Complejidad e Inteligencia Artificial

Juan Giró

Resumen

Se revisan las definiciones de procedimiento y de algoritmo, para luego presentar los dos interrogantes fundamentales asociados a la solución de un problema, que se refieren la existencia de un procedimiento y en caso afirmativo al costo de su aplicación. Concentrando la atención en este último, se presenta la necesidad de conocer la complejidad computacional de los problemas y la búsqueda de algún indicador de su complejidad intrínseca. Se definen la complejidad temporal y la complejidad espacial, se relacionan estos conceptos con la Máquina de Turing y de identifican "clases" de problemas según los esfuerzos que requieren su solución. Finalmente se considera la complejidad de algunas herramientas de la Inteligencia Artificial.

1. Introducción

Desde un punto de vista informático, intuitivamente se asocia la *complejidad* con los recursos de cómputo requeridos para resolver un problema. Una rama de la Ciencia de la Computación denominada Teoría de la *Complejidad Computacional* se ocupa de estudiar estos recursos, que son normalmente tiempo y espacio.

La Teoría de la *Complejidad Computacional* siempre aparece vinculada a la *Teoría de la Computabilidad*, aunque sus objetivos son esencialmente distintos. El foco de la *Teoría de la Computabilidad* está en la existencia, o no, de procedimientos que permitan resolver cierto problema. Es decir, la *Teoría de la Computabilidad* deja de lado los aspectos referidos a la implementación y ejecución de los sistemas computacionales y se concentra en la comprobación de la factibilidad de la existencia de tal solución. En caso afirmativo se dice cierta función es computable o calculable.

La *Complejidad Computacional* fue siempre motivo de interés. En los comienzos de la era de la computación esto fue una consecuencia natural de los magros recursos disponibles: escasa memoria y reducida capacidad de proceso. Más recientemente y en la actualidad, los avances de la tecnología vienen incrementado en forma sostenida la capacidad y calidad de los recursos de computación. Sin embargo, paralelamente, los objetivos de los sistemas informáticos son cada vez más ambiciosos y el interés por la complejidad computacional mantiene así plena vigencia.

La *Inteligencia Artificial* no es ajena a esta realidad. Renovados objetivos y el permanente descubrimiento, desarrollo y aplicación de nuevas técnicas y herramientas han conducido progresivamente a un incremento de la complejidad de los sistemas involucrados. Entre otros, pueden mencionarse las técnicas de búsquedas heurísticas de tiempo real, los juegos con animación, algoritmos genéticos, sistemas evolutivos, redes neuronales artificiales, autómatas celulares, minería de datos, etc, por solo mencionar algunos de los más importantes. En todos estos casos la *complejidad computacional* es uno de los principales factores a ser considerados.

2. Procedimiento y algoritmo

Se convendrá en denominar *procedimiento* a todo conjunto finito y ordenado de instrucciones, de forma tal que puedan ser ejecutadas por una persona o una máquina sin necesidad de conocimiento adicional a lo ya expresado en las propias sentencias. Esto implica que un problema será considerado resuelto cuando se encuentre una forma mecánica de operar, paso a paso, a partir de sus datos hasta alcanzar un resultado. Puede entonces decirse que se ha definido un procedimiento para alcanzar la solución y obsérvese que se trata de un procedimiento mecánico primitivo, definido en sí mismo.

Así, la búsqueda de un procedimiento es siempre el eje en la solución de todo problema computacional, y esta búsqueda estará necesariamente asociada a dos interrogantes fundamentales: 1) ¿existirá tal procedimiento? y 2) en caso de existir, ¿cuál será el costo computacional de su aplicación?

Como ya fue anticipado, el primero de estos dos interrogantes está asociando al concepto de computabilidad y al segundo el de complejidad. Este breve trabajo se ocupará de este último y para ello es necesario comenzar por formalizar las ideas de procedimiento y de algoritmo.

Un caso particular está representado por aquellos procedimientos que permiten alcanzar siempre una solución en un número finito de pasos: estos son definidos *algoritmos* o *procedimientos efectivos*. La noción de *algoritmo* es la idea básica de la Ciencia de la Computación y una de las cuestiones centrales para muchas ramas de las matemáticas.

Aquí debe anticiparse que habrá problemas para los cuales no es posible encontrar un procedimiento y otros problemas para los que existen procedimientos pero no algoritmos. Más aún, que la solución de un problema quede representada por un procedimiento o por un algoritmo puede no ser intrínsico del propio problema y en muchos casos tan solo depender de los datos.

Como ejemplo de esto último puede considerarse el caso del cálculo de la raíz cuadrada de números naturales. Para aquellos números que son cuadrado perfecto (4, 25, 121, 144, 256, etc) se define un procedimiento efectivo o algoritmo para la obtención de esas raíces, mientras que para el resto (2, 3, 11, etc) el procedimiento demanda infinitos pasos, ya que por tratarse de números irracionales, sería necesario calcular los infinitos decimales que representan la solución. No se trata en este último caso de soluciones algorítmicas.

Debe ahora reconocerse que un procedimiento (efectivo o nó) puede presentarse de muy diversas maneras: en su forma más general una secuencia mecánica de instrucciones es una Máquina de Turing, pero en el otro extremo una simple fórmula representa también esta noción de procedimiento. Nótese que los conceptos de procedimiento y de algoritmo no están asociados a problemas de cierta complejidad, ya que ambos términos estarán representados tanto en secuencias de instrucciones muy simples como en otras verdaderamente complejas. El concepto de complejidad es independiente y será tratado más adelante.

Lo hasta aquí expresado queda resumido en una definición de Hopcroft y Ullman [1]: “Un procedimiento es una secuencia finita de instrucciones que pueden ser cumplidas en forma mecánica, como es el caso de un programa de computadora. Un procedimiento que siempre termina lleva el nombre propio de algoritmo”.

Es también importante destacar que la noción de procedimiento no puede estar desvinculada del medio en que se opera, y lo que es considerado un procedimiento en

ciertas circunstancias puede no serlo en otras. Como ejemplo puede citarse la descripción de una partida de ajedrez: se trata de un algoritmo perfectamente claro para un jugador que pretende reproducir las jugadas y sin embargo estas mismas instrucciones puede carecer completamente de sentido para algún otro individuo sin conocimientos de ajedrez. Por lo tanto, cuando se hace referencia a un conjunto finito de instrucciones se sobrentiende que el encargado de interpretarlas, ya sea hombre o máquina, conoce las convenciones adoptadas en su definición.

3. Concepto de complejidad

Intuitivamente, se considera a un problema *complejo* si su resolución requiere el empleo de un algoritmo *complejo*. Este último es a su vez complejo si su aplicación involucra cálculos complicados y/o su lógica subyacente es complicada. En consecuencia, el concepto de *complejidad* no parece ser cuantificable o medible, sino más bien subjetivo. Inclusive la *complejidad* de un mismo problema parece cambiar a medida que el encargado de resolverlo mejora su capacitación, adopta una actitud apropiada y, mas aún, el nivel de complejidad puede variar significativamente dependiendo de la persona que emita el juicio.

El objetivo sería entonces establecer una escala que clasifique los problemas computacionales, permitiendo establecer medidas o "clases" de complejidad. Para ello, se debe comenzar por encontrar un significado al hecho que un problema sea más complejo que otro, dejando de lado todos los aspectos subjetivos que puedan afectar esta evaluación.

En síntesis, se pretenderá buscar algún indicador de la complejidad intrínseca de los problemas, procurando prescindir de la habilidad de la persona que deba resolverlos, de los algoritmos que puedan ser utilizados y de las características de los medios de cálculo disponibles. Y finalmente, cuando no sea posible prescindir de estos factores, poder establecer las bases que permitan definir métricas y hacer comparaciones.

Para la definición de estas métricas es necesario establecer los siguientes tres aspectos:

a. Identificar los parámetros primitivos que serán medidos para disponer de datos característicos del objeto estudiado.
b. Definir las propias métricas, representadas por expresiones destinadas a determinar indicadores objetivos que representen niveles de complejidad.
c. Establecer las condiciones en que se harán las mediciones, con el fin de asegurar que diferentes evaluaciones de complejidad tengan una referencia común y sean verdaderamente comparables.

Una vez definidas las bases que permitan determinar correctamente la complejidad de los problemas, se las aplicará en las próximas secciones al estudio de casos teóricos y prácticos.

4. Medidas y métricas de la complejidad

El enfoque que aquí se propone es el de determinar indirectamente la complejidad de un problema a partir de la medición de los recursos necesarios para resolverlo. Obsérvese que es una determinación indirecta, ya que en realidad no se evalúa la complejidad del propio problema, sino más bien los recursos demandados correspondientes al procedimiento empleado. Esta propuesta se apoya en la obvia convicción que la solución de un problema de elevada complejidad demandará más recursos que la de un problema sencillo.

De acuerdo a esta idea y a lo postulado en el punto anterior referente a la elección de parámetros que representen la complejidad, se seleccionan al Tiempo y Espacio como los indicadores más representativos.

4.1 Complejidad Temporal

Uno de los indicadores de complejidad más utilizados es el tiempo. Se refiere a la cantidad de intervalos o unidades elementales que demanda completar la ejecución de un proceso. Este indicador lleva a la definición de la *complejidad temporal* del problema, expresado en función del tamaño del problema o de sus datos.

Es así que se procurará conocer la razón de crecimiento entre el indicador tiempo y la dimensión de los datos, que representa el parámetro medible. Como resultado se hablará de *complejidad temporal* lineal, polinómica, logarítmica, exponencial, etc, según la naturaleza de la expresión que las vincula. Por ejemplo, si las unidades de tiempo que demanda la solución de un problema de dimensión "n" es $T(n) = 3.n^3 + 6.n + 12$, se dirá que su complejidad temporal es polinómica, en este caso de grado tres.

El límite en el crecimiento de esta medida se denomina complejidad temporal asintótica y es finalmente lo que determina el tamaño del problema que puede ser resuelto con cierto algoritmo. Tómese como ejemplo un algoritmo que procesa un conjunto de datos de entrada de tamaño "n" en un tiempo $C.n^2$, donde "C" representa una constante que es propia del problema: su complejidad temporal es de órden n^2 y esto se representa como $O(n^2)$. En forma general, si se dice que una función g(n) tiene un orden de complejidad temporal O (f (n)) si existe una constante "C" tal que

$$g(n) \leq C\, f(n)$$

para todos los posibles valores positivos de "n".

La selección del tiempo como indicador representativo de la complejidad de un problema no requiere mayores justificaciones. La búsqueda de la reducción de los tiempos de proceso es tan antigua como la misma ciencia de la computación y los ejemplos sobran: parece obvio adoptar una estrategia de búsqueda binaria para operar sobre una lista de gran tamaño en lugar de usar una búsqueda secuencial, por citar solo uno de los casos más comunes.

Podría argumentarse que el tremendo incremento en la rapidez de cálculo que se viene verificando en los computadores digitales modernos llevará a relativizar la importancia de emplear algoritmos eficientes. Es sin embargo igualmente cierto el razonamiento inverso: a medida que los computadores son más rápidos se pretende resolver problemas de mayor tamaño en un contexto de nuevas exigencias (gráficos, animación, tiempo real, etc) y la necesidad de algoritmos eficientes mantiene plena vigencia o cobra aún más importancia que antes.

Tabla 1: Complejidad Temporal de ciertos Algoritmos seleccionados

Algoritmo	Orden de complejidad Temporal T(n)
A_1	n
A_2	n log n
A_3	n^2
A_4	n^3
A_5	2^n

Definida la complejidad temporal como el número de unidades de tiempo requeridas para procesar una entrada de dimensión "n", en la Tabla 1 se presentan los valores resultantes de cinco algoritmos diferentes.

Aho, Hopcroft y Ullman [2] utilizan este ejemplo para ilustrar las importantes diferencias entre ellos. Se asume que con cierta tecnología la unidad de tiempo es el milisegundo y se toma como referencia al algoritmo A_1 que tiene una complejidad temporal lineal. Esto significa que el algoritmo A_1 requiere un segundo para procesar un lote de datos de dimensión n = 1000, requiere un minuto para procesar n = 60.000 y requiere una hora de procesamiento para n = 3.600.000. La Tabla 2 muestra los tamaños de los lotes de datos que pueden ser procesados por los diferentes algoritmos en esos mismos tres intervalos de tiempo:

Tabla 2: Dimensión "n" del problema resuelto por cada Algoritmo en cierto tiempo

Algoritmo	Máximo Tamaño de lote de datos "n"		
	1 seg	60 seg	3600 seg
A_1	1.000	60.000	3.600.000
A_2	140	4.893	200.000
A_3	31	244	1.897
A_4	10	39	153
A_5	9	15	21

Se considera ahora que la rapidez de cálculo del computador se incrementó diez veces con respecto a la del equipo utilizado en el caso anterior e interesa conocer sus consecuencias. Es decir, se busca determinar el incremento del tamaño de problema que puede ser resuelto por cada algoritmo en el mismo intervalo de tiempo de un segundo. La Tabla 3 muestra estos incrementos y se representan con los símbolos "x" y "+" los casos de un factor o de un incremento constante, respectivamente.

Tabla 3: Consecuencia de un medio de cálculo 10 veces más rápido

Algoritmo	Incremento en el tamaño "n" del problema que es procesable en 1 seg.
A_1	x 10,0
A_2	x 10,0 (aprox.)
A_3	x 3,16
A_4	x 2,15
A_5	+ 3,30

Se sugiere al lector que reflexione sobre las ventajas que pueden esperarse con los diferentes algoritmos cuando se incrementa el intervalo de tiempo de proceso disponible o la capacidad de la máquina usada, comparando los tamaños de las muestras que pueden ser procesadas en esos casos.

4.2 Complejidad Espacial|

El otro indicador que se emplea con frecuencia como medida de la complejidad de un problema es la cantidad de espacio de almacenamiento requerido para resolverlo a través de cierto algoritmo. Esto se apoya en la idea de que al aumentar la complejidad de un proceso aumenta también el espacio que demanda y este parámetro es reconocido como la *complejidad espacial* del problema. Esta *complejidad espacial* definirá la razón de

crecimiento entre el espacio requerido y los datos del problema, y se representa por E(n). Al igual que en el caso de la complejidad temporal, el límite en el crecimiento de esta medida se denomina *complejidad espacial asintótica* y determinará finalmente el tamaño del problema que puede resolver cierto algoritmo.

5. Medición de la complejidad y Máquina de Turing

Puede fácilmente comprobarse que la *Complejidad Temporal* y *Complejidad Espacial* demandada por la resolución de un problema dependerá del soporte físico donde el proceso es realizado. En efecto, la *Complejidad Temporal* de un problema medida sobre un computador dependerá de su arquitectura (8 bits, 16 bits, 32 bits, etc), rapidez del microcomputador, disponibilidad de coprocesador aritmético para operaciones de punto flotante, rapidez de los canales de I/O , etc. Similarmente, la *Complejidad Espacial* también dependerá de la arquitectura del equipo, forma de codificación y representación de los datos (binario, bcd, ascii, etc) entre otros factores.

Para eliminar esta dependencia entre las mediciones de los indicadores de complejidad y el medio de cálculo utilizado las comparaciones deben ser realizados en un contexto fijo. Para ello se adopta la Máquina de Turing, que no solo proporciona un ambiente bien definido donde trabajar sino que además las conclusiones que se obtengan son fácilmente transferibles a otros sistemas computacionales

Una de las primeras medidas de complejidad para Máquinas de Turing fue la propuesta por Shannon en 1956, que apuntaba a la complejidad estructural de la Máquina y proponía como parámetro al producto del número de estados por el número de símbolos [3]. Si bien este enfoque ha sido superado por las medidas de espacio y tiempo ya enunciadas, es de gran interés por las técnicas de simulación que quedan evidenciadas en los siguientes teoremas:

Teorema 1 de Shannon

Cualquier maquina de Turing con "m" símbolos y "n" estados puede ser simulada por otra maquina de Turing con exactamente dos estados y 4.m.n + m símbolos. En particular, existe una Máquina Universal de Turing de solo dos estados. Además, toda Máquina Universal de Turing necesita al menos dos estados.

Teorema 2 de Shannon

Cualquier maquina de Turing con "m" símbolos y "n" estados puede ser simulada por otra con exactamente dos símbolos y menos de 8.m.n estados.

Definición de Complejidad en la Máquina de Turing

Retomando las medidas de complejidad propuestas, en una Máquina de Turing se define a la complejidad temporal como el número de pasos requerido para completar el cálculo y complejidad espacial al número de celdas requeridas de la cinta de E/S. Obsérvese que las complejidades temporales y espaciales son parámetros diferentes y normalmente sus valores van a diferir para un mismo cálculo. Sin embargo, no son totalmente independientes ya que en “n” pasos una MT tiene acceso a un máximo de “n+1” celdas [3].

Considérese una Máquina de Turing M con una entrada "x". Son posibles dos casos en donde las complejidades en tiempo T(x) y espacio E(x) asociadas a la maquina M y su entrada “x” se definen de la siguiente manera:

- M no para con "x":

  T(x) = ? (indefinida)

E(x) = ? (indefinida)

- A partir de una configuración inicial (p, x, 0) la máquina M efectúa una computación de "n" pasos y se detiene con un cierto valor "z" en su cinta. Es decir alcanza una configuración o descripción instantánea final (q, z, i):

  T(x) = n

  E(x) = | z |

Abordando ahora la perspectiva de encontrar funciones que acoten el tiempo y espacio necesarios para resolver un cierto conjunto de problemas se introducen las siguientes definiciones:

Definición 1

Una Máquina de Turing se dice acotada superiormente en tiempo por T(n) si para toda entrada "x" de tamaño "n" se verifica $T(x) \leq T(n)$.

Definición 2

Una Máquina de Turing se dice acotada superiormente en el espacio por E(n) si para toda entrada "x" de tamaño "n" se verifica $E(x) \leq E(n)$.

Definición 3

Un lenguaje "L" se dice de complejidad T(n) y E(n) si existe una Máquina de Turing acotada superiormente en tiempo y espacio que acepta "L". Es decir que para toda cadena $x \in L$ se verifica que $T(x) \leq T(n)$ y $E(x) \leq E(n)$, siendo n = | x |.

Una consideración muy importante es que la ejecución de algoritmos en la máquina de Turing y en computadores reales son modelos de computación polinomicamente relacionados. Es decir, para cualquier programa que compute cierta función en tiempo T(n) y espacio E(n) puede hallarse una Maquina de Turing M que compute la misma función en un tiempo TM (n) donde:

$$T_M(n) = P_t(n).T(n) \quad \text{y}$$

$$T_M(n) = P_e(n).E(n)$$

donde $P_t(n)$ y $P_e(n)$ son expresiones polinomiales en “n”. Esto tiene mucha importancia por las posibilidades que ofrece para la simulación de procesos algorítmicos tanto con procesos secuenciales como paralelos.

6. Medición de la complejidad y computadores reales

Para determinar las complejidades temporales y espaciales sobre una aplicación en un computador real deben considerarse los tiempos requeridos por las instrucciones ejecutadas y los registros utilizados. Para ello deben especificarse el tiempo que demanda individualmente cada instrucción (ciclos de máquina) y el espacio de almacenamiento requerido por cada registro.

Una opción es el denominado “criterio de costo uniforme” que asigna a cada instrucción el consumo de una unidad de tiempo y a cada registro utilizado una unidad de espacio. Por este criterio, T(x) es el número de instrucciones ejecutadas por el computador cuando lee cierta cadena de datos “x” y E(x) la cantidad de registros de memoria utilizados. Este criterio conduce a valores aproximados y solo es recomendable para obtener una primer aproximación de indicadores de complejidad. Sin embargo, es muy útil para comparar diferentes estrategias que puedan estar disponibles para resolver un problema.

Un criterio más realista y específico es el de "costo logarítmico". Con éste criterio se considera la incidencia del largo de los números representados en los operandos en el tiempo de ejecución de cada operación. Así, para el caso de un número "n" la cantidad de dígitos de su representación está dada por $\boldsymbol{L}(n) = \log_2 |n| + 1$ y el costo de ejecutar una sentencia de programa será proporcional a esta cifra.

A continuación, a título de ejemplo, se presentan algunos algoritmos clásicos y se resumen sus correspondientes complejidades teniendo en cuenta el criterio de costo uniforme. Para mayor detalle se recomiendan los textos de Burden y Faires [4] y de Wirth [5].

a) Resolución de Sistema de Ecuaciones Lineales por eliminación de Gauss

$$T(n) = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n = O(n^3)$$

Complejidad Temporal: polinomial (3er grado).

b) Cálculo del determinante de una matriz

$$T(n) = (n!)\sum_{k=1}^{n-1}\left(\frac{1}{k!}\right) + n! = O(n!)$$

Complejidad Temporal: ≈exponencial (factorial)

c) Ordenamiento de un vector por inserción

$$T(n) = \frac{3}{4}n^2 + \frac{7}{4}n = O(n^2)$$

Complejidad Temporal: polinomial (2º grado).

d) Ordenamiento de un vector por selección

$$T(n) = \frac{3}{4}n^2 + 3n = O(n^2)$$

Complejidad Temporal: polinomial (2^{o} grado).

e) Ordenamiento de un vector: método shell sort

$$T(n) = O\left(n\ \log(n)\right)$$

Complejidad Temporal: logarítmica

f) Ordenamiento de un vector: método quick sort

$$T(n) = O\left(n\ \log(n)\right)$$

Complejidad Temporal: logarítmica

Cabe acotar que desde el inicio de la era de la computación el problema del ordenamiento ha estimulado la investigación, y a pesar de su aparente simplicidad, fue siempre reconocido como de gran complejidad. Los ordenamientos eficientes son necesarios para optimizar el uso de otros algoritmos, tales como los de búsqueda y fusión, que requieren listas ordenadas para una ejecución rápida. Aunque muchos puedan considerarlo un problema resuelto, hay aún hoy líneas de investigación abiertas en este campo y se siguen proponiendo nuevos algoritmos y variantes a los ya existentes.

7. Clases de problemas

Todos los problemas matemáticos imaginables pueden ser divididos en dos grupos: los que admiten un algoritmo para su solución y los demostrablemente irresolubles.

Al grupo de problemas de solución algorítmica se los puede a su vez dividir en dos: 1) los problemas de *complejidad temporal polinómica* y 2) los de complejidad temporal exponencial. Los problemas de *complejidad temporal polinómica* tienen su tiempo de ejecución acotada por una función de este tipo y son considerados "eficientes". Por el contrario, la opinión generalizada entre los especialistas de Ciencia de la Computación es que los problemas con solución de complejidad temporal exponencial carecen de interés práctico. Se recomienda aquí volver a las tablas presentadas en el punto 4.1 donde se comparan las complejidades temporales de algorítmos polinómicos (A_1, A_2, A_3 y A_4) con el algorítmo exponencial A_5 para terminar de entender la falta de interés por estos últimos. Como puede comprobarse, en estos problemas denominados intratables solo puede alcanzarse una solución cuando la dimensión de los datos “n” es pequeña.

.En la segunda clase se encuentran los problemas que pueden ser resueltos pero que no admiten un método general, es decir no disponen de un procedimiento. En esta segunda clase se encuentran los problemas de solución iterativa y los de solución probabilística.

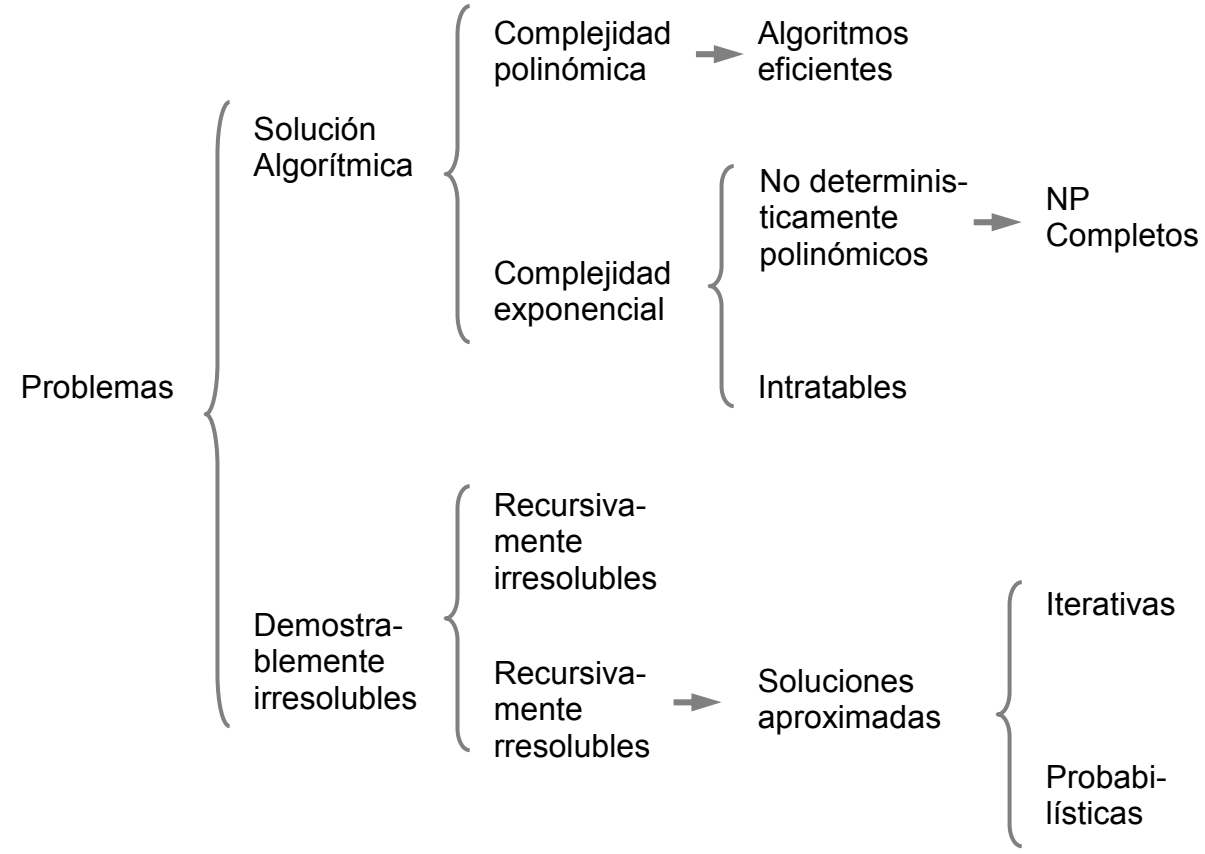

Figura 1: Clases de problemas

Cabe destacar que no siempre es factible establecer a cual de las dos subclases pertenece un cierto problema. Esto es así porque existen una notable clase de problemas que están en una situación intermedia: las mejores soluciones que se

conocen requieren tiempos exponenciales pero sin embargo nadie ha podido demostrar que no tienen una solución de tiempo polinómico. Estos son denominados *No Deterministicamente Polinomicos* o problemas tipo "NP". Un subconjunto de los "NP" son denominados *NP-completos* e incluye algunos problemas clásicos de particular interés en lógica y matemáticas. Los problemas *NP-completos* fueron identificados por Stephen Cook en 1971 y dentro de esta clase se incluyen problemas diversos, algunos de ellos correspondientes a la Teoría de Grafos, como es el caso del problema del viajante de comercio (TSP).

8. Complejidad e Inteligencia Artificial

A partir de los conceptos ya definidos de *Complejidad Temporal y Espacial* se procurará su aplicación para tipificar algunos de los problemas clásicos de la Inteligencia Artificial, como es el caso del entrenamiento de las redes neuronales artificiales y las búsquedas en espacio de estados.

8.1 Complejidad del entrenamiento de las redes neuronales artificiales

Se considerará aquí el entrenamiento de redes multicapa de Perceptrones por el método de backpropagation, quedando definida la arquitectura de la red a partir de los siguientes parámetros:

- m = número de capas
- N = número total de pesos en la red
- n_1 = unidades de la capa de entrada
- n_k = unidades de la capa oculta “k” (1 < k < m)
- n_m = unidades de la capa de salida

Se define además P como el número de patrones de entrenamiento y M el número de ciclos o “epocas” de entrenamiento requerido. Con estos parámetros se calculará su complejidad temporal a partir de la estimación del número de operaciones requeridas y del criterio de costo uniforme. Para conocer la cantidad de operaciones que demanda un proceso de entrenamiento se utiliza la expresión propuesta por Leung [6][7]. Se tiene entonces:

$$T(P,M,m,n_1,..,n_m) = PM\left[\sum_{s=2}^{m-1} n_s(2n_{s+1} + 4n_{s-1} + 3) + n_m(4n_{m-1} + 5)\right] + M\sum_{s=2}^{m} n_s\, n_{s-1}$$

Por razones de simplicidad, se busca expresar el orden de la complejidad en función de la cantidad total de pesos de la red N. Por ejemplo, al evaluar la complejidad temporal para un caso específico en el que P=8, M=10, m=3, N=18, n_1=3, n_2 =4 y n_3 = 2 se obtiene T(8,10,3,4,3,2) = 8.420, lo que implica que $O(N^3) < T(8,10,3,4,3,2) < O(N^4)$.

Los trabajos de Leung [6][7] han determinado la complejidad algorítmica del proceso de entrenamiento por backpropagation en ciertas condiciones y esto permite acotar la *complejidad temporal* del problema con el criterio de costo uniforme. Como conclusión puede decirse que en general

$$O(N^3) < T(N) < O(N^5)$$

Es decir, la complejidad temporal del proceso de entrenamiento de redes multicapa de Perceptrones es de tipo polinómica, de grado entre 3 y 5. Se trata de un problema de solución algorítmica considerado eficiente.

8.2 Complejidad de los algoritmos de búsqueda

La resolución de problemas en el espacio de estados es un campo muy desarrollado por la inteligencia artificial y la determinación de indicadores de complejidad de sus métodos es muy importante para analizar la factibilidad de la solución y la previsión de los recursos necesarios [8]. Se toman aquí como parámetros característicos del problema el factor de ramificación medio "R" y el nivel de profundidad alcanzado en el árbol de búsqueda "p".

Los órdenes de complejidad para los indicadores de *complejidad temporal* y *espacial* de los métodos más difundidos de búsqueda en el espacio de estados se presentan en la Tabla 5.

Como puede apreciarse, la *complejidad temporal* de todos los métodos tiene un límite exponencial. Más aún, la resolución de este tipo de problemas de la IA suele ser caracterizado como NP-completo, pero como mínimo es un problema NP (No-Determinista Polinómico). Como consecuencia, se justifica la necesidad de una buena heurística que restrinja en todo lo posible la zona del espacio de estados en que se desarrollará el proceso de búsqueda de la solución.

Tabla 5: Orden de Complejidad de los principales métodos de búsqueda

Método	Complejidad Temporal	Complejidad Espacial
Primero en anchura	R^p	R^p
Primero en profundidad	R^p	p.R
Descenso iterativo	R^p	p.R
Máxima pendiente	R^p	p.R
Primero el mejor	$\leq R^p$	$\leq R^p$
A*	$\leq R^p$	$\leq R^p$

En efecto, la delimitación de la zona de trabajo en el espacio de estados es el efecto buscado con los algoritmos *Primero el Mejor* y *A**, con los que pueden reducirse notablemente los indicadores de complejidad hasta convertirlos en algoritmos eficientes.

Para comprender el problema es necesario tener presente el tamaño de los espacios de estados (Ne) involucrados, pudiendo citarse como ejemplo el puzzle de 9 posiciones, donde Ne = 9! = 362.880. Nótese que esto es así por la obvia restricción de que una misma pieza no puede estar simultáneamente en distintas posiciones. De lo contrario, es decir si cada dimensión puede tener cualquier valor posible, $Ne = v^d$ donde "v" representa los valores posibles y "d" las dimensiones del vector de estado. En el caso del puzzle Ne sería igual a 9^9 , valor muy superior a 9!.

Volviendo a la necesidad de delimitar la zona de trabajo en el espacio de estados, debe tenerse en cuenta que los métodos *Primero el Mejor* y *A** presentan la seria limitación de poder acabar degenerando en una búsqueda primero en anchura si la función heurística no es suficientemente expresiva de la condición del problema estudiado. Además, si la solución del problema está a mucha profundidad, o el tamaño del espacio de búsqueda es muy grande, es muy probable que aún con los métodos heurísticos el requerimiento de espacio se torne absolutamente prohibitivo.

8.3 Complejidad de los problemas de optimización

Otro tipo de problemas NP-Completos son los de optimización combinatoria, como es el caso del Problema del Viajante de Comercio (TSP), en el que se busca visitar una secuencia de ciudades en el orden más conveniente para hacer mínima la distancia recorrida. A pesar de que los intentos por resolver este problema han sido numerosos y variados, todavía hoy es considerado un problema no resuelto. Las técnicas empleadas requieren muy elevados tiempo de cálculo (Complejidad Temporal) y quedan limitadas a resolver casos pequeños. Por ejemplo, la cantidad de operaciones necesarias para resolver un caso de 100 ciudades es un número de 161 dígitos.

Para superar esta dificultad, algunos de los intentos más recientes han recurrido a heurísticas no deterministas basadas en leyes de la naturaleza, que es un área de estudio que se considera que está en el límite entre la Investigación Operativa y la Inteligencia Artificial.

Cabe aquí mencionar que este problema tiene una solución aproximada que fue propuesta hace ya varios años por Hopfield y Tank [8] a través de la red recurrente que lleva su nombre (Red de Hopfield), demostrando resultados exitosos para problemas de hasta unas 30 ciudades. Para resolver este problema se hace una representación en términos de una función de energía, de manera que el mínimo global de dicha función se corresponda con la solución buscada. El inconveniente que presenta es su sensibilidad a caer atrapado en mínimos locales y brindar soluciones que no son óptimas.

Referencias

[1] Hopcroft J. y Ullman J.; “Introduction to automata theory, languajes and computatios”, Addison - Wesley , 1979.

[2] Aho A., Hopcroft J., Ullman J.; ”The Design and Analysis of computer algorithms”, Addison – Wesley , 1992.

[3] Baum Gabriel; “Complejidad”, Editorial Kapelusz , 1986.

[4] Burden R. y Faires D.; “Análisis Numérico”, 7ª Edición, Ed. Thomson, 2002.

[5] Wirth N.; “Algoritmos y Estructuras de Datos”, Ed. Prentice Hall, 1987.

[6] Leung W., Simpson R.; “Neural metrics-software metrics in artificial neural networks”, Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies, Proceedings, Volume 1,2000.

[7] Leung W.; “An Empirical Study of Software Metrics in Backpropagation Training”, School of Computer Science, Faiculty of Computing, Information Studies and English, University of Central England in Birmingham, U.K, 2002.

[8] Kvitca, A.; “Resolución de problemas con Inteligencia Artificial”, Escuela Brasileño Argentina de Informática – EBAI, 1988.

[9] Hopfield J. y Tank D.; "Neural computation of decisions in optimization problems", Biological Cybernetics, 52, pp.141-152, 1985.