distancia de seguridad con el vehículo que va delante, mantener el coche dentro de su carril, etc., con el fin de hacer la conducción humana más segura.

Lenguaje

Entre los éxitos de la IA en el área de lenguaje destacamos los sistemas Watson, Mastor y Siri, que pasamos a describir a continuación. Estos sistemas, en sus respectivos campos, alcanzan un rendimiento igual o superior al humano, y en algún caso podrían ser tomados por un ser humano real.

Watson

Watson es un programa desarrollado por IBM que ganó un concurso de respuestas a preguntas de cultura general en febrero de 2011, realizado en la televisión estadounidense. El programa *Jeopardy!* formula preguntas a tres concursantes, que han de responder rápidamente: una vez que el presentador lee la pregunta, el primer

concursante que tiene una respuesta aprieta un pulsador y a continuación pronuncia su respuesta. Respuestas incorrectas penalizan, por lo que un concursante ha de tener una cierta confianza de su calidad antes de apretar el pulsador. Las preguntas de *Jeopardy!* son enrevesadas, incluyen pistas y juegos de palabras, y en ocasiones indican el tipo de respuesta. Dada la amplitud de los temas tratados y el formato de preguntas, participar en *Jeopardy!* suponía un desafío muy serio para cualquier equipo que pretendiera desarrollar un contrincante computacional.

Técnicamente, Watson se encuadra en los *sistemas de respuesta a preguntas en dominios abiertos* (*open-domain question answering*). La arquitectura interna de Watson se basa en la tecnología DeepQA, desarrollada por IBM, que combinaba técnicas de recuperación de información, lenguaje natural, representación del conocimiento y razonamiento, y aprendizaje. Dada la amplitud de los temas tratados en *Jeopardy!* se tenían que manejar múltiples fuentes de información sin estructura (como la que se encuentra en un texto libre). La idea básica de Watson era disponer de grandes cantidades de contenidos (texto, voz, imágenes) que se anotaban con significado (semántica) en determinadas regiones. Estas anotaciones las realizaban de forma cooperativa una serie de programas denominados anotadores; cada uno realiza tareas relativamente simples que, en conjunto, producían un resultado complejo. Watson era autocontenido, no estaba conectado a Internet. Antes del juego, su memoria fue *alimentada* con enciclopedias, obras de referencia, toda la Wikipedia, etc. Para el juego, toda esta información (en torno a cuatro terabytes) se cargó en una memoria RAM, pues el acceso a disco enlentecía demasiado el sistema.

Una parte importante de DeepQA se dedicó a comprender bien la pregunta, así como el tipo de respuesta esperada y las pistas que contenía. A partir de ahí, DeepQA no buscaba una comprensión absoluta con respecto a un modelo determinado, ontología o base de datos. Los resultados del análisis de la pregunta (que incluía un análisis sintáctico) eran múltiples interpretaciones, que enfrentadas a los contenidos del sistema generaban un amplio conjunto de respuestas candidatas. Cada respuesta candidata junto con la pregunta representaba una hipótesis independiente que se puntuaba, incluyendo indicios adicionales que se basaban en técnicas gramaticales (sintácticas y semánticas) y en técnicas estadísticas de proceso del lenguaje (capítulo 4). Además, había un tratamiento específico para preguntas que exigían un manejo de datos estructurados.

En ocasiones, las preguntas de *Jeopardy!* eran especiales, en el sentido de que incluían referencias implícitas que se debían resolver para responder con precisión. Otra técnica era la descomposición de una pregunta en subpartes lógicas, que podían ser tratadas de forma independiente y combinarse sus resultados.

Para una pregunta, DeepQA podía reportar 100 respuestas candidatas, cada una de ellas soportada por unos 100 indicios, y cada par indicio-respuesta podía ser puntuado de 100 formas diferentes; para cada candidato había que resolver en torno a 10.000 puntuaciones diferentes. La combinación de estas puntuaciones con sus confianzas

respectivas en una única probabilidad se hacía mediante técnicas estadísticas de aprendizaje. La independencia de los análisis era útil: si diferentes algoritmos convergían en la misma respuesta, aumentaba su probabilidad de que fuera correcta. A veces se hacían verificaciones cruzadas en tiempo o espacio para descartar candidatos.

Para conseguir una respuesta rápida (unos 3 segundos de media, a nivel humano), se recurrió al paralelismo masivo (2.880 procesadores). Sobre DeepQA se desarrollaron dos módulos para realizar el juego: un módulo de estrategia (qué preguntas elegir, cuánto apostar en algunas preguntas, cuándo responder y cuándo no, etc.) y otro módulo de interfaz. Watson no ve ni oye, las preguntas se las proporcionan por escrito y un sintetizador de voz pronuncia las respuestas.

En *Jeopardy!*, Watson ganó a brillantes contrincantes humanos elegidos entre jugadores del concurso. Lo hizo de forma clara, con las mismas reglas de juego que los jugadores humanos. Antes, Watson había jugado un conjunto de partidas de ensayo contra contrincantes humanos, que sirvieron para corregir puntos débiles. En el futuro, IBM conjetura la utilización de esta tecnología para tareas que exigen consultar un gran número de textos o referencias, como el razonamiento médico, el razonamiento legal o servicios *Online* de atención al cliente.

Siri

Siri es un *asistente personal virtual (virtual personal assistant)* desarrollado para los iPhones de Apple. Se comunica mediante el habla: escucha las preguntas del usuario y pronuncia sus respuestas. Responde preguntas, hace recomendaciones y realiza acciones mediante los servicios web del aparato (por ejemplo, encontrar un restaurante cercano o comprar unas entradas de cine). Claramente, Siri procesa el lenguaje natural para sus interacciones con el usuario (recibir peticiones y formular respuestas). Además, ofrece una interacción conversacional con otras aplicaciones como los recordatorios, el estado del tiempo, la bolsa, la mensajería, el correo electrónico, el calendario, los contactos, las notas, la música, el reloj, el navegador web y los mapas. Siri también puede devolver llamadas, leer los mensajes del buzón de voz, etc., y dispone de soporte en varios idiomas. Siri no incluye una tecnología radicalmente nueva: su principal contribución es que ha juntado diferentes tecnologías complementarias existentes en un único sistema.

Un elemento clave en Siri es el reconocimiento del habla. Siri tiene dos modos básicos: reconocimiento de comandos y dictado. Cuando Siri pregunta, puede acotar bastante en ámbito de la respuesta con un vocabulario limitado. En esos casos utiliza un sistema basado en gramática para procesar la respuesta. Por el contrario, cuando recibe un dictado del usuario, utiliza un sistema de reconocimiento de voz intentando transcribir cada palabra que oye. Aunque no es necesario una fase de entrenamiento a la voz del hablante, Siri mejora su comprensión a medida que el usuario lo utiliza

(cómo pronuncia las palabras, qué palabras son más frecuentes, cómo pronuncia nombres y cifras). La parte cognitiva de Siri ha de ser capaz de *comprender* los elementos de los que se habla (identificar nombres con personas, duraciones de eventos, etc.). Tanto por su parte cognitiva como la de reconocimiento del lenguaje, Siri ha sido entrenado con miles de datos obtenidos de la web.

Las colecciones de datos que son explotados por Siri son demasiado grandes para los iPhones. Siri utiliza conexiones de alta velocidad para reconocimiento de voz e interroga a otros proveedores de información (consultando fuentes como Bing, Wikipedia y Twitter) sobre preguntas concretas. Siri puede incluso conectarse a *The New York Times*.

Mastor

Mastor (Multilingual Automatic Speech-To-Speech Translator) es un sistema de *traducción automática de voz* desarrollado por IBM, que combina reconocimiento automático del habla, traducción automática del lenguaje natural y síntesis de voz para facilitar conversaciones en tiempo real entre dos hablantes que no tengan un lenguaje común. Este sistema se ha usado en contextos militares y de emergencias.

Mastor se basa también en métodos estadísticos para realizar la traducción. Sobre el reconocimiento automático del habla, se han utilizado modelos acústicos de los idiomas a tratar combinados con modelos del lenguaje. Estos modelos se basan en fonemas y también en grafemas para idiomas como el árabe, que tiene muchas vocales cortas que no aparecen en la forma escrita y que un hablante nativo puede inferir por el contexto. Estos modelos se procesan a partir de modelos estadísticos basados en las pronunciaciones de muchos textos. Sobre los modelos de los lenguajes, se basan en calcular la probabilidad de secuencias de palabras candi-datas en la traducción. Se utiliza un modelo *trigram* (probabilidad de ocurrencia de tres palabras consecutivas), que se ha construido a partir del análisis de muchos textos. En árabe, el vocabulario puede ser muy extenso, debido a que la raíz de muchas palabras se puede combinar con prefijos y sufijos, por lo que se emplean técnicas de análisis morfológico para reducir hasta en un 30% el tamaño del vocabulario.

Con respecto a la traducción automática, la aproximación también es estadística, a partir de un corpus anotado entre los dos idiomas. Mastor elige la secuencia traducida que aparece con más probabilidad. En un intento para evitar la fragilidad de los sistemas puramente estadísticos, Mastor incluye en el cálculo de la probabilidad el (o los) conceptos que aparecen en la frase original y en la frase traducida.

Otras aplicaciones

Por razones de espacio hemos tenido que dejar en el tintero, o mejor dicho en el teclado, multitud de aplicaciones que también podríamos calificar de grandes éxitos

de la IA. Sin embargo, en esta sección vamos a comentar algunas áreas de aplicación que consideramos muy relevantes. Entre ellas están los videojuegos, en particular aquellos en los que el jugador compite con agentes inteligentes, los llamados *non player characters* (NPC): personajes animados presentes en el juego que el jugador no controla. Los NPC tienen que desplazarse de un lugar a otro evitando obstáculos y tienen que tomar decisiones acerca de cuál de sus posibles acciones es la más adecuada en función del contexto, es decir, de la posición y acciones de otros NPC y del personaje controlado por el jugador.

Entre las técnicas de IA que usan los NPC tenemos la búsqueda heurística, la planificación, las redes neuronales, los sistemas deductivos basados en reglas y el aprendizaje de tácticas e incluso estrategias para adaptarse a las habilidades de cada jugador. Los entornos simulados con alto nivel de realismo que ofrecen los videojuegos hacen que estos sean una interesante plataforma para investigar en IA y en robótica. Por ejemplo, investigadores de la Universidad de Michigan en Ann-Arbor han entrenado un *software* para la conducción autónoma de coches mediante el juego *Grand Theft Auto*, ya que, al ser tan realista, constituye una excelente simulación del mundo real. Empresas como Google y Uber entrenan su *software* de conducción autónoma haciendo que los coches recorran millones de kilómetros en tráfico real, a la vez que graban imágenes desde el interior de un coche que posteriormente etiquetan indicando dónde empieza y termina la imagen de cada uno de los coches presentes en cada escena. Se trata, pues, de un proceso muy laborioso que requiere muchas horas. Sin embargo, en juegos como *Grand Theft Auto* todo lo que aparece está ya automáticamente preetiquetado, ya que está generado por el *software* del juego.

Un estudio comparando el resultado de entrenar el mismo *software* con imágenes sintéticas de *Grand Theft Auto* y con imágenes reales muestra resultados muy parecidos, aunque en el caso del videojuego se necesitaron muchas más imágenes de entrenamiento, aunque esto no supone ningún problema, ya que se pueden generar medio millón de imágenes sintéticas en unas horas. El problema es que haría falta sintetizar imágenes de ciudades distintas, simulando diferentes condiciones de iluminación y de tráfico para realmente tener un conjunto de entrenamiento suficientemente amplio como para que lo que aprende el *software* sea suficientemente general, evitando el *overfitting* del algoritmo de aprendizaje.

Otra importante área es la robótica *ammaloide*, llamada así porque los robots tienen la forma de animales. El ejemplo más popular son los robots AIBO, en forma de perro, desarrollados por SONY. Ha habido multitud de aplicaciones con estíos robots; posiblemente la más conocida y exitosa es la aplicación al fútbol robótico. Efectivamente, en los campeonatos de fútbol robótico conocidas como RoboCup, había una categoría dedicada exclusivamente a equipos formados por robots AIBO. La idea de desarrollar robots jugadores de fútbol como plataforma para progresar en IA, y en particular en la integración de la percepción, el razonamiento y la acción, fue

sugerida por el Alan Mackworth en un artículo publicado en 1993. Paralelamente, también en 1993, varios grupos japoneses de investigación en robótica liderados por Hiroaki Kitano pusieron en marcha una liga de fútbol robótico en su país que, muy pronto, se internacionalizó bajo el nombre de RoboCup, ya que otros grupos de investigadores en robótica, principalmente en Estados Unidos, también estaban desarrollando robots futbolistas. El primer campeonato oficial de RoboCup fue un gran éxito, con la participación más de 40 equipos y con miles de seguidores.

Existen varias categorías de ligas en función del tamaño de los robots participantes, además de una liga de robots simulados. Actualmente muchos países tienen sus ligas nacionales. Esta competición ha permitido progresos muy significativos en IA, ya que, tal como sugirió Mackworth, requiere la integración de diversos componentes fundamentales de la inteligencia como son la percepción, el razonamiento, la planificación, la acción, la comunicación y el aprendizaje. Por otra parte, el aprendizaje permite que los robots puedan jugar de forma colaborativa, planificando jugadas relativamente complejas que impliquen pases de balón.

La medicina es otra área de aplicación que históricamente ha sido clave en el desarrollo de la IA. Una de las técnicas de IA más usadas en aplicaciones médicas son los sistemas expertos. Además, hay muchos otros en este campo que se siguen usando regularmente en hospitales y centros médicos en todo el mundo. Uno de ellos es ATHENA, un sistema de ayuda a la toma de decisiones de los médicos a la hora de gestionar pacientes con problemas de hipertensión. Procesa los datos clínicos de cada paciente y, gracias a su base de conocimientos sobre hipertensión, produce una serie de recomendaciones sobre cómo gestionar mejor la atención clínica personalizada. Otra aplicación muy importante es el sistema GIDEON de ayuda al diagnóstico de un total de 337 enfermedades infecciosas específicas de cada uno de los 224 países de su base de datos. Su base de conocimientos cubre 1.147 taxones microbianos y 306 agentes antibacterianos y vacunas. La información que maneja incluye también más de 20.000 imágenes, gráficos, mapas interactivos, etc. Todo ello permite alcanzar más del 94% de diagnósticos correctos y lo convierten en uno de los sistemas más usados en el ámbito de la medicina.

Más recientemente han irrumpido con fuerza en la medicina, basada en la evidencia, las técnicas de análisis de grandes cantidades de datos. En poco tiempo se ha extendido enormemente su uso y se han conseguido resultados espectaculares. Vamos a comentar brevemente dos casos de éxito. Analizando grandes cantidades de muestras de células mamarias cancerosas, investigadores de la Universidad de Stanford han descubierto tres nuevos indicadores para evaluar con más confianza la probabilidad de que una biopsia de células mamarias pueda resultar positiva. Por otra parte, científicos de la Universidad de Carnegie Mellón, en colaboración con cuatro hospitales de Chicago, han desarrollado un sistema capaz de predecir infartos con cuatro horas de antelación en enfermos ingresados en la UCI, mejorando en más de tres horas los tiempos de predicción de los cardiólogos. Este *software* fue entrenadlo

con datos de 133.000 pacientes, incorporando 72 parámetros presentes en la historia clínica de los enfermos, incluyendo signos vitales, edad, glucemia y recuentos de plaquetas.

También en biología molecular y farmacología se está aplicando con éxito la IA. Por ejemplo, muchos fármacos tienen efectos secundarios inesperados que pueden resultar muy beneficiosos. Un ejemplo es la Viagra, que, aunque inicialmente se*desarrolló para controlar la hipertensión, resultó ser muy efectiva para tratar la disfunción eréctil; o la lovastatina, un efectivo tratamiento de la hipercolesterolemia que ha resultado ser un potente antibiótico. Pues bien, en lugar de esperar que estos beneficiosos efectos secundarios se descubran por casualidad, investigadores en farmacología aplican técnicas de IA para predecir qué fármacos ya existentes pueden tener otros usos terapéuticos, y en particular predecir las propiedades antibióticas de dichos fármacos con el consiguiente ahorro de tiempo que ello supone frente al desarrollo y aprobación de un nuevo fármaco. Creemos que el papel de la IA en el sector de la salud será cada vez más importante.

En otros ámbitos como, por ejemplo, el medioambiente y el ahorro energético (la aplicación de Deep Mind permite que el centro de datos de Google recorte en un 15% su consumo eléctrico), la IA se convertirá en una tecnología imprescindible para afrontar los grandes desafíos a los que se enfrenta la humanidad.

La economía y la sociología también usan cada vez más modelos de IA, en la línea de *simulación basada en agentes*. Un ejemplo es el modelo propuesto por Farmer y Foley en 2009, que permite simular interacciones entre grandes cantidades de agentes y poder predecir los efectos que causaría introducir elementos nuevos en determinados sistemas (como, por ejemplo, los efectos que tendría sobre la movilidad urbana la construcción de un parque o una zona peatonal, o los efectos sobre la economía y la ecología de la construcción de una autovía o un aeropuerto). De esta forma, las decisiones sobre dichas actuaciones se pueden tomar con mucha más y mejor información disponible.

También existen importantes resultados de aplicaciones de la IA en ciencias y, en particular, en matemáticas. Un ejemplo lo tenemos en la demostración de la conjetura de Robbins. En 1934, Herbert Robbins conjeturó que las álgebras de Robbins son álgebras de Boole. Un álgebra de Robbins es un álgebra que contiene un solo operador binario (la unión $\vee$) y un operador unario (la negación $\neg$) y estos operadores satisfacen los axiomas siguientes: asociatividad ($A \vee (B \vee C) = (A \vee B) \vee C$), conmutatividad ($A \vee B = B \vee A$) y la ecuación de Robbins ($\neg(\neg(A \vee B) \vee (A \vee \neg B)) = A$. En 1997, William McCune demostró la conjetura usando el demostrador automático de teoremas Equational Prover, que demuestra teoremas de lógica ecuacional, es decir, cálculo de predicados de primer orden cuyo único predicado es la igualdad.

De hecho, la lista de aplicaciones de la IA es interminable, por lo que simplemente vamos a enumerar algunas más sin dar detalles. Los sistemas de inyección de gasolina en nuestros automóviles están diseñados utilizando algoritmos

genéticos, así como las turbinas de los aviones a reacción. En el metro de Hong Kong durante la noche 10.000 ingenieros llevan a cabo 2.600 trabajos de mantenimiento, a lo largo de los 218 km y 152 estaciones de la red, programados por un *software* de planificación. La detección automática de transacciones fraudulentas de tarjetas de crédito se realiza mediante algoritmos de aprendizaje automático. El enrutamiento de las llamadas de teléfonos móviles se basa en IA. La detección de hábitos de consumo se basa en el análisis automático de grandes cantidades de datos mediante algoritmos de aprendizaje automático.

Capítulo 6
Futuro

A lo largo de este libro nos hemos centrado en describir el estado del arte de la IA actual, la que se conoce como IA débil, es decir, inteligencias artificiales que únicamente muestran comportamiento inteligente en un ámbito muy especializado. Esperamos haber demostrado que en ciertos dominios los avances de la IA débil superan en mucho la capacidad humana. Sin embargo, en la introducción ya anticipamos que el objetivo último de la IA es lograr que una máquina tenga una inteligencia de tipo general similar a la humana. El tema de este último capitulo es la IA de tipo general. Para ello en una primera sección del capítulo argumentaremos que un paso previo imprescindible hacia la IA general son los *sistemas integrados* y a continuación hablaremos de la posibilidad de conseguir desarrollar inteligencias artificiales generales.

Sistemas integrados

La inmensa mayoría de investigadores en LA hemos oído hablar de lo que se conoce como *metáfora de la catedral*, considera que construir una LA de propósito general es como construir una catedral. La construcción de la primera catedral requirió de varias generaciones y por ello la mayoría de los que trabajaron en su construcción no llegaron a verla terminada. Muchos eran artesanos que se dedicaron a construir ladrillos cada voz más perfectos y resistentes, que en su momento formarían parte de la catedral. Actualmente, y desde hace varias décadas, los investigadores en IA estamos construyendo también los ladrillos que compondrán la catedral, es decir, los algoritmos capaces de, por ejemplo, analizar el lenguaje, razonar, planificar y aprender. Y estamos continuamente mejorándolos para que hagan osas funciones cada voz mejor. De hecho, los humanos, antes de poder construir una gran catedral, construyeron muchos otros edificios mucho más sencillos comenzando por cabañas de barro y paja, que posteriormente se convirtieron en casas cada vez más resistentes y edificios cada vez más complejos, aprendiendo de sus errores y fracasos a lo largo de muchos años hasta que finalmente consiguieron encontrar los materiales adecuados para construir no solamente grandes catedrales, sino también enormes rascacielos, puentes y otras grandes estructuras. Por otra parte, es obvio que disponer de ladrillos perfectos no es suficiente para construir estructuras complejas, también es

imprescindible tener una buena arquitectura. Es decir, por muy sofisticados que sean los algoritmos de razonamiento, planificación, aprendizaje, etc., nunca conseguiremos construir inteligencias artificiales generales si no sabemos cómo integrar adecuadamente todos estos elementos. Solamente combinando estos elementos dentro de sistemas cognitivos integrados podremos empezar a construir IA general; sin embargo, la mayoría de las actuales investigaciones en IA siguen insistiendo en mejorar los ladrillos de forma demasiado aislada, sin colaborar estrechamente con aquellos que tratan de construir los edificios, los científicos que investigan lo que so conoce como arquitecturas cognitivas. Por otra parte, si hubiera más de estos últimos, y menos constructores de ladrillos, los progresos hacia la IA general serían más rápidos.

Las arquitecturas cognitivas intentan resolver el problema de cómo integrar los distintos componentes de la inteligencia explorando hipótesis acerca de la naturaleza de la inteligencia y de las posibles interacciones entre dichos componentes. La primera propuesta de arquitectura cognitiva fue el GPS (Feigenbaum y Feldman, 1963), del que ya hemos hablado en el capítulo 1. Muchos años después, en 1998, Anderson y Lebière propusieron la arquitectura cognitiva ACT-R inspirada en los trabajos de Allen Newell sobre una *teoría unificada de la cognición*, que, además de integrar teorías de atención visual y movimiento motor (para percibir y actuar en el entorno), está diseñada para poder modelar el proceso de resolución de problemas en una variedad de tareas mediante una memoria declarativa, formada por piezas de conocimiento llamados *chunks* (por ejemplo, “París es la capital de Francia”), y otra memoria procedural (conocimiento acerca de cómo actuar) expresado mediante un sistema de producción conteniendo reglas procedurales del tipo *si-entonces* similares a las del GPS. El sistema de producción selecciona aquella regla cuya condición se cumple en función del estado del sistema y la aplica. El estado del sistema se modifica en tiempo real a través de los módulos de atención visual y movimiento motor, así como del contenido de la memoria declarativa. ACT-R se ha usado por cientos de investigadores para, entre otros usos, modelar cómo un ser humano resuelve problemas tales como las Torres de Hanoi, la resolución de ecuaciones algebraicas o la comprensión del lenguaje.

SOAR es otra arquitectura cognitiva clásica, desarrollada por John Laird a finales de los ochenta, fuertemente inspirada en el GPS. En SOAR la resolución de un problema se basa en la búsqueda, dentro del espacio de posibles soluciones, de un estado objetivo que representa una solución al problema. La estrategia consiste en aplicar las reglas de producción para ir alcanzando estados subobjetivos hacia los cuales moverse con el fin de acercarse gradualmente al estado “solución final”. De hecho, esta estrategia de búsqueda es muy similar a la que utilizaba el Logic Theorist[34]. SOAR, además, aprende por experiencia, guardando las trazas de las soluciones que ha encontrado durante la resolución de problemas, de forma que pueda volver a utilizarlas en futuros problemas análogos; en este sentido, podemos

decir que guarda relación con el razonamiento basado en casos descrito en el capítulo 4. A lo largo de los años se ha desarrollado una serie de arquitecturas SOAR, desde SOAR1 a mediados de los años ochenta hasta SOAR9 en 2008, cada una de ellas introduciendo mejoras a la anterior. Sin duda SOAR y ACT-R son las arquitecturas cognitivas más importantes, pero no las únicas; posteriormente se han propuesto otras, como, por ejemplo, ICARUS, de Pat Langley, en 2006, que, entre otras cosas, también se basa en el concepto de sistema de producción POLYSCHEME, de Nicholas Cassimatis, también en 2006. Este, entre otros aspectos, usa el concepto de descomposición en subobjetivos y COMPANION ARCHITECTURE de Ken Forbus en 2009, que se basa principalmente en el razonamiento y aprendizaje por analogía. De todas ellas, COMPANION es posiblemente la más singular, ya que no pretende dar lugar a la construcción de sistemas inteligentes completamente autónomos, sino, como su nombre indica, sistemas de ayuda que colaboren con los usuarios para resolver problemas complejos; por ejemplo, recuperando de su memoria precedentes similares, ayudando a tomar decisiones mostrando argumentos a favor o en contra de posibles decisiones alternativas, etc.

Además de estos desarrollos de arquitecturas cognitivas, una buena parte de las actuales investigaciones en robótica también son relevantes en tanto que sistemas integrados, ya que incluyen componentes de razonamiento, planificación, aprendizaje, comunicación, percepción y actuación. Los robots constituyen sin duda una plataforma excelente para la investigación en sistemas integrados, un paso imprescindible hacia la IA de tipo general de la que-hablaremos en el siguiente apartado.

¿Hacia una IA de tipo general?

En la introducción hemos afirmado que el objetivo último de la IA es lograr que una máquina temga una inteligencia de tipo general. Objetivo que, por su dificultad, es comparable a otros grandes objetivos de la ciencia como explicar el origen de la vida, el origen del universo o conocer la estructura de la materia. A lo largo de los capítulos de este libro esperamos haber dejado claro que, con la excepción de los primeros años, prácticamente todos los esfuerzos de la investigación en IA se han centrado en construir inteligencias artificiales especializadas. También esperamos haber convencido al lector de que los éxitos alcanzados en solamente 60 años de existencia son impresionantes, y en particular durante el último decenio, gracias a la conjunción de dos elementos: la disponibilidad de enormes cantidades de datos y el acceso a la computación de altas prestaciones. Efectivamente, el éxito de sistemas como, por ejemplo, AlphaGo, Watson y los vehículos autónomos ha sido posible gracias a esta conjunción y también a un poco de *ciencia nueva* (de hecho, los conceptos fundamentales que rigen el aprendizaje profundo tienen unos 20 años de antigüedad). Sin embargo, prácticamente no hemos avanzado hacia la consecución de IA general.

De hecho, posiblemente la lección más importante que hemos aprendido a lo largo de los 60 años de existencia de la IA es que lo que parecía más difícil (diagnosticar enfermedades, jugar al ajedrez y a Go al más alto nivel) ha sido realizable y lo que parecía más fácil ha resultado ser lo más difícil.

La explicación a esta aparente contradicción hay que buscarla en la dificultad de dotar a las máquinas de conocimientos de sentido común. A lo largo de varios capítulos del libro hemos hablado de la importancia de poder dotar a la máquina de conocimiento de sentido común y hemos argumentado que, sin estos conocimientos, no es posible una comprensión profunda del lenguaje ni una interpretación profunda de lo que capta un sistema de percepción visual, entre otras limitaciones. De hecho, es requisito fundamental para conseguir IA similar a la humana en cuanto a generalidad y profundidad. Los conocimientos de sentido común son fruto de nuestras vivencias y experiencias interactuando con nuestro entorno, ya que la cognición humana es una cognición situada y corpórea. Las aproximaciones no corpóreas no permiten interacciones directas con el entorno, por lo que, inevitablemente, dan lugar a falsos problemas y, por lo tanto, a falsas soluciones. Tienden a definir los problemas en términos de tareas en un entorno especificadas desde una perspectiva abstracta de objetos y relaciones. Las capacidades cognitivas no se deberían estudiar haciendo abstracción del sistema sensor y el sistema motor. Las capacidades más complicadas de alcanzar son aquellas que requieren interaccionar con entornos no restringidos ni previamente preparados. Diseñar sistemas que tengan estas capacidades requiere integrar desarrollos en muchas áreas de la IA. En particular, necesitamos lenguajes de representación del conocimiento que codifiquen información acerca de muchos tipos distintos de objetos, situaciones, acciones, etc., así como de sus propiedades y de las relaciones entre ellos. También necesitamos nuevos algoritmos que, basándose en estas representaciones, puedan responder, de forma robusta y eficiente, a preguntas sobre prácticamente cualquier tema. Finalmente, dado que necesitarán conocer un número prácticamente ilimitado de cosas, estos sistemas deberán ser capaces de aprender nuevos conocimientos de forma continua a lo largo de toda su existencia. En definitiva, tal como hemos argumentado en la sección anterior, es imprescindible diseñar sistemas que integren percepción, representación, razonamiento, acción y aprendizaje.

Entre las actividades futuras, creemos que los temas de investigación más importantes seguirán estando basados en lo que se conoce por *massive data-driven AI*, es decir, en explotar la posibilidad de acceder a cantidades masivas de datos y poder procesados con *hardware* cada vez más rápido con el fin de descubrir relaciones entre ellos, detectar patrones y realizar inferencias y aprendizaje mediante modelos probabilísticos, como, por ejemplo, los sistemas de aprendizaje profundo (LeCun *et al*., 2015). Sin embargo, estíos sistemas basados en el análisis de enormes cantidades de datos deberán, en el futuro, incorporar módulos que permitan explicar cómo se ha llegado a los resultados y conclusiones propuestas, ya que la capacidad de

explicación es una característica irrenunciable en cualquier sistema inteligente. Actualmente, la principal limitación de los sistemas basados en aprendizaje profundo es que son *cajas negras* sin capacidad explicativa, por ello un objetivo interesante de investigación será como dotar de capacidad explicativa a los sistemas de aprendizaje profundo y otros sistemas de IA basados en el procesamiento de datos masivos. Otras técnicas más clásicas de IA que seguirán siendo objeto de investigación extensiva son los sistemas multia-gente, la planificación de acciones, el razonamiento basado en la experiencia, la visión artificial, la comunicación multimodal persona-máquina, la robótica humanoide y animaloide y en particular las nuevas tendencias en robótica del desarrollo, que puede ser la clave para dotar a las máquinas de sentido común. También veremos progresos significativos gracias a las aproximaciones biomiméticas para reproducir en máquinas el comportamiento de animales. No se trata únicamente de reproducir el comportamiento de un animal, sino de comprender cómo funciona su cerebro. Se trata de construir y programar circuitos electrónicos que reproduzca la actividad cerebral que genera este comportamiento. Algunos biólogos están interesados en los intentos de fabricar un cerebro artificial lo más complejo posible, porque es una manera de comprender mejor el órgano, y los ingenieros buscan información biológica para hacer diseños más eficaces. Mediante la biología molecular y los recientes avances en optogenética será posible identificar qué genes y qué neuronas juegan un papel en las distintas actividades cognitivas.

Otra importante fuente de inspiración para la IA es la ciencia de materiales, y en particular los nanomateriales. Por ejemplo, para el desarrollo de músculos artificiales una posible tecnología consiste en intercalar capas de caucho de silicio con capas de polímero electro-activo de tal forma que el conjunto flexiona al aplicar un campo eléctrico. También hay resultados interesantes en cartílagos artificiales construidos mediante filamentos de polímeros con moléculas atractoras de agua para mimetizar las propiedades lubricantes de los cartílagos naturales. Finalmente, se están usando compuestos de caucho de silicona cargado con nanopartículas de níquel para pieles artificiales, ya que la resistencia del compuesto disminuye con la presión a que es sometido, o también sensores capacitivos, es decir, cuya capacidad cambia con la presión. Esta aproximación pluridisciplinar a la IA mimetizando a la biología y el uso de resultados en ciencia de materiales puede producir un efecto sinèrgico que cambie profundamente la naturaleza de la IA e incluso quizá nuestra comprensión de qué es la inteligencia.

En cuanto a las aplicaciones, algunas de las más importantes seguirán siendo aquellas relacionadas con la web, los videojuegos y los robots autónomos (en particular vehículos autónomos, robots sociales, robots para la exploración de planetas, etc.). Las aplicaciones para el medioambiente y el ahorro energético también serán importantes, así como para la economía y la sociología.

Por último, las aplicaciones de la IA al arte (artes visuales, música, danza, narrativa) cambiarán de forma importante la naturaleza del proceso creativo. Los

ordenadores ya no son solamente herramientas de ayuda a la creación, empiezan a ser agentes creativos. Ello ha dado lugar a una nueva y muy prometedora área de aplicación de la inteligencia artificial denominada *creatividad computacional* que ya ha producido resultados muy interesantes (Colton *et al*., 2009) en música, artes plásticas y narrativa, entre otras actividades artísticas.

En cualquier caso, por muy inteligentes que lleguen a ser las futuras inteligencias artificiales, incluidas las de tipo general, nunca serán iguales a las inteligencias humanas, pues, tal como hemos argumentado, el desarrollo mental que requiere toda inteligencia compleja depende de las interacciones con el entorno, y estas dependen a su vez del cuerpo, en particular del sistema perceptivo y del sistema motor. Ello, junto al hecho de que las máquinas no seguirán procesos de socialización y culturización como los nuestros, hace que, por muy sofisticadas que lleguen a ser, serán inteligencias distintas a las nuestras. El hecho de ser inteligencias ajenas a la humana y, por lo tanto, ajenas a los valores y necesidades humanas nos debería hacer reflexionar sobre posibles limitaciones éticas al desarrollo de la IA. En particular, estamos de acuerdo con Weizenbaum (1976) en que ninguna máquina debería nunca tomar decisiones de forma completamente autónoma o dar consejos que requieran, entre otras cosas, de la sabiduría, producto de experiencias humanas, y de los valores humanos. Ejemplos claros de usos de la IA que no deberían permitirse por motivos éticos son las armas autónomas, los *bots* que operan en bolsa tomando decisiones de compra y venta en mili-segundos y las aplicaciones que atenían a nuestra privacidad.

El camino hacia la IA de tipo general seguirá siendo largo y difícil; al fin y al cabo la IA tiene solo 60 años y, como diría Carl Sagan, 60 años son un brevísimo momento en la escala cósmica del tiempo; o, como muy poéticamente dijo Gabriel García Márquez:

Desde la aparición de vida visible en la Tierra debieron transcurrir 380 millones de años para que una mariposa aprendiera a volar, otros 180 millones de años para fabricar una rosa sin otro compromiso que el de ser hermosa, y cuatro eras geológicas para que los seres humanos fueran capaces de cantar mejor que los pájaros y morirse de amor.

Bibliografía

ALLEN, J. (1995): *Natural language understandings*, Benjamin Cummings, San Francisco.

BIERE, A.; HEULE, M.; VAN MAAREN, H. y WALSH, T (2009): *Handbook of Satisfiability*, IOS Press, Amsterdam.

BLUM, C. y ROLI, A. (2003): "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison", *ACM Computing Surveys*, 35, pp. 268-308.

BOBROW, D. y COLLINS, A. (eds.) (1975): *Representation and understanding: Studies cognitive science*, Academic Press, Cambridge, MA.

BROWNE, C.; POWLEY, E.; WHITEHOUSE, D.; LUCAS, S.; COWLING, P. I.; ROHLFSHAGEN, P; TAVENER, S.; PÉREZ, D.; SAMOTHRAKIS, S. y COLTON, S. (2012): "A Survey of Monte Carlo Tree Search Methods", *IEEE Trans. Computational Intelligence and Al in games*, 4(1), pp. 1-49.

COLTON, S.; LÓPEZ DE MÁNTARAS, R. y STOCK O. (2009): "Computational Creativity: Coming of Age" *AI Magazine*, 30(3), pp. 11-14.

DE LEÓN., M. y TIMÓN, A. (2013): *Rompiendo códigos. Vida y legado ele Turing*, colección ¿Qué sabemos de?, Los Libros de la Catarata-CSIC, Madrid.

DECHTER, R. (2003): *Constraint Processing*, Morgan Kaufmann, San Francisco.

DREYFUS, H. L. (1992): *What Computers Still Can't Do: A Critique of Artificial Reason*, The MIT Press, Cambridge, MA.

FAUGERAS, O. (1993): *Three-dimensional Computer Vision (Artificial Intelligence)*, The MIT Press, Cambridge, MA.

FEIGENBAUM, E. A. y FELDMAN, J. (eds.) (1963): *Computers and Thought*, McGraw-Hill, Nueva York, pp. 134-152.

GHALLAB, M.; NAU, D. y TRAVERSO, P. (2004): *Automated Planning: Theory and Practice*, Morgan Kaufmann, San Francisco.

GARCÍA-ARMADA, E. (2015): *Robots*, colección ¿Qué sabemos de?, Los Libros de la Catarata-CSIC, Madrid.

GINSBERG, M. (1993): *Essential of Artificial Intelligence*, Morgan Kaufmann, San Francisco.

HODGES, A. (2012): *Alan Turing: The Enigma*, Vintage, Random House, Londres.

KOLODNER, J. (1993): *Case-Based Reasoning*, Morgan Kaufmann, San Francisco.

KOWALSKI, R. A. (1980): *Logic for Problem Solving*, North Holland Ámsterdam.

LECUN,Y.; BENGIO, Y. y HINTON, G. (2015): "Deep Learning", *Nature*, 521, pp. 436-444.

LÓPEZ DE MÁNTARAS, R. (1991): *Approximate Reasoning Models*, Ellis Horwood Amsterdam.

LÓPEZ DE MÁNTARAS, R.; MCSHERRY, D.; BRIDGE, D.; SMYTH, B.; CRAW, S.; FALTINGS, B.; MAHER, M. L.; COX, M.; FORBUS, K.; KEANE, M.; AAMODT, A. y WATSON, I. (2006): "Retrieval, reuse, revision, and retention in CBR", *Knowledge Engineering Review*, 20(3), pp. 215-240.

MANNING, C. D. y SCHÜTZE, H. (1999): *Foundations of Statistical Natural Language Processing*, The MIT Press, Cambridge, MA.

MARR, D. (1982): *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, W. H. Freeman and Co, San Francisco.

MITCHELL, T. (1997): *Machine Learning*, McGraw-Hill, Nueva York.

NEWELL, A. y SIMON, H. A. (1972): *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, NJ.

NEWMAN, A. (2007): *The Correspondence Theory of Truth*, Cambridge Studies in Philosophy, Cambridge University Press, Cambridge.

PEARL, J. (1984): *Heuristics: Intelligent Search Strategies for Composed Problem Solving*, Addison-Wesley, Boston, MA.

—(1988): *Probabilistic Reasoning Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Francisco.

ROSSI, F.; VAN BEEK, P. y WALSH, T (eds.) (2006): *Handbook of Constraint Programming*, Elsevier, Nueva York.

RUSSELL, S. y NORVIG, P. (2010): *Artificial Intelligence: A Modern Approach*, 3.ª ed., Prentice Hall, Nueva Jersey.

SCHANK, R. C. y ABELSON, R. P. (1977): *Scripts, Plans, Goals, an Understanding: An Inquiry into Human Knowledge Structures*, Lawrence Erlbaum Associates, Hillsdale, NJ.

SEARLE, J. R. (1980): "Minds, Brains, and Programs", Behavioral and Brain Sciences, vol. 3, pp. 417-457.

SHORTLIFFE, E. H. (1975): *Computer-Based Medical Consultations: MYCIN*, Elsevier, Nueva York.

SICILIANO, B. y KHATIB, O. (eds.) (2016): *Springer Handbook of Robotics*, 2.ª ed., Springer, Berlín.

SUTTON, R. y BARTO, A. G. (1998): *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.

THORNDIKE, E. L. (1911): *Animal Intelligence*, The Macmillan Co, Nueva York.

TORRAS, C. (2016): “Service robots for citizens of the future”, *European Review*, 24(1), pp. 17-30.
TURING, A. M. (1950): “Computing Machinery and Intelligence”, *Mind*, vol. LIX (236), pp. 433-460.
VAN HARMELEN, E; LIFSCHITZ, V. y PORTER, B. (eds.) (2008): *Handbook of Knowledge Representation*, Elsevier, Nueva York.
WEIZENBAUM, J. (1976): *Computer Power and Human Reasoning: From Judgment to Calculation*, W. H. Freeman and Co, San Francisco.
WINOGRAD, T. (1972): *Understanding Natural Language*, Academic Press, Cambridge, MA.
—(1983): *Language as a Cognitive Process*, vol. 1, Syntax, Addison-Wesley, Boston, MA.
WINSTON, P. H. (ed.) (1975): *The psychology of computer vision*, McGraw-Hill, Nueva York.
WOOLDRIDGE, M. (2009): *An Introduction to Multiagent Systems*, 2.ª ed., John Wiley & Sons, Nueva York.
ZADEH, L. (1965): “Fuzzy Sets”, *Information and Control*, vol. 8, pp. 338-353.

RAMON LÓPEZ DE MÁNTARAS BADIA es profesor de Investigación del CSIC y director del Instituto de Investigación en Inteligencia Artificial. Máster en Ingeniería Informática por la Universidad de California Berkeley, doctor en Física (Control Automático) por la Universidad de Toulouse y en Ingeniería Informática por la Universidad Politécnica de Barcelona. Es pionero de la inteligencia artificial (IA) en España.

Escritos: *El futuro de la IA: hacia inteligencias artificiales realmente inteligentes*, 27 de febrero de 2019; *La inteligencia artificial y las artes. Hacia una creatividad computacional*, 31 enero 2017.

PEDRO MESEGUER GONZÁLEZ es Investigador Científico del Consejo Superior de Investigaciones Científicas (CSIC) en el Instituto de Investigación en Inteligencia Artificial, en donde es jefe del departamento de Lógica. Doctor en informática por la Universidad Politécnica de Cataluña, fue Profesor Titular de la Facultad de Informática de Barcelona.

Ha publicado numerosas contribuciones científicas en las conferencias y revistas más prestigiosas de IA sobre varios temas, con especial énfasis en el razonamiento con restricciones. Ha recibido dos veces el premio al mejor artículo en dos conferencias de IA. Es coautor de un capítulo del *Handbook of Constraint Programming*.

Notas

[1] Sobre la vida de Turing, véase Hodges (2012). Con motivo del centenario de su nacimiento, en 2012 se realizó un blog en el periódico El País sobre Turing. Con 40 entradas, este blog contiene mucha información (http://blogs.elpais.com/turing/). Véase también el libro sobre Turing de De León y Timón (2013). <<

[2] Véase el apartado “Los primeros lenguajes de programación” en este capítulo. <<

[3] Véase “Búsqueda heurística” en el capítulo 2. <<

[4] Véase “La explosión combinatoria” en el capitulo 3. <<

[5] Ibíd. <<

[6] Véase el apartado “Representación del conocimiento” en el capítulo 2. <<

[7] Véase el apartado “Sistemas expertos” en el capítulo 2. <<

CAPÍTULO 2

[8] Ibíd. <<

[9] Para una visión más profunda, véase Van Harmelen et al. (2008). <<

[10] Véase el apartado “Planificación” en este mismo capítulo. <<

[11] Véase el apartado “Búsqueda heurística” en este mismo capítulo. <<

[12] Como el de los apartados “Planificación” y “Visión por computador” en este mismo capítulo. <<

CAPÍTULO 3

[13] Véase el apartado “Los primeros intentos de procesar el lenguaje natural” del capítulo 1. <<

[14] En el apartado “¿Hacia una inteligencia artificial de tipo general?” del capítulo 6 utilizamos argumentos similares para plantear posibles limitaciones éticas a la IA. <<

[15] Para más detalles, véase Biere *et al*. (2009). <<

[16] Véase el apartado “Sistemas expertos” en el capítulo 2. <<

[17] La bibliografía sobre restricciones publicada en los últimos 40 años es muy amplia. Para una visión homogénea del área, se recomienda Rossi et al. (2006). <<

[18] Véase “Búsqueda heurística” en el capítulo 2. <<

[19] Los juegos por ordenador se basan en estrategias de búsqueda. Para una visión actualizada véase el apartado “Juegos por ordenador” del capítulo 5. <<

[20] Un buen resumen de estos métodos, denominados globalmente metaheurísticos, se encuentra en Blum y Roli (2003). <<

[21] Para una visión actualizada, véase Russell y Norvig (2010) y Edelkamp y Scrödl (2012). <<

[22] Véase el apartado “Aprendizaje automático” en el capítulo 2 y en este capítulo. <<

[23] Ibíd. <<

[24] Véase el apartado “Planificación” en el capítulo 2. <<

[25] Véase el apartado “Nuevas estrategias de búsqueda” en este mismo captado. <<

[26] Para profundizar en el tema de planificación, véase Ghallab *et al*. (2004). <<

[27] Véase “Búsqueda heurística” en el capítulo 2. <<

[28] 28. Véase el apartado “Redes bayesianas” en este mismo capítulo. <<

[29] Véase el apartado “Nuevas estrategias de búsqueda” en este mismo capítulo. <<

[30] En el apartado “Aprendizaje automático” de este mismo capítulo se describe brevemente este proceso en el caso particular de reconocimiento de sillas. <<

[31] En este apartado nos centraremos en aquellos aspectos de la robótica que están estrechamente relacionados con la IA, es decir, los robots autónomos. Para otros aspectos de la robótica, incluyendo robots industriales y exoesqueletos, véase García-Armada (2015). <<

[32] Para más detalle acerca de robots de servicios, véase Torres (2016). <<

[33] Para otros éxitos de la robótica no relacionados con la IA, de nuevo remitimos al lector al libro de García-Armada (2015). <<

[34] Véase el apartado “Los primeros programas” del capítulo 1. <<

Inteligencia artificial

Ramon López de Mántaras Badia
y Pedro Meseguer González