Sistemas Intelixentes

614G010202021

Tema 1:

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

HISTORIA DE LA INTELIGENCIA ARTIFICIAL

Las investigaciones en Neurología en 1940 determinaron que el cerebro era una red eléctrica de neuronas que se activaban en pulsos eléctricos. Wiener realizó un estudio sobre las similitudes entre los sistemas de control y la comunicación entre los seres vivos.

ÉPOCAS DE LA IA

- **Nacimiento de la IA (1952-1956)**

Se considera que la IA nace a partir de la publicación de 3 artículos clave. En uno de ellos, los investigadores **Wiener, Rosenblueth** y **Bigelow** sugieren métodos para que las máquinas sean capaces de cumplir las finalidades y objetivos que se le asignen. Otro artículo de **McCulloch** y **Pitts** manifiesta cómo las máquinas pueden emplear la lógica y la abstracción para demostrar que los datos que reciben de entrada pueden transformarse a un conjunto de datos de salida configurables empleando RNA (redes neuronales artificiales). Finalmente, en su artículo **Craik** propone que las máquinas pueden ser capaces de emplear modelos y analogías a la hora de resolver problemas al igual que lo hacemos los humanos.

Alan Turing diseñó el **Test de Turing** en la década de 1950 para permitirnos detectar si estamos viendo resultados generados por una máquina inteligente o por un humano, planteándose la pregunta "¿puede pensar una máquina?". En verano de 1956 se celebró en Darmouth un proyecto de investigación sobre inteligencia artificial. En él se relacionaron conceptos como la teoría de autómatas, las redes neuronales y el estudio de la inteligencia en los seres vivos y en cómo podrían aplicarse a las máquinas. A partir de aquí surgieron 2 corrientes de pensamiento:

- **Pulcros**

Tienen una perspectiva formal con respecto a la inteligencia. Para ellos la IA debe seguir una lógica establecida e invariante que se puede representar siguiendo un razonamiento simbólico.

- **Desaliñados**

No se preocupan por saber qué causa que un cuerpo sea inteligente: "si es inteligente lo es y punto".

- **Años dorados (1956-1974)**

Las IA desarrolladas mostraban un comportamiento inteligente a un nivel increíble: seguían un razonamiento a la hora de buscar soluciones a los problemas y empleaban un lenguaje muy natural como interfaz para comunicarse. En esta época se ponen las expectativas muy altas en la IA, siendo necesaria la creación del Test de Turing para lograr diferenciar dichos resultados obtenidos.

Sin embargo, las IA comenzaron a sufrir limitaciones cuando se les planteaba resolver problemas de mayor dificultad, llegando a un punto en el que no eran capaces de resolverlo. Esto causó que las instituciones dejasen de invertir capital en las investigaciones, limitando fuertemente el desarrollo de la IA, llegando a provocar la muerte del conexionismo (IA) durante más de 10 años.

- **Primer invierno (1974-1980)**

Hasta ahora los programas que ejecutaban las IA se consideraban "problemas juguete", porque su facilidad era relativamente sencilla. A la hora de resolver problemas más complejos y adaptados del mundo real, sus complejidades computacionales eran altísimas ($O(2^n)$, $O(n!)$, …), conocidos como **explosión combinatoria**. Se dio lugar a la **paradoja de Moravec's**, la cual dice que las computadoras son capaces de derrotar a los mejores jugadores del mundo en ajedrez, pero todavía no son capaces de hacerlas pensar como a un niño de 4 años.

- **Primera primavera: “el Boom” (1980-1987)**

En esta época se da importancia al conocimiento, naciendo los **sistemas expertos**. Gracias a que las IA se lograron comercializar para el uso público, se logró un ROI (retorno de inversión) formado por millones de dólares, lo cual permitió continuar con las investigaciones en IA.

- **Segundo invierno (1987-1993)**

En esta época se produce un cuello de botella a causa del hardware necesario para construir las máquinas. El mantenimiento y la actualización de los sistemas expertos resultaba muy costosa y complicada de mantener (muchos cables y conexiones que controlar). Esto causó que el rendimiento de los sistemas se ralentizase y no generase resultados tan rápido, de modo que provocó otro corte en las financiaciones. Además, surgió un pensamiento de rechazo hacia estos sistemas expertos basándose en que “una mente no puede existir sin un cuerpo”.

- **Éxito entre bambalinas (1993-2001)**

Gracias a la competencia existente entre empresas como *Apple* e *IBM*, la industria del hardware logró avanzar y mejorar los procesos de fabricación. En este instante las IA llegan a la industria para mejorar dichos procesos. Se produjo un cambio de mentalidad: las IA deben centrarse en resolver problemas más pequeños, pero de manera más efectiva (victoria de los pulcros).

- **Segunda primavera (2000-actualidad)**

Gracias al uso intenso de las nuevas tecnologías se produce el fenómeno del ***Big Data***, siendo necesario buscar maneras de gestionar toda esa información, lo cual resulta más ameno con las IA. Todo el conocimiento generado se empleó para entrenar a los sistemas expertos y lograr mejores resultados, destacando ***Alpha Go*** (2015), una IA entrenada para jugar al ajedrez a partir de millones de partidas jugadas entre humanos.

DEFINICIÓN DE INTELIGENCIA

Para **Malraux**, la inteligencia es la posesión de los medios necesarios para dominar cosas y hombres. Para **Minsky**, la inteligencia es la capacidad para resolver problemas que aún no se entienden. Para **Hassenstein**, la inteligencia se caracteriza porque una vez que identificamos un nuevo problema o situación, se es capaz de buscar una solución sin recurrir al ensayo.

Podemos definir la inteligencia en los seres inteligentes mediante una serie de **fenómenos**: estos seres pueden comunicarse, tienen conocimientos propios, memoria y son capaces de procesar nuevas experiencias. Así mismo, son capaces de expresar creatividad, tener intenciones e inferir y razonar conceptos. La **inferencia** es entonces comprender un significado a partir de información relacionada con un concepto, de modo que el **razonamiento** pasa a ser una colección de inferencias conectadas entre sí.

- **Definición de inteligencia artificial**

Se trata de una ciencia que estudia los mecanismos necesarios para lograr que los ordenadores hagan cosas que los humanos hacemos mejor. Así mismo, es una disciplina encargada de diseñar máquinas capaces de realizar tareas que hacen uso de la inteligencia por parte de los humanos. Formalmente, es una rama de la informática que intenta encontrar esquemas para representar el conocimiento y formalizar procesos de razonamiento para resolver problemas complejos dentro de un dominio concreto. La IA puede emplear símbolos y números, así como un conjunto de normas (heurística) y mecanismos para simular el proceso de razonamiento de la mente humana.

SISTEMAS INTELIGENTES

Un **programa de IA** se trata de un programa de ordenador que muestra cierto comportamiento inteligente a causa de seguir una serie de heurísticas. El **conocimiento heurístico** resulta difícil de formalizar, pero establece implícitamente un método para encontrar respuestas relativamente correctas (pero siempre válidas) para un problema concreto. Dicho conocimiento no garantiza encontrar soluciones óptimas, pero sí encontrar soluciones aceptables (si existiesen).

Un **sistema basado en conocimiento** es un conjunto de programas de IA donde los conocimientos del dominio y las estructuras empleadas para manipular dicho conocimiento están físicamente separadas. De este modo se crea una **base de conocimiento**, que agrupa todos los conocimientos sobre el dominio, y un **motor de inferencias**, formado por todos los procesos y estructuras de datos necesarias para manipular dichos conocimientos. Así, una misma base de conocimiento puede ser empleada por varios motores de inferencias, haciendo uso de **interfaces** para realizar un conjunto de tareas determinadas.

Los **sistemas expertos** son un conjunto de sistemas basados en conocimiento que emplean conocimientos sobre un dominio muy concreto para poder resolver problemas del mundo real de gran complejidad pero limitados en tamaño (preferiblemente pequeños). Generalmente los sistemas expertos contienen conocimientos aportados por fuentes humanas expertas, habiendo **3 tipos de conocimientos**:

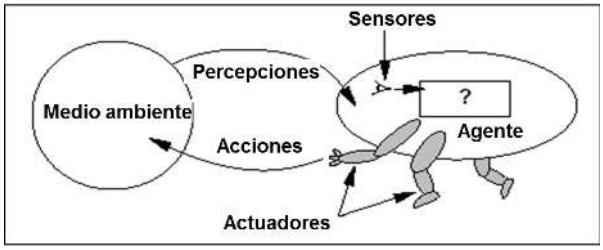
- **Conocimiento público**: puede obtenerse directamente de fuentes públicas, libros, enciclopedias o manuales. Es **comúnmente aceptado** y **reconocido**.
- **Conocimiento semipúblico**: aquel conocimiento que tienen grupos de especialistas y expertos en un ámbito (como médicos, ingenieros, filólogos, etc). Este conocimiento es **explícito**, porque es verídico y puede comprobarse que existe, pero **no es comúnmente aceptado ni reconocido** universalmente (porque no todo el mundo dispone de dichos conocimientos).
- **Conocimiento privado**: es aquel conocimiento heurístico que se emplea únicamente con el objetivo de resolver tareas concretas y que generalmente se adquiere de la experiencia personal de cada individuo, por lo que **no es comúnmente aceptado ni reconocido**.

AGENTES

Según Russel y Norvig, el objetivo de la IA es construir y explicar agentes que reciban percepciones de su entorno y procedan a ejecutar acciones determinadas. Cada agente puede implantarse mediante una función (aplicación matemática) que relaciona percepciones con acciones:

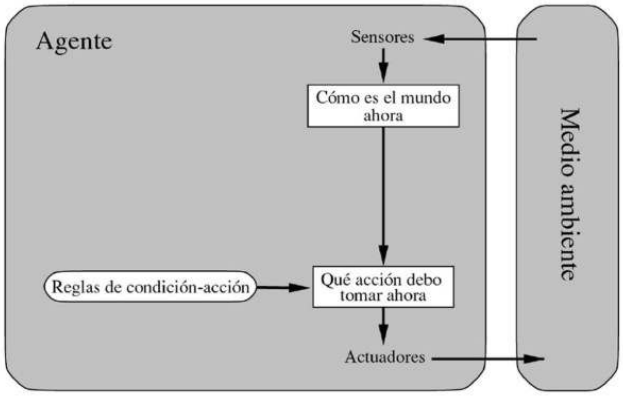
$$f : P^* \rightarrow A$$

$$Agente = Arquitectura + Programa$$

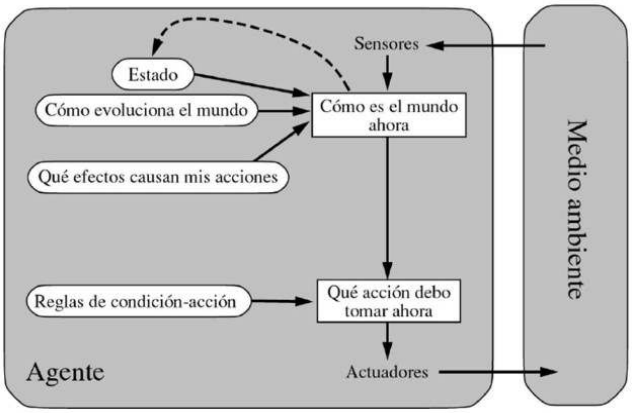


Un agente consta de una arquitectura y un programa. El **programa** es el software que determina el comportamiento del agente e implementa su función. La **arquitectura** engloba a los módulos que componen al agente, como sus componentes de percepción (sensores), componentes de selección de acciones y componentes de acción (actuadores). Los programas para los agentes se diferencian en los métodos que emplean para seleccionar las acciones:

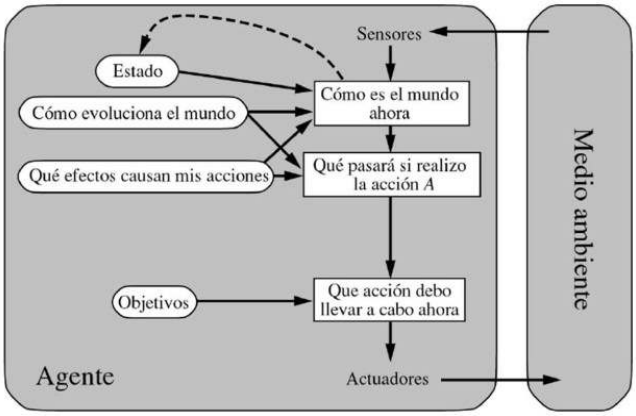
TIPO DE AGENTE	CARACTERÍSTICAS
Reactivo simple	Es el agente más simple. Selecciona la acción en base a sus percepciones actuales y dispone de una inteligencia limitada.
Basado en modelos	Mantiene un estado interno, donde almacena: • cómo evoluciona su entorno • cómo afecta a su entono las acciones que ejecuta.
Basado en objetivos	Conoce los objetivos que debe alcanzar y planifica una secuencia de acciones a realizar para cumplir dichas metas.
Basado en utilidad	Asocia cada estado con un valor numérico que representa la utilidad del estado. Busca alcanzar los estados de mayor utilidad.



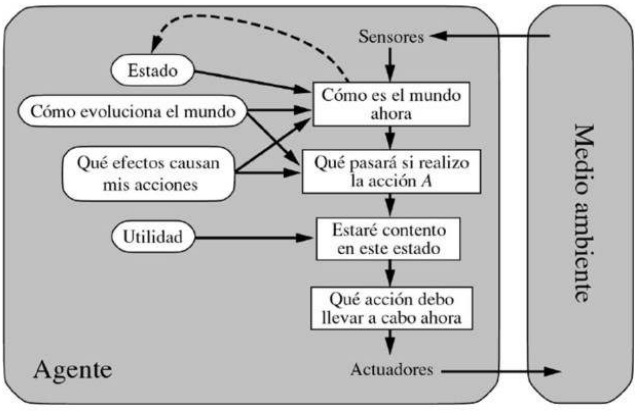
Agente reactivo simple



Agente basado en modelos



Agente basado en objetivos



Agente basado en utilidad

METAS

Un agente inteligente debe maximizar una medida de rendimiento, es decir, satisfacer una meta. El problema que se le plantee resolver puede ser complejo, como elegir un viaje entre dos ciudades basándose en la época del año, distancia, número de viajantes, etc; o puede ser simplificado, como saber que mañana va a viajar a una ciudad concreta y no hará viajes a ningún otro sitio. **Las metas ayudan a organizar el comportamiento limitando los objetivos**. Los pasos para cumplir dichas metas son:

1. **Formulación de la meta**

La meta a cumplir está basada en la situación actual del agente y en su rendimiento. La **meta** es el conjunto de estados del entorno en los que se satisface el objetivo. La tarea del agente es encontrar la secuencia de acciones que permitan alcanzar alguno de los estados meta de dicho conjunto.

2. **Formulación del problema**

A partir de una meta dada, debemos decidir qué procedimiento seguir para decidir qué acciones y estados debemos considerar para alcanzar dicha meta. Antes debemos plantearnos cómo es el entorno que rodea al agente. Puede ser un **entorno desconocido**, donde no es posible elegir un estado porque no conoce que resultado generará, o puede ser un **entorno conocido**, donde se conoce el resultado de elegir un estado.

En un entorno conocido tenemos el problema de que podríamos disponer de varias opciones posibles y no sabríamos cual es la mejor. En este caso, examinaremos todas las secuencias de acciones posibles que nos lleven a un estado conocido y seleccionaremos la mejor de todas esas secuencias. Sabemos que la solución a cualquier problema es una secuencia fija de acciones (que cumplen con estas características), de modo que necesitamos realizar una **búsqueda** para buscar dicha secuencia.

3. **Búsqueda**

Se ejecuta un algoritmo de búsqueda, que toma como **entrada** un problema y devuelve como **salida** una secuencia de acciones (la solución al problema). Se estudiarán más adelante.

4. **Ejecución**

Ya encontrada la solución, se llevan a cabo las acciones generadas por el algoritmo de búsqueda.

PROBLEMAS

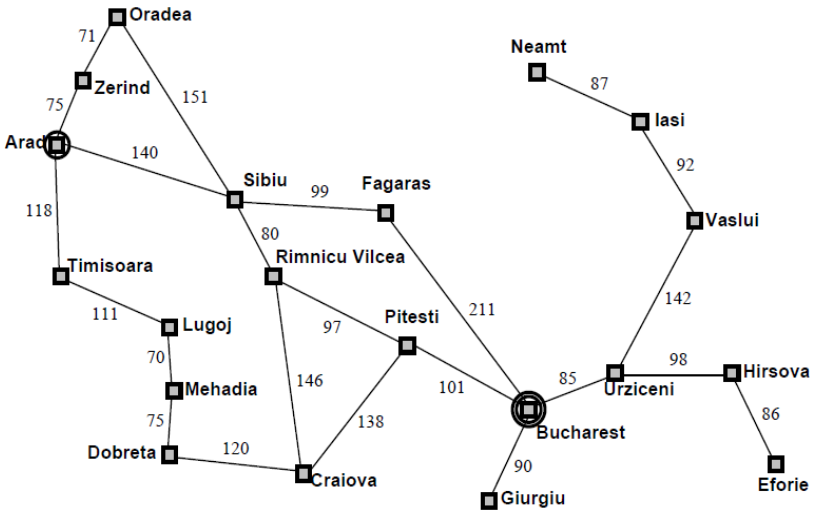
Se pueden clasificar los problemas en tres grupos:

- **Deterministas y completamente observables**: son deterministas porque su solución es un único estado meta, y son completamente observables porque el agente sabe exactamente en que estado estará tras ejecutar una acción. La solución es una secuencia de acciones única.
- **Deterministas y no observables**: sabemos que existe una única solución, pero no cómo llegar a ella (**problema conformante**). El agente no conoce en que estado estará tras ejecutar una acción.
- **No deterministas y/o parcialmente observables**: cada percepción proporciona información nueva sobre el estado actual, de modo que pueden existir varias soluciones. Se trata de un **problema de contingencia** porque el agente intenta contener (acotar) qué secuencias seguir para llegar a las posibles soluciones.
- **Espacio de estados desconocido**: se denominan **problemas de exploración** porque se ha de recorrer el espacio de estados hasta encontrar la meta (**no determinista y no observable**).

■ **Ejemplo de problema: encontrar una ruta en Romanía**

El agente está de vacaciones en la ciudad de Arad (Romanía). Su vuelo de vuelta sale mañana desde Bucarest. El **objetivo** es llegar a Bucarest lo más rápido posible. Para resolver el problema disponemos de unos **estados**, estar en ciudades de Romanía, y una serie de **acciones**, conducir de una ciudad a otra.

La **solución** consiste en encontrar una secuencia de ciudades que nos lleve desde Arad hasta Bucarest.



Disponemos de 5 componentes destacables a la hora de **definir formalmente el problema**:

1) **Estado inicial del agente**. Partimos desde Arad, así que el estado inicial es `EN(Arad)`.

2) **Conjunto de acciones disponibles**. Nuestras acciones consisten en viajar de una ciudad a otra, de modo que realizar una acción en un estado concreto nos devolverá otro conjunto de estados alcanzables. Por ejemplo, el conjunto de acciones a partir del estado inicial `EN(Arad)`: `ACCIONES(EN(Arad)) = {VIAJAR(Zerind), VIAJAR(Timisoara), VIAJAR(Sibiu)}`.

3) **Modelo de transición**. Consiste en describir qué hace cada acción. Se toma un estado origen y una acción y devuelve como resultado el estado que se obtiene tras ejecutar dicha acción. Por ejemplo, viajar de Arad a Zerind nos lleva a estar en la ciudad de Zerind, de modo que: `RESULTADO(EN(Arad), VIAJAR(Zerind)) = EN(Zerind)`.

   Un **espacio de estados** queda definido implícitamente a partir del estado inicial, las acciones y el modelo de transición. Consiste en el conjunto de todos los estados alcanzables desde el estado inicial a través de cualquier secuencia de acciones. El espacio de estados queda representado como un **grafo**, en el cual los nodos son estados y las aristas son las acciones; un camino es una secuencia de estados conectados por una secuencia de acciones.

4) **Prueba de meta** (test objetivo). Determina si un estado pertenece al conjunto de estados meta. Para ello comprueba si el estado a probar pertenece a un conjunto bien definido de estados meta, o si el estado cumple con una propiedad abstracta (como el *jaque mate* en ajedrez). Por ejemplo, el objetivo era llegar a Bucarest, de modo que el estado meta es `EN(Bucarest)`.

5) **Función de coste C**. Asigna un coste numérico positivo a cada camino posible para poder cuantificar el rendimiento de la solución. El coste de un camino se calcula como la suma de los costes de cada acción individual a lo largo del camino (como sumar las distancias, número de acciones ejecutadas...). El coste de la acción a para ir del estado s al estado s' es $C(s, a, s')$. La solución óptima es aquella que tiene el menor coste.

Tema 2:

BÚSQUEDA DEL ESPACIO DE ESTADOS

Fundamentos de Intelixencia Artificial

Vicente Moret, Bertha Guijarro, Eduardo Mosqueira, Mariano Cabrero, Amparo Alonso

ESPACIO DE ESTADOS

El comportamiento inteligente obliga a emplear de manera eficaz y eficiente un conjunto mínimo de conocimientos. Para resolver distintos problemas necesitamos técnicas distintas. Por ejemplo, podríamos plantearnos si hace falta ser inteligente para resolver la ecuación $Ax^2+Bx+C=0$. Una IA debe producir programas que capten generalizaciones de los problemas y que, mediante la inclusión de conocimiento explícito, sean capaces de resolver los problemas que se les plantea; así mismo deben ser fácilmente actualizables (para modificar su comportamiento) y adaptarlas a problemas diferentes.

GENERALIDADES

En IA, la técnica a aplicar para resolver el problema está condicionada por el dominio en el cual será aplicada. Normalmente, para resolver un problema en la vida real no hay planteamientos que sean excesivamente excluyentes, de modo que no existen grandes dificultades a la hora de plantear el problema a resolver. De este modo se establece una diferenciación entre un programa para IA y un programa común para ordenador:

Programas para IA

- Manipulan **dominios simbólicos**.
- Siguen **procesos heurísticos** (basados en la experiencia y los conocimientos).
- **Secuencia de acciones implícitas**, que deben mostrarnos para conocerlas.
- **Información y Control separados**.

Programas convencionales

- Manipulan **dominios numéricos**.
- Siguen **procesos algorítmicos** (empíricos y/ o demostrables matemáticamente).
- **Secuencia de acciones explícitas**, por lo que sabemos qué hacen en cada instante.
- **Información y Control unificados**.

RESOLUCIÓN DE PROBLEMAS

Comencemos analizando el problema de los cubos: "*Disponemos de 2 cubos inicialmente vacíos: un cubo A de 8 L de capacidad y otro cubo B con capacidad para 6 L. Ninguno de los cubos tiene marcas ni divisiones que permitan medir cuánto líquido contienen. También disponemos de un grifo para llenar los cubos. ¿Qué tenemos que hacer para llenar el cubo de 8 L exactamente por la mitad, es decir 4 L?*". En la siguiente tabla podemos observar una posible solución al problema:

Número de orden	Acción	Resultado de la acción
1	Llenar B	A está vacío; B tiene 6 L
2	Vaciar B en A	A tiene 6 L; B está vacío
3	Llenar B	A tiene 6 L; B tiene 6 L
4	Llenar A con B	A tiene 8 L; B tiene 4 L
5	Vaciar A	A está vacío; B tiene 4 L
6	Vaciar B en A	A tiene 4 L; B está vacío

Una vez encontrada una solución al problema debemos plantearnos si la solución es única. Si es así, se da por finalizada la resolución del problema; en caso contrario, deberemos buscar otras posibles soluciones y quedarse con la óptima.

Hemos dicho que hay que seguir una secuencia de acciones entre un conjunto de estados para llegar a la solución, de modo que representaremos la solución mediante una secuencia de acciones y estados. Para ello estableceremos cómo representar el problema:

- A : cubo de 8 L
- [A] : contenido del cubo A
- B : cubo de 6 L
- ([A], [B]) : estado actual del problema (contenido de cada cubo actualmente)
- [B] : contenido del cubo B
- n : número que indica la acción ejecutada.

Número de acción	Precondiciones	Acción realizada
1	A no está lleno	Llenar A
2	B no está lleno	Llenar B
3	A no está vacío	Vaciar A
4	B no está vacío	Vaciar B
5	A no está vacio y B no está lleno y [A] + [B] ≤ 6	Vaciar A en B
6	B no está vacio y A no está lleno y [A] + [B] ≤ 8	Vaciar B en A
7	B no está vacio y A no está lleno y [A] + [B] ≥ 8	Llenar A con B
8	A no está vacio y B no está lleno y [A] + [B] ≥	Llenar B con A

Secuencia solución: **(0, 0) 2 (0, 6) 6 (6, 0) 2 (6, 6) 7 (8, 4) 3 (0, 4) 6 (4, 0)**

Acabamos de definir un **espacio de estados (EE)** porque acabamos de hacer una descripción formal del universo del discurso (el enunciado): definimos un **conjunto de estados iniciales** ((0, 0)), definimos un **conjunto de operadores** que podemos realizar entre los estados (el número de acción), y establecimos un **conjunto de estados meta** ((4, 0)).

La **resolución de un problema en IA** consiste en aplicar un conjunto de técnicas conocidas, cada una de ellas definida como un paso simple (acción) dentro del EE, y definir un proceso de búsqueda dentro del EE para lograr encontrar la solución. Para lograr esto debemos ser capaces de construir un modelo computacional a partir del dominio del problema planteado. Formalmente:

- ***I*** es el conjunto de **estados iniciales** del problema. $I = \{ i_1, i_2, ..., i_n \}$
- ***O*** es el conjunto de **operaciones permitidas** (acciones). $O = \{ o_1, o_2, ..., o_m \}$
- ***M*** es el conjunto de **metas** (soluciones aceptables). $M = \{ m_1, m_2, ..., m_t \}$
- La **búsqueda** es un proceso de exploración en el espacio de estados tal que: $O : (I \rightarrow M)$
- Un **paso simple** es una acción que permite viajar entre dos estados, tal que: $o_x : (i_z \rightarrow i_w) \quad i_z, i_w \in I, \; o_x \in O$. En caso de que $i_w \in M$, entonces i_w verifica la prueba de meta, de modo que es una solución del problema.

El espacio de estados representa únicamente el dominio de un problema concreto, pero no indica cómo obtener la solución. Es por ello que necesitamos procesos de búsqueda para poder explorar el espacio de estados en busca del conjunto de estados meta. Necesitamos establecer unos criterios de selección para avanzar al siguiente estado y saber decidir cuál será el siguiente movimiento (acción) a realizar dentro del espacio de estados. Así podremos realizar una búsqueda sistemática a través de éste.

Es fundamental **tratar de generar siempre nuevos estados sin explorar**, sino quedaríamos atrapados en un bucle (siempre estaríamos explorando los mismos estados, sin encontrar una solución). Para ello es necesario que el sistema disponga de un **autoconocimiento**, es decir, saber en que estado se encuentra en cada momento y qué estados ha explorado previamente, lo cual puede lograrse empleando estructuras de datos auxiliares (pilas, colas, conjuntos, etc).

PROCESOS DE BÚSQUEDA

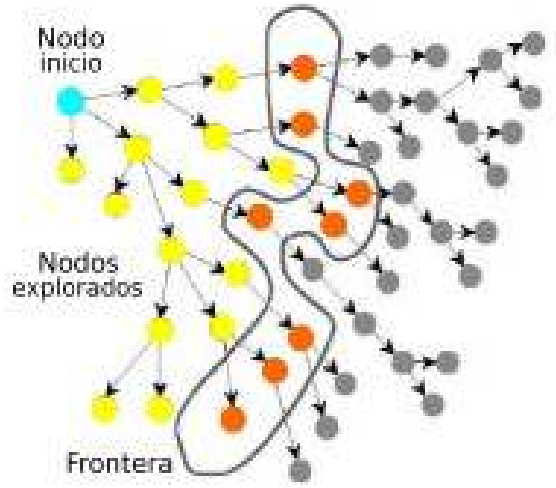
DIRECCIÓN DE BÚSQUEDA

El proceso de búsqueda puede ser **progresivo**, partiendo desde los estados iniciales hacia los estados meta (dirigido por los datos), o **regresivo**, partiendo desde los estados meta hacia los estados iniciales (dirigido por objetivos). A la hora de decidir qué proceso emplear debemos fijarnos en:

1. El **tamaño relativo de los conjuntos I y M** del espacio de estados a recorrer. Resulta más productivo explorar partiendo de un conjunto inicialmente pequeño de información de partida y dirigirse hacia un conjunto mayor de estados.
2. El **factor de ramificación**: número promedio de estados que podemos alcanzar directamente desde un estado dado. Es mejor dirigirse hacia aquellos estados con un menor factor de ramificación. Inicialmente dispondremos de muchos estados que podremos explorar a partir de un estado dado, pero cuanto más se reduzca el factor a medida que avancemos, menor será el número de alternativas a explorar hasta encontrar un estado del conjunto meta.
3. Incluir **estructuras de datos intuitivas** y cuyo uso facilite lo mayor posible explicar el funcionamiento de la búsqueda para el pensamiento humano. Debemos recordar que a la hora de buscar una solución al problema empleando una IA, debemos hacerlo de manera parecida a como lo haría un humano.

TOPOLOGÍA DEL PROCESO

Por lo general, la topología del espacio de estados tendrá forma de grafo, pero resulta más sencillo de explorar cuando se trata de un árbol (grafo simple acíclico). Por lo general, partiremos de un estado inicial (nodo inicio) e iremos generando un **árbol de búsqueda** sobre el propio grafo del espacio de estados. La **raíz** es el estado inicial, los **nodos** son los estados y las **hojas** serán aquellos estados que no tienen sucesor o que aún no lo conocemos (porque aún no hemos explorado lo suficiente). Las **ramas** (aristas) se corresponden con las acciones a realizar para transitar de un nodo a otro nodo. En cada iteración seleccionaremos una hoja y la expandiremos, generando así una nueva frontera. La **frontera** será aquel conjunto de estados aún sin explorar que están conectados con las hojas actuales de nuestro árbol de búsqueda.



Emplear grafos de búsqueda reduce los esfuerzos para explorar el espacio de estados. Sin embargo, a pesar de que el espacio de estados sea finito, el grafo de búsqueda puede ser infinito, dado que podrían repetirse nodos al realizar la búsqueda. Para ello debemos mantener un registro de qué nodos se han explorado ya para no repetirse (algo que con los árboles de búsqueda no sucede porque solo existe un único camino para llegar a cada nodo). Por ello, los árboles de búsqueda son más eficientes a nivel computacional, pero requieren más memoria para poder generarse, al contrario de los grafos de búsqueda, que no son eficientes a nivel computacional pero requieren muy poca memoria.

REPRESENTACIÓN DE ESTADOS

Con respecto al dominio del problema, podemos realizar una **representación estática**, en donde cada nodo representa un objeto, una entidad, un hecho importante en el dominio, etc. Por otro lado, podemos realizar una **representación dinámica**, donde se representan las relaciones entre objetos, entidades o hechos del dominio, es decir, se asocian operadores con acciones. Finalmente, podemos realizar una **representación dinámica del proceso de búsqueda de soluciones**, en donde se representan las secuencias de estados que se van generando durante los procesos de búsqueda (las estrategias).

CRITERIOS DE SELECCIÓN DE OPERADORES

Ya sabemos que la aplicación de un operador a un estado nos genera otro nuevo estado, pero ahora queremos ser capaces de extraer, de entre el conjunto de todos los operadores posibles, aquel subconjunto de operadores que nos resulten útiles. Se denomina **emparejamiento** al proceso de selección sistemática de operadores relevantes para nuestro uso, y consiste en reconocer qué operadores se pueden aplicar a los estados de nuestro espacio de estados. Es una de las tareas más costosas en los programas de IA y se puede dividir en dos tipos de emparejamiento:

- **Emparejamiento literal**

Se trata de buscar los operadores dentro del conjunto de acciones *O* del espacio de estados. Para seleccionar el operador debemos fijarnos en si es dirigido por los datos (atiende a las precondiciones) o dirigido por los objetivos (atiende a los consecuentes). Resulta ineficiente si el conjunto *O* es muy grande, y no siempre resulta evidente ver qué operadores podemos aplicar a un estado dado de nuestro espacio de estados. Este tipo de emparejamiento resulta de utilidad en dominios pequeños.

Algunos casos particulares se obtienen cuando: el operador empareja completamente con el estado dado, las precondiciones del operador son un subconjunto de la descripción del estado actual o coinciden parcialmente con su descripción, o ni siquiera coinciden.

- **Emparejamiento con variables**

Es útil cuando el problema a resolver necesita una búsqueda intensa donde hay variables involucradas. Pongamos un ejemplo para explicarlo:

Los **hechos del dominio** son los siguientes:

- Juan es hijo de María → `HIJO(María, Juan)`
- Pedro es hijo de Juan → `HIJO(Juan, Pedro)`
- Tomás es hijo de Pedro → `HIJO(Pedro, Tomás)`
- Rosa es hija de Pedro → `HIJA(Pedro, Rosa)`
- Ana es hija de Juan → `HIJA(Juan, Ana)`
- Rosa es hija de Ana → `HIJA(Ana, Rosa)`

Los **operadores del dominio** son los siguientes:

- Op1: `HIJO(x, y) AND HIJO(y, z) → NIETO(x, z)`

- Op2: `HIJA(x, y) AND HIJO(y, z) → NIETO(x, z)`
- Op3: `HIJO(x, y) AND HIJA(y, z) → NIETA(x, z)`

El **problema** surge cuando, dados los hechos anteriores (estados iniciales), queremos encontrar un estado meta que incluya los mismos hechos y un hecho nuevo que indique quién es el nieto de Juan. Para la **solución**, resulta de interés emplear Op1 u Op2 porque son los operadores que indican la existencia de algún nieto (por ejemplo, cuando hacemos `x=Juan`, lograríamos obtener que Juan es el nieto de `z`). Ahora debemos encontrar un `y` que verifique que `HIJO(Juan, y) AND HIJO(y, z)`, para algún valor de `z`. Tenemos 2 opciones: la primera sería comprobar a todos los hijos de Juan y verificar que alguno de ellos tenga un hijo; la segunda sería comprobar que, de todos los individuos que tienen algún hijo, hay algún individuo que es hijo de Juan.

Se produce un **conflicto** cuando se identifica más de un operador aplicable a nuestro estado actual, de modo que debemos ser capaces de elegir qué operador emplear de forma que nos ofrezca mayores garantías de éxito (para acercarnos al estado meta). Un **sistema de resolución de conflictos** es básico para resolver estos problemas, y debe basarse en 4 principios:

1. Si podemos evitarlo, no aplicaremos operadores que ya hayan sido aplicados.
2. Aplicar primero aquellos operadores que emparejen con hechos incorporados recientemente.
3. Aplicar primero aquellos operadores que tengan las precondiciones más restrictivas.
4. En caso de no poder seleccionar ningún operador siguiendo las 3 reglas anteriores, seleccionar un operador al azar.

OPTIMIZACIÓN MEDIANTE FUNCIONES HEURÍSTICAS

Una **función heurística** está basada en un número que nos permite estimar el beneficio de una transición concreta dentro del espacio de estados. Se emplean para optimizar los procesos de búsqueda porque se buscan aquellos caminos que tengan el "mejor valor" en caso de que existan varias posibilidades, escogiendo el camino óptimo.

EXPLORACIÓN DEL ESPACIO DE ESTADOS

A la hora de determinar qué estrategia usar para buscar en el espacio de estados debemos elegir aquella que nos ofrezca una mayor eficiencia al recorrer el grafo. Existen técnicas óptimas para grafos dentro de un dominio concreto, pero estudiaremos técnicas que nos permitirán recorrer cualquier tipo de grafo del espacio de estados de manera genérica.

EVALUACIÓN DE ESTRATEGIAS DE BÚSQUEDA

Una estrategia se define por el orden en el que expandirá los nodos del espacio de estados. Para ello debemos atender a: su **completitud**, comprobar si, siempre que exista una solución, la encuentra; su **complejidad temporal**, calculando cuánto tarda en encontrar una solución; **complejidad espacial**, estimando cuanta memoria necesita; **optimización**, viendo si encuentra siempre la solución más óptima.

En el caso de la complejidad temporal y espacial, podemos realizar mediciones en función de:

- **Factor de ramificación** (b): número medio de sucesores (hijos) por cada nodo.
- **Profundidad de la solución menos costosa** (d): también incluye el nodo meta óptimo.
- **Profundidad máxima del espacio de estados** (m).

En el caso de la **complejidad temporal**, podemos calcularla midiendo el tiempo que se tarda en generar el máximo número de nodos posibles. La **complejidad espacial** podemos medirla a partir del espacio que ocupa el máximo número de nodos generables. En todo caso, podemos medir la **efectividad** general de una estrategia de búsqueda basándonos en su coste total, que incluye el coste de la búsqueda dentro del espacio de estados y el coste de recorrer dicho camino hacia el nodo meta.

Coste total = coste búsqueda + coste camino

Desde ahora debemos diferenciar que **espacio de estados y árbol de búsqueda no son lo mismo**. Un estado se corresponde con una configuración del mundo real, mientras que un nodo es un elemento de una estructura de datos (grafo u árbol) que representa al árbol de búsqueda. De este modo, **dos nodos diferentes pueden contener el mismo estado**. Los componentes de un nodo son su estado, su nodo padre, la acción que puede realizar el nodo y el coste del camino (al viajar de un nodo a otro).

- **Implementación de las estrategias de búsqueda**

El espacio de estados define un grafo de búsqueda implícito (con un nodo inicial y una función para calcular a su sucesor), y no un árbol. Los algoritmos de exploración solo recordarán un único padre para cada nodo. Definimos **expansión** de un nodo a la aplicación de operadores relevantes sobre dicho nodo para generar nuevos sucesores. Para ello definimos dos conjuntos de nodos:

- **Nodos abiertos (frontera)**: son los nodos a la espera de ser expandidos. Permiten separar al grafo del espacio de estados en una región explorada y una región sin explorar, de modo que cualquier camino desde el estado inicial hacia un estado sin explorar tiene que pasar a través de un estado perteneciente a la frontera. Suelen incluirse en estructuras de datos del tipo cola
- **Nodos cerrados (explorados)**: son los nodos ya expandidos. Generalmente suelen incluirse en estructuras de datos del tipo tablas de dispersión o en colas. Necesitamos almacenarlos para no expandir el mismo nodo varias veces.

La **estrategia de búsqueda** a seguir consiste en seleccionar el siguiente nodo a expandir a partir de la lista de nodos abiertos. Debemos resaltar que el tipo de cola que empleemos en los nodos cerrados (FIFO, LIFO, …) afectará al orden en que se realiza la búsqueda. La estructura de datos más apropiada para la frontera es una cola, cuyas operaciones básicas son: `EMPTY`, para ver si hay más elementos en la cola; `POP`, para extraer el primer elemento de la cola, e `INSERT`, que inserta un elemento en la cola. Las estrategias de búsqueda pueden ser de 2 tipos:

- **No informada (*a ciegas*)**: no conocen información sobre los estados que no son proporcionados inicialmente por el problema.
- **Informada (*heurística*)**: tienen información suficiente sobre el problema como para llegar a un nodo meta de manera eficiente.

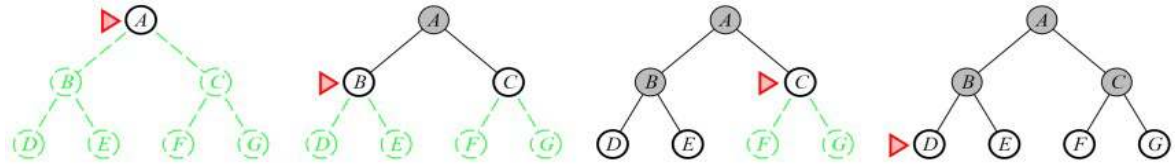
ESTRATEGIAS DE BÚSQUEDA NO INFORMADA (A CIEGAS)

Una búsqueda a ciegas no establece ninguna preferencia sobre el siguiente nodo a expandir. Estas estrategias de búsqueda se diferencian por el orden en el que se expanden los nodos, lo cual afecta fuertemente a la calidad del proceso a la hora de evaluar la estrategia (4 puntos de evaluación anteriores).

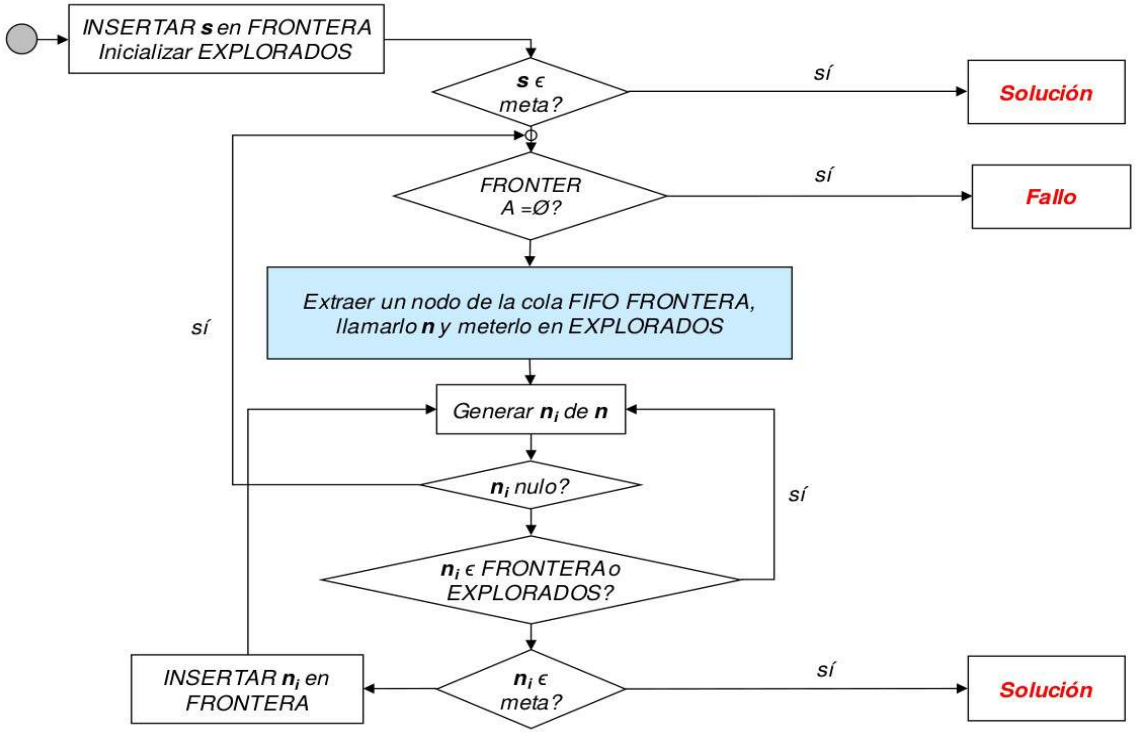
- **Búsqueda en amplitud (anchura, *breadth-first*)**

Expandir primero el nodo raíz, luego los sucesores del nodo raíz, a continuación todos los sucesores de los sucesores y así recursivamente. Esta estrategia se caracteriza porque **se expanden todos los nodos del mismo nivel antes de expandir nodos de niveles inferiores**. Existen 2 variantes:

- Seleccionar para expandir primero aquel nodo sin expandir cuya profundidad sea la mínima. Dado que estamos en una búsqueda a ciegas, no podemos conocer la profundidad de un nodo.
- Implementar la frontera como una cola FIFO donde los nuevos nodos se insertarán al final.



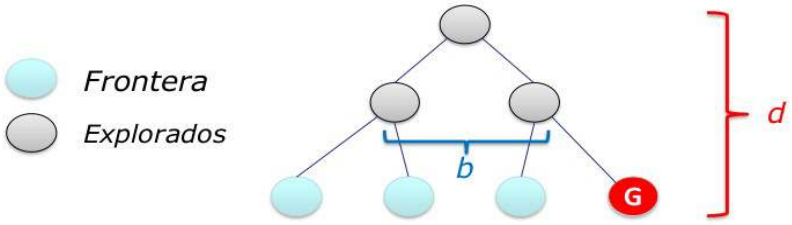
Emplearemos la segunda variante. En ella, la **cola** representa el camino conteniendo sólo la raíz. Los pasos a seguir se resumen en que, mientras la cola no está vacía y no se haya alcanzado la meta, deberemos: eliminar el primer nodo de la cola, aplicar los operadores relevantes al nodo extraído para poder generar sus sucesores y finalmente añadir dichos nodos sucesores al final de la cola. Si haciendo esto hemos alcanzado un nodo meta, la búsqueda ha sido un éxito; en caso contrario, no existe una meta.



◆ **Evaluación de la búsqueda en amplitud**

La búsqueda en amplitud es **completa** porque si el nodo meta está a una profundidad d , entonces expandirá antes los nodos más próximos (suponiendo que el factor de ramificación b es infinito). Por otro lado, no es **óptima** porque el nodo meta más cercano no tiene por qué ser el óptimo: en caso de que fuese el nodo más a la derecha en el nivel actual o en el siguiente, nos seguiría obligando a expandir todos los hijos de los nodos del mismo nivel o del siguiente. El peor caso sería su fuese el nodo hoja más profundo del grafo situado a la derecha (obligaría a explorar todo el grafo). Sin embargo, también se considera una búsqueda **óptima** porque todas las acciones tienen el mismo coste.

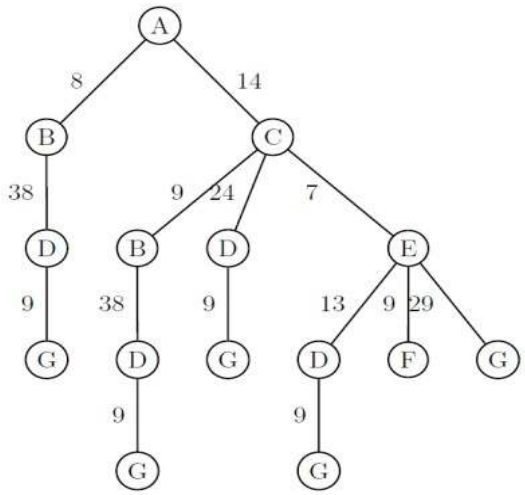
Atendiendo a su **complejidad temporal**, en caso de que un nodo meta se encontrase a una profundidad d , habría que generar todos los nodos correspondientes a ese nivel: en el nivel 1 explorará b nodos, en el nivel 2 explorará b^2 nodos… Por ello su complejidad es: $\boxed{b+b^2+b^3+...+b^d = O(b^d)}$.
Estudiando su **complejidad espacial**, cada nodo generado permanece en memoria, de modo que al llegar al nivel del nodo meta se almacena la mayor cantidad de nodos. Cuando el nodo meta llegue a formar parte de la frontera, habremos explorado b^{d-1} nodos y almacenado b^d nodos en la frontera, por lo que su complejidad espacial también es: $\boxed{b+b^2+b^3+...+b^{d-1}+b^d = O(b^d)}$.



Veamos un ejemplo de esta búsqueda con datos reales. Supongamos un árbol de búsqueda con un factor de ramificación $b=10$, en donde cada nodo ocupa 1000 bytes y nuestra máquina es capaz de procesar 10^6 nodos cada segundo. El aumento de la complejidad se refleja en la siguiente tabla:

Profundidad (d)	**2**	**4**	**6**	**8**	**10**	**12**
Nodos	$110 \approx 10^2$	$11110 \approx 10^4$	10^6	10^8	10^{10}	10^{12}
Tiempo	0,11 ms	11 ms	1,1 s	2 min	3 h	13 d
Espacio	107 KB	10,6 MB	1 GB	103 GB	10 TB	1 PB

- **Ejemplo de búsqueda en amplitud**



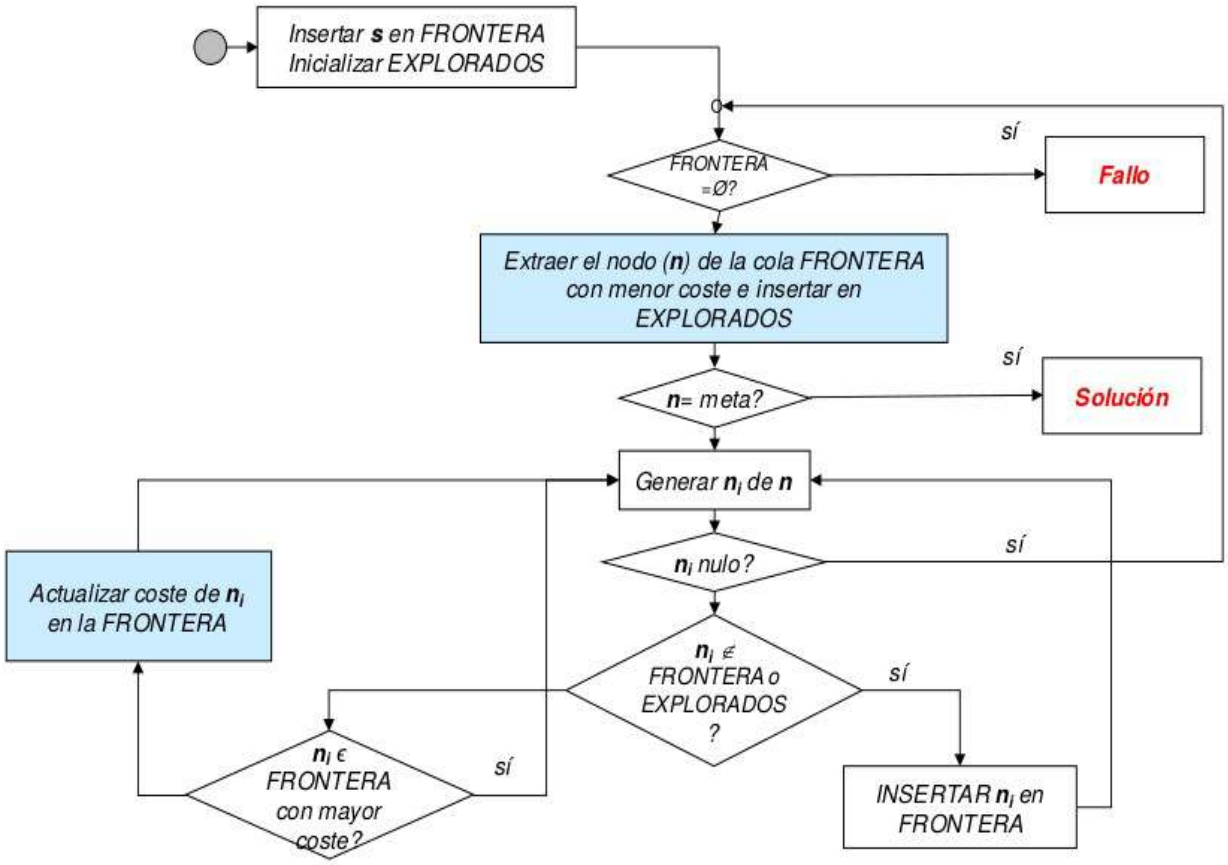
Dado el grafo de la izquierda, donde el nodo G es la meta. Al generar y expandir los nodos frontera y explorados obtenemos:

Paso	**Frontera**	**Explorados**
1	A	-
2	B, C	A
3	C, D	A, B
4	D, E	A, B, C
5	E, **G**	A, B, C, D

- **Búsqueda de coste uniforme**

Emplear búsqueda por anchura al inicio resulta óptimo si los costes de transición entre estados son constantes. Aún así, podemos optimizar el algoritmo ampliándolo para que en cada paso de la búsqueda emplee una **función de coste** $g(n)$ para expandir los nodos *n* con el menor coste de camino. En caso de que todos los costes sean iguales, la búsqueda de coste uniforme es igual a la búsqueda en amplitud. Cuando se encuentre un nuevo camino hacia un nodo aún sin explorar que tenga coste menor que el camino actual hacia dicho nodo, se eliminará el camino de mayor coste y se actualizará con el nuevo.

Su implementación consiste en hacer que la **frontera** sea una cola de prioridad ordenada por el coste g de cada camino (de menor coste a mayor coste). El **test de meta** se aplicará al nodo cuando se selecciona para ser expandido, no tras haberse generado, y se comprobará si ya existe un camino mejor para ir hacia un nodo en la frontera.

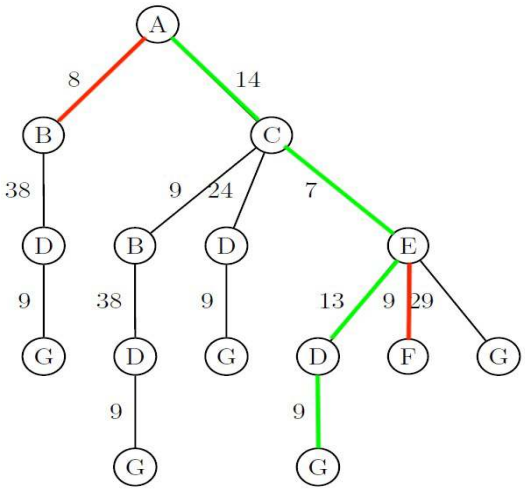


- **Evaluación de la búsqueda de coste uniforme**

La búsqueda es **completa** si el coste de cada paso es mayor que el coste de cada acción E , es decir, si $g(n) \geq E$, $E > 0$. Por otro lado, la búsqueda es **óptima** porque todos los nodos se expanden en orden creciente según su coste, por lo que el nodo meta será el más óptimo al expandir.

Atendiendo a su **complejidad espacial y temporal**, el número de nodos cuyo coste del camino es menor o igual que el coste de la solución óptima (C^*) tiene una complejidad $\boldsymbol{O(b^{1+[C^*/E]})}$. Si coste de todos los pasos es igual, la complejidad espacial y temporal pasa a ser $\boldsymbol{O(b^{1+[C^*/E]}) = O(b^{d+1})}$.

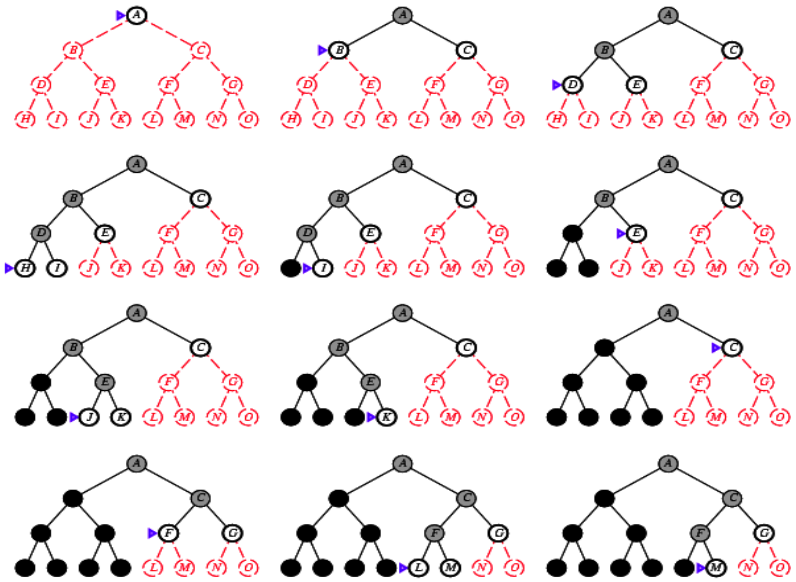
- **Ejemplo de búsqueda de coste uniforme**

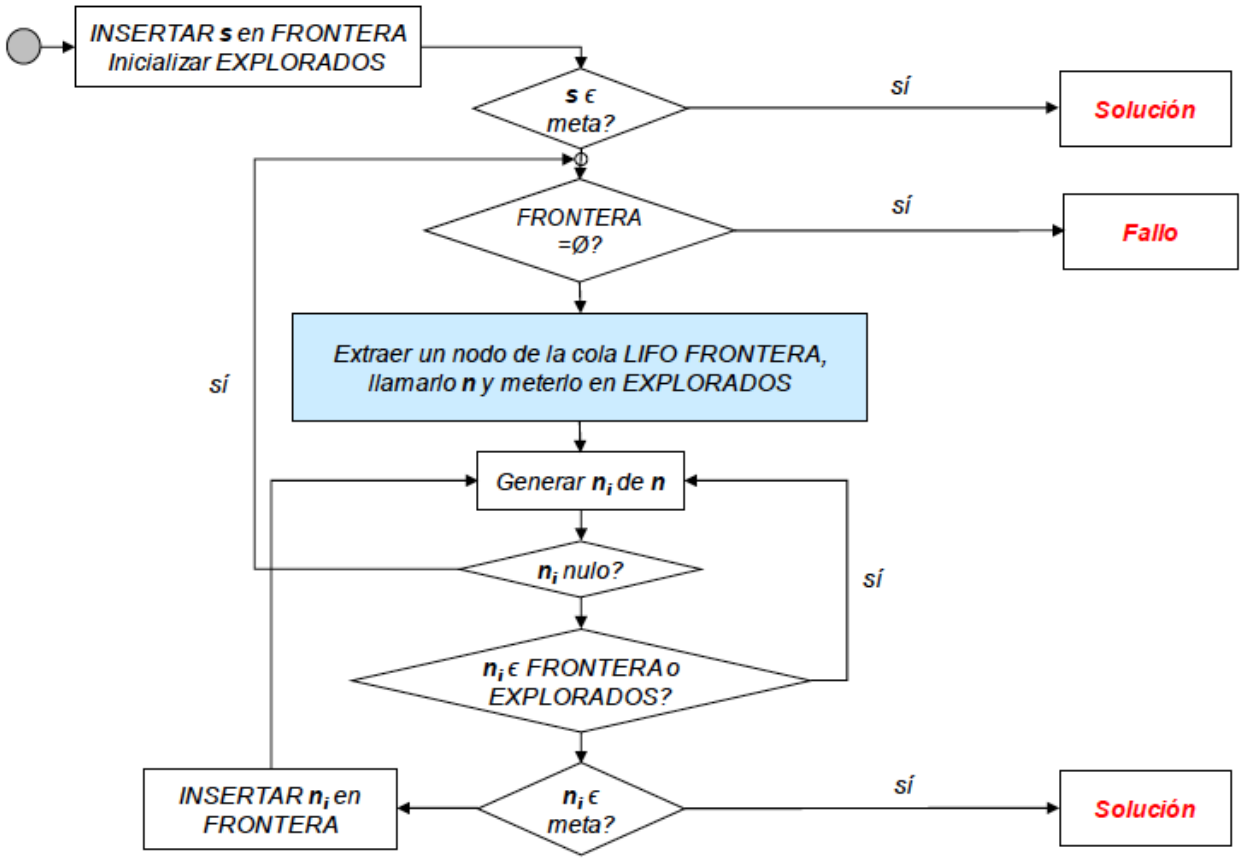


Paso	Frontera	Explorados
1	A(0)	-
2	B(8), C(14)	A(0)
3	C(14), D(46)	A(0), B(8)
4	E(21), ~~B(23)~~, D(38), ~~D(46)~~	A(0), B(8), C(14)
5	F(30), D(34), ~~D(38)~~, ~~D(46)~~, **G(50)**	A(0), B(8), C(14), E(21)
6	D(34), ~~D(46)~~, **G(50)**	A(0), B(8), C(14), E(21), F(30)
7	~~G(50)~~, **G(43)**	A(0), B(8), C(14), E(21), F(30), D(34)
8		A(0), B(8), C(14), E(21), F(30), D(34), G(43)

- **Búsqueda en profundidad (*depth-first*)**

Esta búsqueda consiste en expandir los nodos más profundos de la frontera actual del árbol. Una vez generados los nodos de la frontera, se retrocede hasta el siguiente nodo más profundo que tenga sucesores sin explorar. Para ello debemos expandir el nodo generado más recientemente, lo cual podemos hacer empleando una **pila o cola LIFO (nuevos nodos se insertan por el principio**).





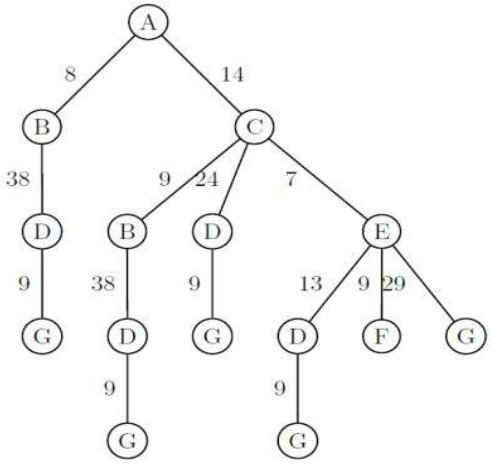
- **Evaluación de la búsqueda en profundidad**

La búsqueda es **completa** cuando se aplica a grafos basados en espacios de estados finitos porque expandirá todos los nodos existentes. Sin embargo, **no es completa** si la usamos en un árbol o un espacio de estados infinito donde un camino no conduce hacia la meta. Por otro lado, **no es óptima** porque podría existir una solución a menor profundidad en una rama sin expandir que la profundidad de la solución en la rama que está expandiendo actualmente.

Atendiendo a su **complejidad temporal**, es similar al tamaño del espacio de estados, o incluso infinito si se busca en un espacio de estados basado en un grafo. En el peor caso la solución estaría en el hijo derecho más profundo de la rama derecha, obligándole a explorar todas las ramas. Comenzando desde la raíz hasta las ramas: $\boxed{b^m + b^{m-1} + \ldots + 1 = O(b^m)}$.

Por otro lado, estudiando **complejidad espacial** vemos que el mayor número de nodos almacenados en memoria se alcanza en el nodo inferior de más a la izquierda. Dado que sólo se va almacenando un camino (una raíz y una hoja) para la búsqueda en árbol, tendrá que almacenar $\boxed{1+b+b+b+...+b=O(m\cdot b)}$

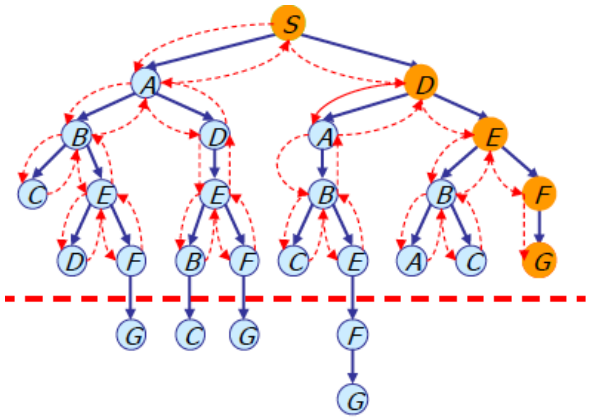
- **Ejemplo de búsqueda en profundidad**



Paso	Frontera	Explorados
1	A	-
2	B, C	A
3	D, C	A, B
4	G, C	A, B, D

- **Búsqueda de profundidad limitada**

Una alternativa para evitar recorrer árboles de un espacio de estados infinito consiste en establecer un **límite de profundidad máxima** l , de modo que no se buscarán soluciones a mayor profundidad que dicho límite. Esto permite solucionar caminos infinitos porque se fija un límite a la profundidad máxima. Sea d la **profundidad de la meta óptima**, pueden darse dos casos:

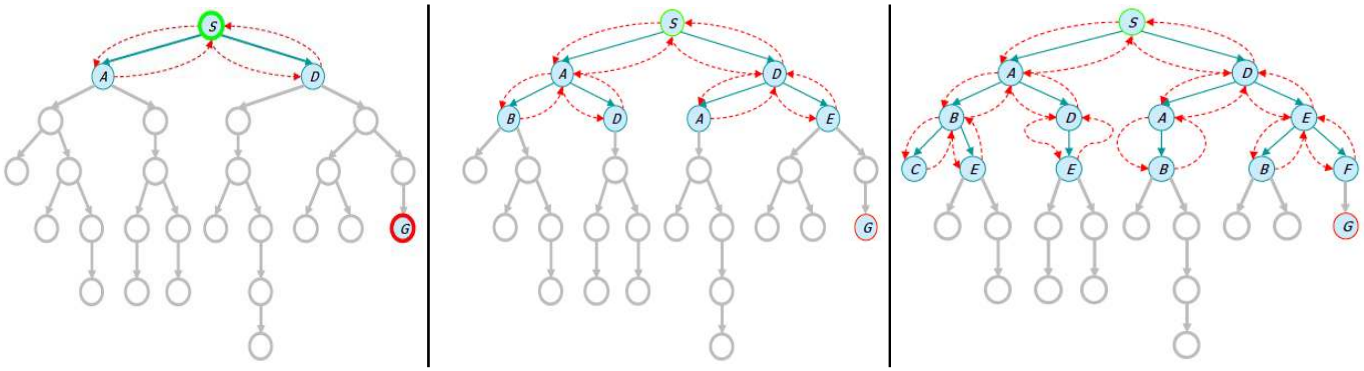


- $l < d$: meta es más profunda que el límite, por lo que nuestra búsqueda es incompleta.
- $l > d$: se recorren más niveles (y más nodos) de lo necesario, haciendo la búsqueda no óptima.

Con este límite la **complejidad temporal** pasa a $\boxed{O(b^l)}$ y la **complejidad espacial** pasa a $\boxed{O(l\cdot b)}$.

- **Búsqueda de profundidad iterativa**

Es una variante de la búsqueda de profundidad limitada, pero en donde el límite l se va incrementando gradualmente ($l=\{0,1,2,3,...\}$) hasta que se encuentra una solución.



■ **Comparación de las estrategias de búsqueda a ciegas (en árboles)**

	Amplitud	**Coste uniforme**	**Profundidad**	**Profundidad limitada**	**Profundidad iterativa**
¿Completa?	Sí [a]	Sí [a), b)]	No	Sí (si $l \geq d$)	Sí [a]
¿Óptima?	Sí (coste cte.)	Sí		No	Sí [c]
¿Complejidad espacial?	$O(b^d)$	$O(b^{1+\lfloor \frac{C^*}{E} \rfloor})$	$O(m \cdot b)$	$O(l \cdot b)$	$O(d \cdot b)$
¿Complejidad temporal?			$O(b^m)$	$O(b^l)$	$O(b^d)$

a) Es completa si b es finito.

b) Es completa si el coste de cada paso g es mayor o igual que el coste de cada acción E .

c) Es óptima si los costes de transición son iguales para todas las acciones (aristas).

ESTRATEGIAS DE BÚSQUEDA INFORMADA (HEURÍSTICA)

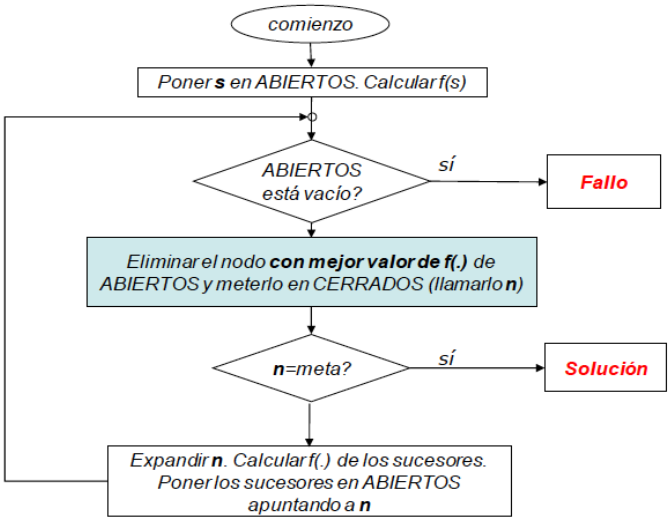
Estas estrategias están basadas en la búsqueda en anchura y examinan primero aquellos nodos que, siguiendo una heurística, están situados en el mejor camino que lleva hacia la solución. La gran diferencia es que ahora sí disponemos de información sobre todos los estados, pudiendo saber si un estado no-meta es más prometedor que otro estado para tomarlo como camino hacia la meta.

■ **Funciones heurísticas**

La búsqueda informada mediante heurísticas está basada en los algoritmos de búsqueda en árboles. El nodo *n* a expandir es seleccionado siguiendo una **función de evaluación** $f(n)$: primero el menor valor (que en nuestro caso será el mejor). Estas funciones f suele incluir una **función heurística** $h(n)$: calcula el coste estimado del camino menos costoso desde el estado actual (nodo *n*) hasta el estado meta.

Una función heurística permite estimar el beneficio de una transición en el espacio de estados y su valor está descrito en cada estado. Establecemos una restricción importante: si *n* es un nodo meta, entonces su heurística vale cero: $h(n) = 0 \quad \forall n \ meta$. El uso de una heurística permite optimizar el proceso de búsqueda, guiándonos hacia aquellos estados que mejor provecho nos pueden ofrecer y elegir un camino cuando existen varias alternativas posibles. Emplearemos una **cola de prioridad** que guarde en orden ascendiente la lista de nodos en la frontera según su valor de f . El proceso a seguir es:

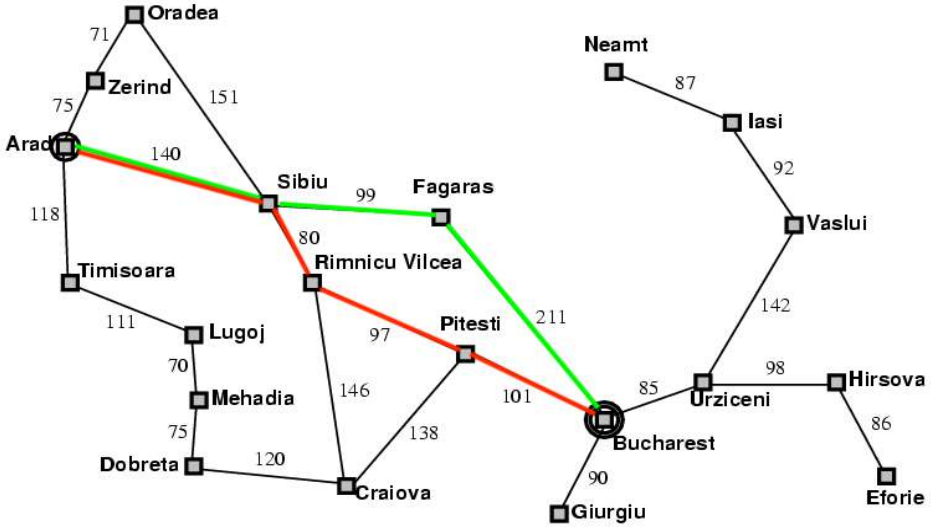
1) Seleccionar un nodo a expandir en función del valor de la heurística. Por ejemplo, podemos hacer que $f(n)$ mida la distancia al objetivo.
2) A continuación decidiremos cuál es el mejor nodo a expandir. Como hemos dicho, siempre tomaremos aquellos caminos que nos lleven a un nodo con un valor heurístico más pequeño.
3) La búsqueda termina cuando el nodo a expandir sea un nodo meta.



- **Búsqueda avara**

En esta búsqueda, la función $f(n)$ sólo considera el **coste mínimo estimado** para llegar a la solución a partir del nodo actual *n*, es decir, solo considera la función heurística $h(n)$, por lo que $\boxed{f(n)=h(n)}$.

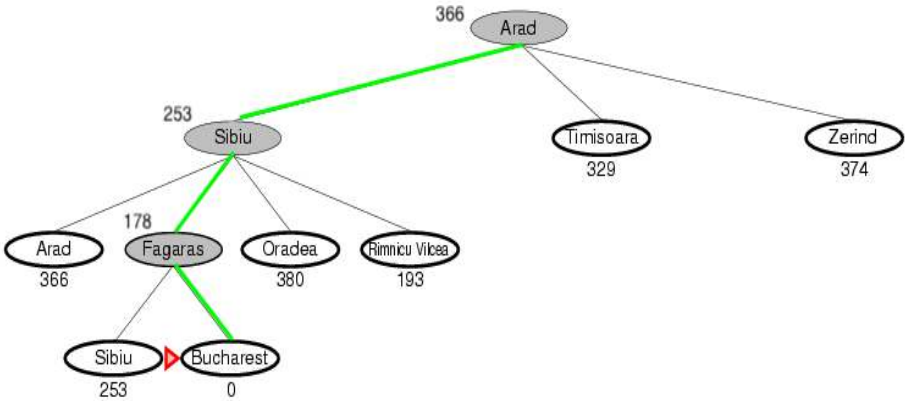
Para poner un ejemplo retomaremos el **problema de las rutas en Rumanía** (explicado en el tema 1), en el cual buscábamos el mejor camino para ir desde Arad hasta Bucarest. En este ejemplo podemos tomar como heurística a la distancia en línea recta desde la ciudad actual (cada nodo) hasta el estado meta (ciudad de Bucarest). Definimos así la función heurística como $h(n)=distLineaRecta(n,Bucharest)$. Obviamente, desde Bucarest hasta Bucarest hay distancia 0, por lo que se cumple que $h(n)=0$.



Distancia en línea recta a Bucarest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Partiendo desde el estado inicial (*Arad*), tenemos como opciones (nodos frontera) a $\{Sibiu(253), Timisoara(329), Zerind(374)\}$. Dado que *Sibiu* tiene la menor heurística, tomamos ese nodo para expandir, obteniendo ahora la frontera $\{Fagaras(178), Oradea(380), Rimnicu\,Vilcea(193)\}$. La ciudad de *Fagaras* tiene la menor heurística, así que tomamos ese nodo para expandir y obtenemos la frontera $\{Bucharest(0)\}$.

Encontramos así el camino $\boldsymbol{Arad \rightarrow Sibiu \rightarrow Faragas \rightarrow Bucarest}$ cuyo coste real es $140+99+211=450$, mientras que la heurística nos dice que la distancia de Arad a Bucarest en línea recta es 366.

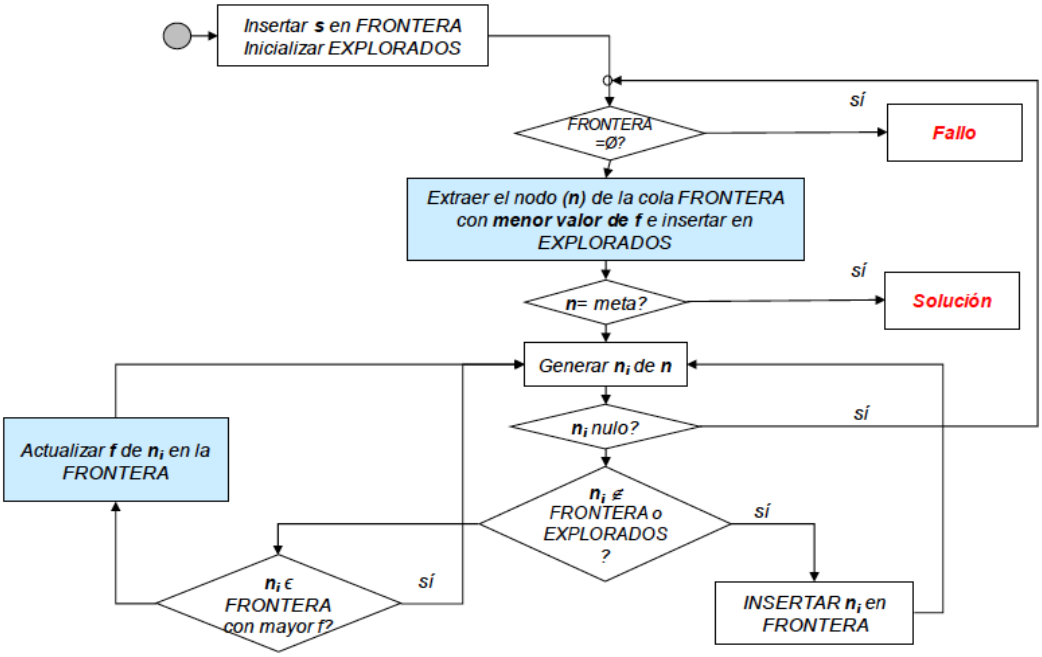


Sin embargo, existe mejor ruta: el camino $\boldsymbol{Arad \rightarrow Sibiu \rightarrow Rimniu\,Vilcea \rightarrow Pitesti \rightarrow Bucarest}$, cuyo coste real es de $140+80+97+101=418$ (marcada en rojo en el grafo de las ciudades). Concluimos que la búsqueda avara **no es óptima ni completa**, porque puede llegar a perderse en bucles y recorrer rutas infinitas. Así mismo, su **complejidad temporal** es $\boldsymbol{O(b^m)}$, donde *m* es la profundidad máxima del árbol de búsqueda. Su **complejidad espacial** también es $\boldsymbol{O(b^m)}$ porque se han de mantener todos los nodos explorados en memoria.

- **Búsqueda A***

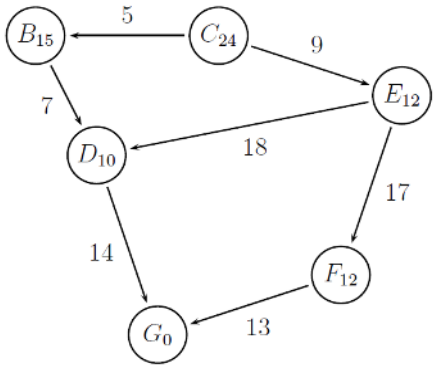
En esta búsqueda, la función $f(n)$ también considera el coste mínimo total del camino desde el nodo actual *n* hasta un nodo solución. Por lo tanto, al valor de la heurística se le debe añadir otra función $g(n)$ que mida el **coste consumido para llegar a *n***, por lo que $\boldsymbol{f(n)=h(n)+g(n)}$. De este modo:

- $g(n)$: calcula el **coste real** del mejor camino para llegar al nodo *n*.
- $h(n)$: calcula el **coste estimado** del camino menos costoso desde *n* hasta la meta.
- $f(n)=g(n)+h(n)$: coste estimado de la solución menos costosa que pasa por el nodo *n*. Tengamos en cuenta que si el nodo *n* ya se ha explorado, no se actualizará su valor de $f(n)$



- **Ejemplo de búsqueda A***

Escribiremos la secuencia de nodos frontera y explorados partiendo del nodo C y siendo G la meta.



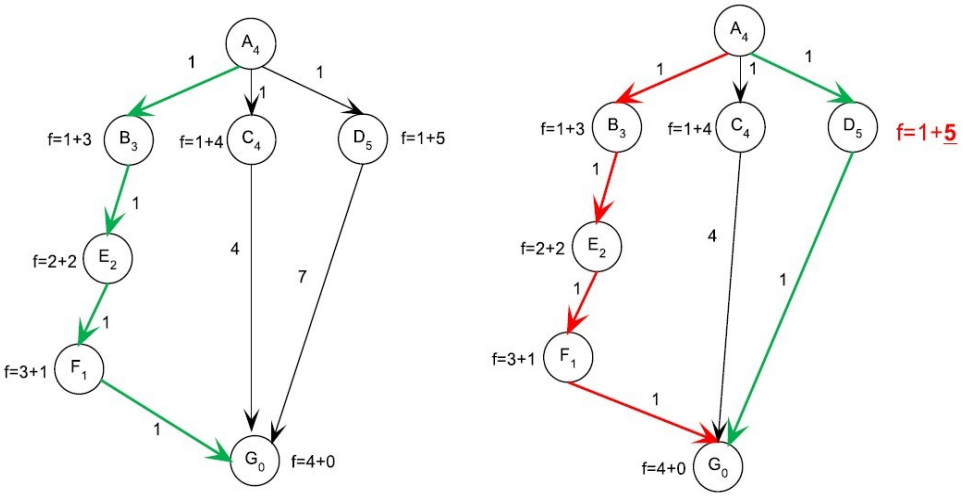
Paso	Frontera	Explorados
1	C(0 + 24)	-
2	B(5 + 15), E(9 + 12)	C(0)
3	E(9 + 12), D(7+5 + 10)	C(0), B(5)
4	D(7+5 + 10), F(9+17 + 12)	C(0), B(5), E(9)
5	F(9+17 + 12), **G(14+7+5 + 0)**	C(0), B(5), E(9), D(12)
6	F(9+17 + 12)	C(0), B(5), E(9), D(12), **G(26)**

- **Optimización de A*: admisibilidad y consistencia**

La búsqueda A* basada en árbol es óptima si la heurística es **admisible**: nunca sobreestima el coste real de alcanzar la meta. Por otro lado, la búsqueda A* basada en grafo es óptima si la heurística es **consistente** (óptima).

Si $g(n)$ es el coste real para llegar desde n, y sabemos que $f(n)=g(n)+h(n)$, entonces $\boldsymbol{f(n)}$ **nunca sobreestimará el coste real** de una solución por el camino que pase por *n*. En nuestro problema de las rutas en Rumanía, la distancia en línea recta es una heurística admisible porque es la distancia más corta entre dos puntos, y por ello nunca podrá sobreestimarse (valer más de lo que realmente vale).

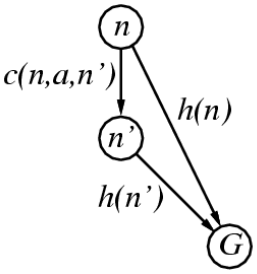
Pongamos un ejemplo sencillo sobre los **efectos de la sobreestimación** de la heurística. En el grafo de la izquierda, la heurística h subestima al coste real, por lo que se elige el camino óptimo para llegar al nodo meta, que es $A \rightarrow B \rightarrow E \rightarrow F \rightarrow G$. Sin embargo, tomando el mismo grafo y cambiando el coste del camino que une a D y G (pasa de costar 7 a costar 1), ahora vemos que el camino óptimo es $A \rightarrow D \rightarrow G$, pero dado que $h(D)=5 > h(B)=3$, no se toma como primera alternativa al camino $A \rightarrow D$, por lo que no estamos seleccionando el camino más óptimo en este caso. Esto se debe a que h sobrestima el coste real de la solución óptima (que es 1+1=2), llegando así a la meta por un camino más largo (1+1+1+1=4).



La búsqueda A* puede emplearse para encontrar el camino con el mínimo coste total, o el camino más rápido posible en el menor número de pasos (solo cuando todas las aristas tienen coste g uniforme). Sin embargo, A* se comporta como una **búsqueda en anchura** cuando $h=0$ y g se incrementa en 1 unidad, por lo que los nodos con igual valor de f se ordenan de menos a más reciente. Por otro lado, A* se comporta como **búsqueda en profundidad** cuando $g=h=0$, de modo que los nodos se ordenan de más a menos reciente.

Cuando existen distintos caminos hacia un mismo nodo, la búsqueda A* basada en grafo seleccionará siempre el primer camino generado, que puede no ser el óptimo. Como soluciones posibles podemos descartar el camino más costoso de entre los que llevan al mismo nodo. También podríamos asegurar que el camino óptimo a cualquier nodo repetido sea siempre el primero que se encuentra, lo cual logramos haciendo que nuestra heurística tenga una determinada consistencia.

Una **heurística h es consistente** si, para cada nodo n y cada sucesor n' de n, el coste estimado para llegar a la meta desde n $(h(n))$ no es mayor que la suma entre el coste de llegar de n a n' mediante la acción a (denotado por $c(n, a, n')$) más el coste estimado de llegar a la meta desde n' $(h(n'))$.



$$h(n) \leq c(n, a, n') + h(n') \quad \text{(desigualdad triangular)}$$

Si h es consistente, entonces los valores de f por cualquier camino no son decrecientes, lo cual provoca una **monotonía**, es decir: el primer nodo meta seleccionado para expandir deberá ser una solución óptima, dado que todos los nodos posteriores a expandir serán, por lo menos, igual de costosos.

■ Heurísticas aceptables para A*: problema del *8-puzzle*

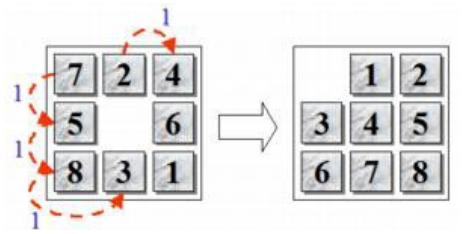
El principal inconveniente de A* no es su tiempo de ejecución, sino la memoria que consume: se queda antes sin espacio que sin tiempo (porque almacena todos los nodos), lo cual no resulta útil para espacios de estados muy grandes. Sin embargo podemos diseñar heurísticas aceptables que nos ayuden a explorar un menor número de nodos, minimizando así el gasto de memoria.

Supongamos el siguiente problema basado en un juego real, el ***8-puzzle***: Disponemos de un tablero de 3x3 celdas con 8 fichas numeradas del 1 al 8 y una celda vacía. El problema consiste en mover las fichas horizontalmente o verticalmente una sola celda y colocarlas en el hueco. El objetivo es conseguir que el tablero esté ordenado según algún criterio (generalmente en hueco en la celda superior izquierda y que los números estén ordenados de menor a mayor). El coste real de la solución es igual al número total de movimientos hasta llegar al tablero solicitado partiendo del estado inicial. Este puzzle suele resolverse en 22 pasos típicamente. Dado que el factor de ramificación es $b \approx 3$ (porque un hueco situado en el borde tiene 3 vecinos que podrían ocupar su espacio, pero si está en una esquina solo tiene 2, y si está en el centro tiene 4 vecinos), el árbol de búsqueda tendrá $3^{22} \approx 3{,}1 \cdot 10^{10}$ estados, mientras que el grafo de búsqueda tendrá 181.440 .

El gran número de estados que se pueden llegar a generar obliga a buscar una buena función heurística para buscar el camino más corto hasta la solución. Para ello debemos **abstraer el problema**: transformar el problema en otro más sencillo eliminando ciertas restricciones ("relajar el problema"), de modo que la función de coste siga siendo una heurística admisible y consistente para el problema original. Además, cuantas más restricciones tengamos en cuenta, más precisa será la heurística.

Para abstraer el el 8-puzzle se establece una sola operación básica: "mover una pieza de A a B". Solo estableceremos dos restricciones: se puede mover la pieza si A es adyacente a B (horizontal o verticalmente) y sólo si B está vacío. A causa de estas restricciones surgen tres posibles heurísticas h_n :

- h_1) Mover una pieza de A a B, si A es adyacente a B (solapamiento) → **Distancia Manhattan**



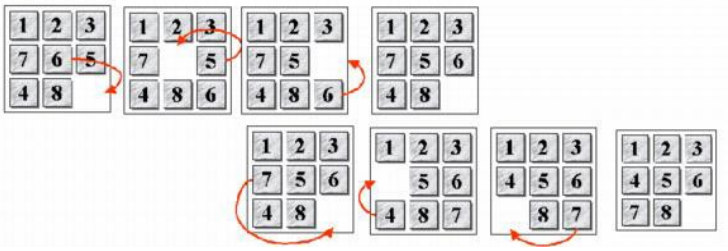
La heurística suma el número de movimientos necesarios de cada pieza hasta que llegue a ocupar su lugar correspondiente. Por ejemplo, los movimientos de cada pieza *p* son:

$$p(1)=3,\ p(2)=1,\ p(3)=2,\ p(4)=2,$$
$$p(5)=2,\ p(6)=3,\ p(7)=3,\ p(8)=2$$

En total necesitaremos $3+1+2+2+2+3+3+2 = 18$ **movimientos**. Sin embargo, sabemos que el coste de la **solución óptima es 26 movimientos**. No se llega a subestimar el coste real, por lo tanto h_1 es una heurística admisible, dado que cualquier pieza mal colocada debe moverse, por lo menos, una vez.

- h_2) Mover una pieza de A a B, si B está vacío (teletransportar la pieza). → **Distancia Gaschnig**

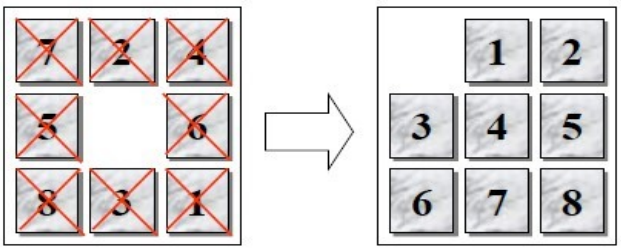
La heurística cuenta el número de intercambios de fichas que se pueden realizar. Dado que en una permutación somos capaces de reubicar correctamente 2 piezas, para intercambiar será necesario:



1) Situar A en el hueco temporalmente.
2) Mover B al hueco creado tras mover A (donde A estaba inicialmente), dejando así un hueco en la posición destino (donde estaba B). Ahora B está en la posición que le corresponde y falta A.
3) Mover A al hueco creado tras mover B (la posición inicial de B). Así se han intercambiado A y B.

Podemos ver que por cada dos fichas en posiciones que no les corresponden se realizarán 3 movimientos. Para este tablero podemos: $intercambiar(p(5), p(6))$ e $intercambiar(p(4), p(7))$, haciendo así 2 intercambios, de modo que realizaremos **6 movimientos**. Sabemos que el coste de la **solución óptima es 18 movimientos**, así que tampoco llega a subestimarse el coste real. Por ello es una heurística admisible: cualquier movimiento supone acercar una pieza un paso más a la meta.

- h_3) Mover una pieza de A a B (solapamiento y teletransporte).



Esta heurística consiste en contar el número de piezas que están mal colocadas. En el tablero de la izquierda podemos ver que hay 8 piezas mal colocadas, así que la heurística dice que necesitamos **8 movimientos**. El coste de la **solución óptima es 26 movimientos**, por lo que tampoco llega a subestimarse el coste real.

Como podemos ver, siempre es mejor emplear aquellas heurísticas que tenga valores más altos que otras heurísticas, siempre y cuando dicha heurística no esté sobreestimada. Para un problema dado es posible que existan $h_1, h_2, \ldots, h_n$ heurísticas, pero definimos una **heurística dominante** como aquella que alcanza los mayores valores para todos los nodos *n*. Esto es así porque si el valor de la heurística es el mayor posible y no está subestimado, será el valor más próximo al coste real, por lo que encontrará el mejor camino al emplearse con A* (también se dice que la "heurística dominante es *la más informada*"). Entre los ejemplos previos decimos que h_2 es la heurística dominante pues $h_2 \geq h_i(n) \quad \forall n,\ i \neq 2$.

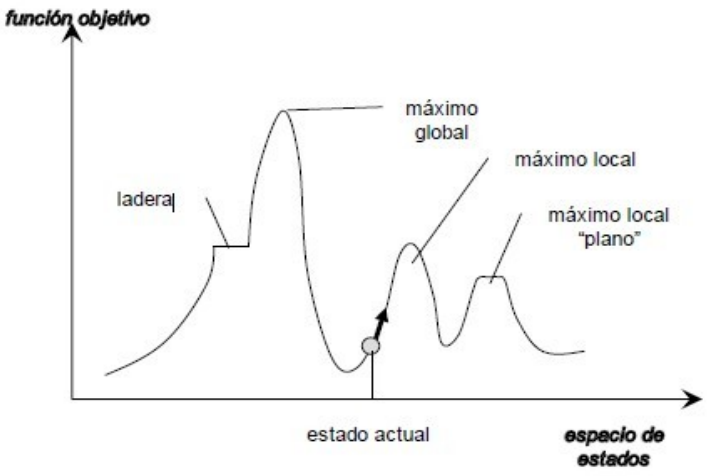
ESTRATEGIAS DE BÚSQUEDA INFORMADA LOCAL

Estas estrategias tratan de mejorar el estado actual en vez de explorar sistemáticamente todos los caminos desde el estado inicial. Aquí el coste del camino no resulta de interés y lo único importante es alcanzar el estado meta. En esta búsqueda también conocemos la información de todos los estados. Para ello comprueban constantemente el espacio de búsqueda y solo conservan unos de los pocos mejores caminos que existen hacia la meta, comprobando cuales han sido ya explorados y cuales no. Cuando encuentran una meta óptima, ya conocerán el camino hacia esta.

La principal diferencia de la búsqueda local es que solo manipula un solo estado actual (sin fronteras) y solo se desplaza hacia los estados vecinos, por lo que el camino hacia la meta es irrelevante (por ello los caminos no se retienen en memoria). Estas estrategias no son sistemáticas, pero tienen dos ventajas: consumen una cantidad escasa pero constante de memoria y encuentran soluciones razonables en espacios de estados grandes (o incluso infinitos), por lo que resultan aptos para problemas de optimización.

- **Algoritmo de escalada (*Hill-climbing*)**

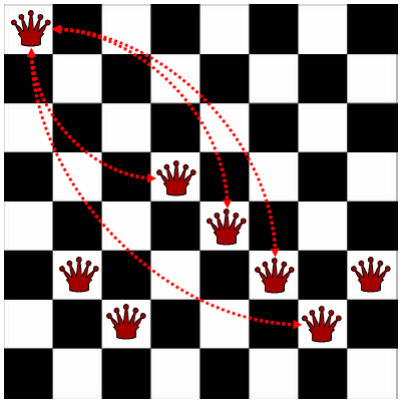
Es una variante de la búsqueda en profundidad. Para seleccionar el siguiente nodo a expandir se realiza una medición heurística previa para estimar la distancia que queda por recorrer hasta la meta. Se puede representar gráficamente el valor de la heurística para cada nodo, de modo que buscamos encontrar un **máximo global** (punto más alto de la función).



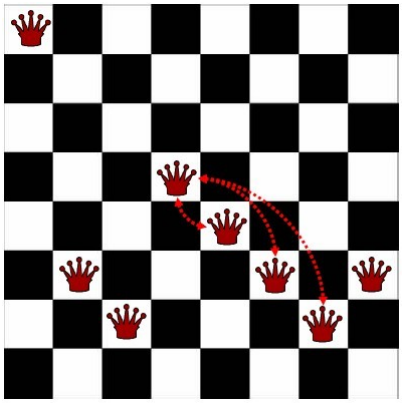
El siguiente nodo se elige en la dirección en la que el valor se incrementa, y se termina cuando ningún descendiente tiene mejores valores. Así no se mantiene ningún árbol de búsqueda, tan solo el nodo actual.

■ **Problema de las 8 reinas mediante algoritmo de escalada**

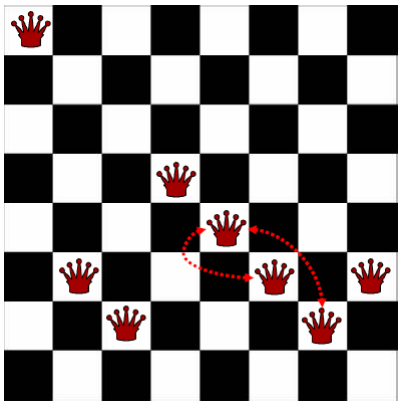
Supongamos que tenemos un tablero de ajedrez de 8x8 celdas y queremos poner 8 reinas de forma que ninguna de ellas se puedan atacar. Nuestra operación disponible (acción) es mover una reina dentro de la misma columna. La heurística medirá el número de pares de reinas que se atacan en cada posición. En nuestro caso el ínfimo valor que puede tomar la heurística es $h=12$. Supongamos el caso en el que ya tenemos 7 reinas colocadas desde la 2ª hasta la 8ª columna y queremos poner la reina en la 1ª columna; comenzando por la primera fila la heurística será $h=18$. Veamoslo gráficamente:



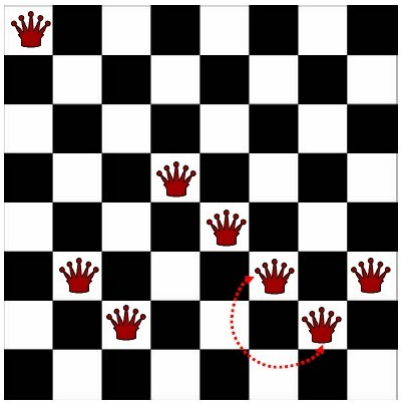
4 pares se atacan



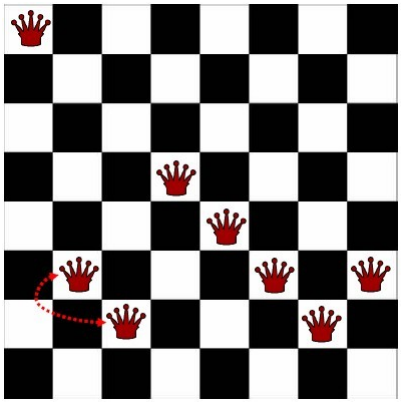
3 pares se atacan



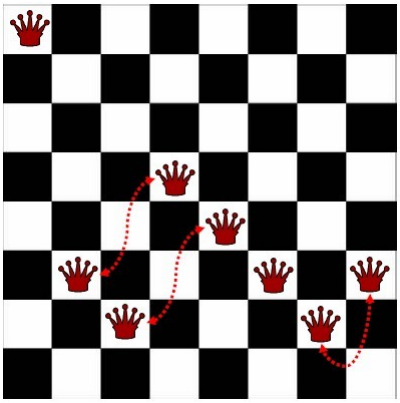
2 pares se atacan



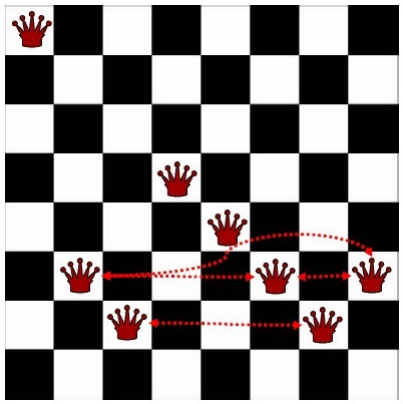
1 par se ataca



1 par se ataca



3 pares se atacan



4 pares se atacan

La búsqueda local tiene como inconvenientes las siguientes situaciones:

- **Máximo local**: estado mayor que cualquiera de sus vecinos pero menor que otros estados más alejados (los máximos globales). Puede parecer una solución válida, pero no es la óptima. Se soluciona con ***backtracking***: volver a algún nodo anterior e intentar expandir otro descendiente distinto.
- **Meseta**: región en la que todos los estados vecinos tienen el mismo valor heurístico, por lo que no es posible determinar la mejor dirección para avanzar. Se soluciona realizando con un **salto de gran distancia** en el espacio de búsqueda para llegar a otra región donde sí haya variaciones en los valores heurísticos.
- **Cresta**: región en la que los estados tienen mayores heurísticas que las regiones vecinas, pero a las que no podemos llegar viajando únicamente a los estados vecinos (como si fuese un conjunto de máximos locales agrupados en linea recta). Se soluciona **comprobando más de un descendiente** antes de aplicar la prueba de meta al mejor descendiente.

Tema 3:

REPRESENTACIÓN DEL CONOCIMIENTO

Fundamentos de Intelixencia Artificial

Vicente Moret, Bertha Guijarro, Eduardo Mosqueira, Mariano Cabrero, Amparo Alonso

MÉTODOS ESTRUCTURADOS

Resulta útil tener estructuras para representar información que permita agrupar propiedades de objetos complejos y nos permita obtener descripciones de ellos y para ello nos resulta bastante cómoda la lógica. También sería útil poder representar determinados escenarios o secuencias de acontecimientos comunes de manera eficaz para poder estudiarlos.

Los esquemas no formales de representación del conocimiento cumplen estas propiedades:

- **Adecuación representacional**: debe ser capaz de representar las distintas clases de conocimiento que se tiene sobre el dominio.
- **Adecuación inferencial**: ha de permitir manipular el conocimiento para crear conocimiento nuevo.
- **Eficiencia inferencial**: el esquema debe ser versátil, usando información que permita optimizar el proceso de inferencia de nueva información.
- **Eficiencia adquisicional**: debe proporcionar vías para añadir nuevos conocimientos y datos.

A la hora de clasificar los esquemas de representación estructurados hay dos clases de métodos:

- **Métodos declarativos**: representan el conocimiento como una colección estática de hechos, que solo se pueden manipular con un conjunto genérico y restringido de procedimientos. En otras palabras, se dice qué hacer con los datos una única vez, sean del tipo que sean los datos.

  Sus **ventajas** son que aquellos hechos verídicos del dominio se almacenan una sola vez y resulta más sencillo aumentar el conocimiento sin modificar el ya existente. Aquí se incluyen métodos como las **redes semánticas**, que permiten describir objetos y acontecimientos (hechos) al mismo tiempo, o los **marcos (*frames*)**, que son estructuras que permiten representar desde distintos puntos de vista a objetos muy complejos.

- **Métodos procedimentales**: representan cada tipo de conocimiento con un procedimiento concreto para cada uno, lo cual hace que sea un método bastante dinámico. En otras palabras, para cada conocimiento concreto hay que decir qué se hará con él.

  Las **ventajas** son que ponen un mayor énfasis en la capacidad del sistema para inferir nuevo conocimiento. Para ello incorporan conocimiento heurístico y manipulan distintos modelos y técnicas de razonamiento (por ello son dinámicos) para generar nuevo conocimiento, pero siempre basado en un punto de vista de la probabilidad matemática. Aquí se incluyen métodos como las **reglas de producción**, que permiten generar conocimiento cuando se cumplen unas determinadas condiciones (básicamente es usar `IF-THEN-ELSE`).

REDES SEMÁNTICAS

Consisten en representar el conocimiento mediante conceptos (elementos) y las relaciones que existen entre ellos mediante un grafo y se emplean para hacer mapas conceptuales (esquemas). Los nodos representan a un concepto y las aristas representan la relación entre ambos conceptos, las cuales son unidireccionales de modo que para establecer una doble relación hay que trazar dos aristas distintas. Puede decirse que un enlace es una relación binaria entre nodos y por ello puede haber varias relaciones:

- **Ocurrencia (pertenece, $\in$)**: se relaciona un elemento con una categoría (grupo de elementos).
- **Generalización (`ES_UN`)**: relaciona una entidad concreta con otra entidad más genérica.
- **Agregación (`ES_PARTE_DE`)**: relaciona componentes de un objeto con el propio objeto.

- **Acción**: vincula una entidad con otra mediante una acción (un verbo, una responsabilidad, etc).
- **Propiedades**: relaciona objetos con características de esos objetos (propiedad, descripción, etc).

Milú `ES_UN` perro.
Un perro `ES_UN` animal.
Milú `ES_UN` animal.

La nariz `ES_PARTE_DE` la cara.
La cara `ES_PARTE_DE` la cabeza.
La nariz `ES_PARTE_DE` la cabeza.

A nivel computacional, una red semántica se implementa con una tabla de *n-tuplas* que almacenan la información `Objeto-Atributo-Valor`, de modo que: el **nodo origen** sea el `Objeto`, el **nodo destino** sea el `Valor` y la **arista** que los uno sea el `Atributo`.

Objeto	Atributo	Valor
Ana	`TIENE`	Dinero
Ana	`ES_UN`	Persona
Persona	∈	Mamífero
Mamífero	`ES_UN`	Animal

Comparemos cómo se representaría el conocimiento usando lógica formal y una red semántica:

Lógica formal

$TIENE(Dinero, Ana)$

$\text{ES\_UN}(Persona, Ana)$

$\in(Mamífero, Persona)$

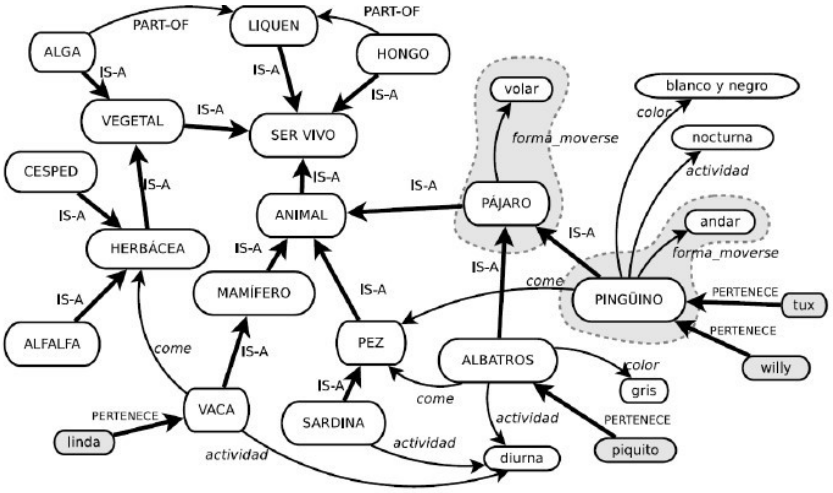
$\text{ES\_UN}(Animal, Mamífero)$

Red semántica

(**Ana**
$(TIENE(Dinero))$
$(\text{ES\_UN}(Persona))$)
(**Persona**
$(\text{ES\_UN}(Animal))$)
(**Mamífero**
$(\text{ES\_UN}(Animal))$)

Una relación existente puede **generar nuevo conocimiento** haciendo uso de:

- **Herencia de propiedades**: cualquier propiedad que sea cierta para una clase de elementos debe ser cierta para cualquier elemento de la clase que los contenga.
- **Razonamiento**: puede realizarse de dos maneras:
  - **Por rastreo**: se siguen las relaciones entre dos objetos y se establece una nueva relación que los conecte, aunque puede suceder que la inferencia realizada no sea válida.
  - **Emparejamiento**: se crean nuevos fragmentos de la red semántica y se agregan a la red principal estableciendo nuevas relaciones. El problema es buscar qué relaciones son las correctas al unir las redes.



MARCOS (*FRAMES*)

Hay que resaltar que, cuando abordamos un nuevo problema, nadie comienza por analizarlo a fondo para ir construyendo estructuras que representen el conocimiento necesario para resolverlo, sino que el primer paso es recordar experiencias anteriores y ver si son semejantes al problema planteado. Los ***frames*** hacen justamente eso: intentar representar el problema mediante un razonamiento por semejanzas, y para ello describen clases de objetos, que son representaciones estructuradas del conocimiento sobre una entidad (como si fuera el índice de un libro con más detalles). La **ventaja** de los *frames* es que permiten definir procedimientos para inferir rápidamente conocimiento aún a pesar de tener información incompleta o que no está representada explícitamente. Un *frame* consta de:

- **Cabeza**: le da nombre al frame y representa la clase de objetos que se describe.
- **Slot**: elementos que representan a un objeto o una propiedad del *frame*. Pueden anidarse, de modo que la profundidad de un *slot* representa el nivel de conocimiento que tiene y su contenido se especializa a medida que profundiza en los niveles (cada nuevo sangrado es un nuevo nivel).

Para obtener nuevos conocimientos mediante semejanza se hace uso de la **relación de herencia** (`ES_UN`). Pongamos el ejemplo de la representación de un objeto `Pájaro` y como representar un `Gorrión` mediante la generación de nuevo conocimiento:

```
(Pájaro
    (MORFOLOGÍA
        (plumas)
        (pico)
        (...)

    (REPRODUCCIÓN
        (ovípara))
        (...)
)

(Gorrión
    (ES_UN (Pájaro))
    (TAMAÑO (pequeño))
    (COLOR (pardo))
)
```

```
(Gorrión
    (MORFOLOGÍA
        (plumas)
        (pico)
        (...)

    (REPRODUCCIÓN
        (ovípara))
        (...)

    (TAMAÑO
        (pequeño))
    (COLOR
        (pardo))
    (...)
)
```

- **Procedimientos (*demons*)**

Existe un conjunto de procedimientos para generar nuevos conocimientos que están inactivos la mayoría del tiempo, pero que se activan cuando se cumplan determinadas condiciones para ejecutar acciones concretas. Por ejemplo: `IF_NEEDED`, `IF_ADDED`, `IF_REMOVED`, `IF_STORED`, `IF_RETRIEVED`. Cuando un *demon* se activa por un valor en una entrada del frame (dentro del nivel que le corresponda) se desencadenará una acción con el nombre de `D_"nombreAcción"` y luego el *demon* vuelve a ponerse inactivo.

```
Base_de_reglas
    IF_REMOVED
        D_reglas_de_eliminado
    REGLAS
        Regla_1
        Regla_2
        (...)
    PARAMETROS_DE_LOS_IF
        Parametro_1
            Regla_1
```

REGLAS DE PRODUCCIÓN

Son esquemas empleados para representar el conocimiento procedimental que están basados en **premisas** o condiciones (`IF`), que en caso de ser verdaderas pasan a ejecutar una **acción** o generan una **conclusión** (`THEN`), y que en caso de ser falsas ejecutarán otra acción u otra conclusión (`ELSE`). La premisa debe estar construida con la lógica formal y los operadores `AND (^)`, `OR (∨)` o `NOT (¬)` y pueden anidarse varias premisas.

La principal ventaja de las reglas de producción es que las condiciones y las acciones/conclusiones involucradas son explícitas, por lo tanto sabemos cuales podrían ser los posibles resultados en función del valor lógico que tomen las condiciones. Por lo general buscamos que las reglas de producción conformen una **unidad completa de razonamiento**, es decir, que una única regla englobe todas las acciones o conclusiones que estén relacionadas con las premisas que se evalúen en el `IF`.

Podemos realizar tanto un razonamiento usando lenguaje natural como con lenguaje numérico o matemático; veamos un ejemplo de un diagnóstico médico empleando ambos:

Regla_Diagnóstico_Sanguíneo
IF: (1) (*presión arterial sistólica*) > 160
And: (2) (*presión arterial diastólica*) > 95
And: (3) (*presión arterial media*) > 130
THEN: (1) (*diagnóstico sanguíneo* = hipertensión_arterial)

Regla_Diagnóstico_Respiratorio
IF: (1) (*gases arteriales* CO_2) = *hipercapnia*
And: (2) (*gases arteriales pH*) *acidemia*
And: (3) (*gases arteriales bicarbonatos*) = *normal*
THEN: (1) (*diagnóstico respiratorio* = *acidosis respiratoria*)

```
diagnóstico(
    (sanguíneo
        (hipertensión_arterial) )
    (respiratorio
        (acidosis
            (respiratoria) )
)
```

inferencia

```
presion (
    (arterial
        (sistólica(177))
        (diastólica(99))
        (media(131))
)

gases (
    (arteriales
        (CO2
            (hipercapnia))
        (pH
            (acidemia))
        (bicarbonatos
            (normal))
)
```

TIPOS DE REGLAS

- **IF_ALL**

Todas las sentencias de la premisa han de ser ciertas para que se ejecute la acción o se genere una conclusión en la parte del `THEN`. Es equivalente a una regla en la que todas las sentencias de la parte condicional están anidadas con operadores `AND (^)`.

- **IF_ANY**

Todas las sentencias de la premisa están conectadas con operadores `OR (∨)`, de modo que basta con que tan solo una de ellas sea verdadera para ejecutar la acción o generar la conclusión en el `THEN`.

- **IF_SOME**

Equivale a la regla `IF_ANY`, pero se evalúan todas las sentencias de la premisa, aún a pesar de que exista al menos una de ellas que sea verdadera. Permite realizar una búsqueda exhaustiva de todos los valores que toma la premisa.

Tema 3:

SISTEMAS DE PRODUCCIÓN

Fundamentos de Intelixencia Artificial

Vicente Moret, Bertha Guijarro, Eduardo Mosqueira, Mariano Cabrero, Amparo Alonso

SISTEMAS DE PRODUCCIÓN

SISTEMAS DE INFERENCIA

Existen dos paradigmas a la hora de realizar un programa:

- **Programas secuenciales**: son aquellos que dependen de los datos, las condiciones iniciales, parámetros, respuestas introducidas por los usuarios, resultados de cálculos previos, etc. En ellos el flujo de control a seguir y el uso de los datos está programado implícitamente en el código. El **defecto** de estos programas está en su secuencialidad: las bifurcaciones que se ejecutan al manipular la información solo se realizan en puntos concretos dentro del código del programa. Si quisiésemos emplearlos en entornos que están cambiando constantemente, el hecho de que existan bifurcaciones y tengamos que elegir entre uno o más caminos sería lo más común, y no hacerlo ocasionalmente.
- **Programas basados en eventos**: se adecúan a las situaciones donde hay estímulos continuos (tanto del entorno como externos) que obligan a tomar decisiones sobre el flujo de ejecución. El programa debe responder a dichos estímulos tomando un camino u otro, siendo dichos datos inesperados (no como en los secuenciales, donde se sabe de dónde se obtienen los datos). Aquí no se usan estructuras de control ni se sabe en qué punto del programa habrá bifurcaciones, pero los conocidos como **sistemas de inferencia dirigidos por patrones (SIDP)** deben ser capaces de reconocer patrones en los datos para poder ejecutar determinados trozos del código y realizar las tareas correspondientes.

Un SIDP emplea una estructura modular, es decir, cada módulo se encarga de detectar las distintas situaciones (reconocer los patrones) y crear la respuesta adecuada. Los módulos se dividen según su función en **antecedente** (lado izquierdo) y **consecuente** (lado derecho). El primero se encarga de acceder a los datos para verificarlos y compararlos con “plantillas” (los patrones) del módulo, mientras que el segundo se encarga de modificar y/o escribir los datos generados. Estos módulos también se denominan “reglas”, de modo que los SIDP compuestos por reglas también se denominan **Sistemas de Producción** o Sistemas Basados en Reglas (SBR).

Sistemas de Producción

Un sistema de producción es un sistema inteligente basado en unas reglas que trabajan con una base de hechos y usan mecanismos de emparejamiento que forman parte explícita de su arquitectura.

- **Sistemas dirigidos por los datos (*dirigidos por antecedentes*)**

Las inferencias (resultados) se obtienen cuando los antecedentes de alguna(s) de sus reglas de producción se emparejan con, al menos, una parte de los hechos que describen el estado actual. Cuando se produce el emparejamiento decimos que “se ha activado dicha regla” y que puede ser ejecutada. Según la estrategia de exploración que tenga asignada, dicha regla puede ejecutarse o no. Son sistemas poco específicos porque ejecutan todas las reglas disponibles en función de la información que reciban.

- **Sistemas dirigidos por los objetivos (*dirigidos por consecuentes*)**

Tanto los antecedentes como los consecuentes de las reglas deben ser ciertos al compararlos con los datos. Si es así, las reglas son activadas regresivamente (hacia atrás) y el emparejamiento se realiza a través de las conclusiones de las reglas (los resultados). Para alcanzar una meta se sigue un proceso recursivo en donde los antecedentes (entradas) de un módulo son los consecuentes (salidas) de su módulo anterior. Son sistemas más específicos porque su ejecución conlleva un proceso de búsqueda.

ARQUITECTURA

- **Base de conocimientos**

Este componente describe el dominio para el que el sistema de producción debe plantear sus soluciones. Está formado por dos elementos que deben comunicarse continuamente entre ellos, por lo que la información que almacenan/generan debe ser fácilmente comprensible por ambos. Consta de:

- **Base de Hechos (BH)**: almacena y manipula todos los hechos importantes del dominio (todos los conocimientos que tiene), formando el esqueleto declarativo del sistema.
- **Bases de Reglas (BR)**: permite construir circuitos de inferencia a través de los cuales es capaz de generar nuevas conclusiones válidas, formando el esqueleto procedimental del sistema.

Pongamos un ejemplo basado en la meteorología. En nuestra BH tenemos 3 marcos (*frames*) que almacenan información sobre el `Atardecer`, el `Cielo` y el `Pronóstico` meteorológico. Así mismo, en nuestra BR tenemos una regla que permite inferir si hará buen o mal tiempo según el estado del cielo, y otra regla para determinar qué acción debería realizarse según el valor del pronóstico.

Regla_Pronóstico_Metereológico
IF: (*Atardecer* COLOR) = *Rojo*
And: (*Cielo NUBES*) = *Ausencia*
And: (*Cielo TONALIDAD*) = *Normal*
THEN: (*Pronóstico METEREOLÓGICO* = Buen_Tiempo)

Regla_Actividad_Final
IF: (*Pronóstico METEREOLÓGICO*) = Buen_Tiempo
THEN: (*Actividad RECOMENDADA* = Ir_Al_Campo)
AND: (*Actividad DESCARTADA* = Ir_Al_Cine)

Atardecer (
(COLOR (...))
(...)
)
Cielo (
(NUBES (...))
(TONALIDAD (...))
(...)
)
Pronóstico (
(METEREOLOGÍA (...))
(...)
)
Actividad (
(RECOMENDADA (...))
(DESCARTADA (...))
(...)
)

Hay que resaltar que cada regla emplea los *frames* cuyo contenido (*slots*) se adecúe a los valores que necesita evaluar. Es por eso que la segunda regla emplea su propio marco (el último).

- **Memoria activa**

Estructura que almacena toda la información estática necesaria para resolver el problema. Contiene información como: los datos iniciales del problema, los datos añadidos posteriormente, aquellos hechos generados como resultado durante los procesos de inferencia y las hipótesis de trabajo (las metas o submetas) que todavía no han sido resueltas. La memoria activa almacena todos los cambios de estado del sistema, de modo que su contenido siempre representará el estado actual del sistema. Se encarga principalmente de interactuar con el mundo exterior para recibir aquella información que el sistema no es capaz de inferir por sí mismo (como los datos que el usuario inserta).

Cuando el proceso de inferencia se detiene, la memoria activa almacenará el estado final del problema en donde se incluirán los datos, hechos e hipótesis que ha manipulado e inferido. Todos los hechos y datos se corresponden con entidades dentro de la BH, pero con valores concretos asignados. Los datos existentes representan información que procede del mundo real y las hipótesis resueltas son los resultados, cuyos valores se han de investigar y verificar si son correctos con la realidad.

Para el ejemplo del pronóstico meteorológico, la memoria activa podría almacenar inicialmente 3 valores en sus *frames*, como que <Atardecer COLOR = Rojo>, <Cielo NUBES = Ausencia> y <CIELO Tonalidad = Normal>. Inicialmente la única hipótesis planteada para resolver es comprobar si **<Actividad RECOMENDADA = Ir_Al_Campo>** es una afirmación correcta o no. Para ello ejecutamos un proceso de inferencia dirigido por los objetivos, es decir, a partir de la conclusión obtenemos los hechos.

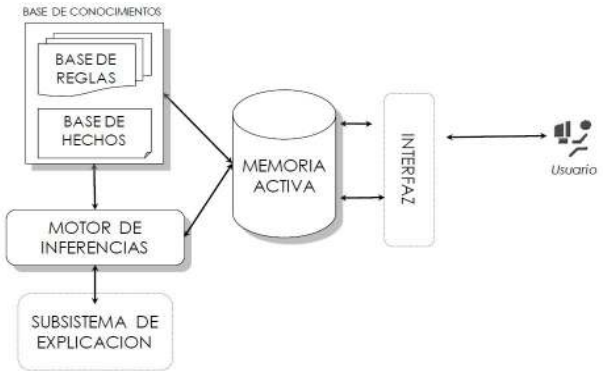
Comenzamos investigando aquellas reglas que aporten alguna conclusión relacionada con <Actividad RECOMENDADA>, que en este caso es Regla_Actividad_Final. Vemos que esta regla tiene como antecedente a <Pronóstico METEREOLÓGICO> y que para que el resultado sea Ir_Al_Campo debe cumplirse que <Pronóstico METEREOLÓGICO = Buen_Tiempo>, por lo que añadimos esta hipótesis a la memoria activa. Esta hipótesis aparece como una conclusión en Regla_Pronóstico_Metereológico, así que repetiremos el proceso de análisis en esta regla.

Dado que los datos de sus antecedentes ya estaban cargados previamente en la memoria activa, y dichos datos permiten hacer verdadera la conclusión de esta regla, se deduce que el hecho es verdadero, confirmando así la hipótesis planteada y añadiendo la conclusión y su resultado a la memoria activa para futuras inferencias. De este modo se actualiza la memoria activa con los resultados de las inferencias, y dado que no hay más hipótesis para comprobar, el estado actual de la memoria activa tras este proceso de inferencia sería el siguiente:

Atardecer (
(COLOR *Rojo*)
)
Cielo (
(NUBES *Ausencia*)
(TONALIDAD *Normal*)
)
Pronóstico (
(METEREOLOGÍA *Buen_Tiempo*)
)
Actividad (
(RECOMENDADA *Ir_Al_Campo*)
(DESCARTADA *Ir_Al_Cine*)
)

- **Motor de inferencias**

El Motor de Inferencias (MI) está separado físicamente separado de la sección que manipula el conocimiento y está formado por un Intérprete de Reglas (IR) y una Estrategia de Control (EC). El MI se trata de un programa secuencial cuyo objetivo es determinar el siguiente paso a ejecutar a la hora de inferir un resultado. Las principales funciones del MI son:



- Examinar la memoria activa y determinar qué reglas deben ejecutarse, para lo cual emplea una estrategia de búsqueda (del espacio de estados) y una estrategia de resolución de conflictos. Esto le permite encontrar conexiones entre los estados iniciales del problema y los estados meta.
- Controlar y organizar la ejecución de las reglas seleccionadas en el paso anterior.
- Actualizar la memoria activa cuando sea necesario con los datos que se generen en el proceso.
- Asegurar que el sistema mantiene un estado de autoconocimiento: conocer qué reglas se han activado y cuales se han ejecutado, cuales fueron los últimos datos agregados a la memoria activa, gestionar las prioridades de las reglas que se ejecutan, etc.

La EC examina la memoria activa y elige qué regla ejecutar según los **ciclos básicos del sistema de producción** y atendiendo también a ciertos parámetros, como el criterio de activación de la regla, las estrategias de búsqueda a seguir o la dirección de avance por el espacio de estados.

CICLOS BÁSICOS DEL SISTEMA DE PRODUCCIÓN

EJEMPLO DE CICLO EN UN SISTEMA DIRIGIDO POR DATOS

Supongamos que nuestro motor de inferencia tiene las siguientes **restricciones** aplicadas:

- Empleará un **encadenamiento progresivo**, es decir, sigue una inferencia dirigida por datos.
- Activará todas aquellas reglas que emparejen con la memoria activa.
- Realiza una **búsqueda en profundidad** en el espacio de estados, ejecutando la primera regla de entre aquellas que hayan sido activadas más recientemente.
- No ejecutará dos veces la misma regla.
- La inferencia terminará cuando la hipótesis (*H*) sea un hecho demostrado y se haya agregado a la memoria activa.

Su **funcionamiento** es el siguiente: el emparejador examina los antecedentes de las reglas y selecciona aquellas que se corresponden con los hechos y datos existentes en la memoria activa. Ejecutará la primera regla de las reglas activadas y su resultado se agregará a la memoria activa. Finalmente comprobará si en la memoria activa aparece *(H)* como un hecho demostrado; en caso contrario, repetirá el ciclo hasta encontrar una solución o hasta que todas las reglas hayan sido ejecutadas y no se llegue a ninguna solución. Los datos iniciales de nuestro ejemplo son los siguientes:

➢ **Base de reglas**

$R1:\ IF\ (X \text{ And } Y)\ THEN\ Z$
$R2:\ IF\ (C \text{ And } D)\ THEN\ G$
$R3:\ IF\ (E \text{ And } V)\ THEN\ H$
$R4:\ IF\ (A \text{ And } B)\ THEN\ C$
$R5:\ IF\ (F \text{ Or } G)\ THEN\ X$
$R6:\ IF\ (Z \text{ And } B)\ THEN\ V$
$R7:\ IF\ (E \text{ And } C)\ THEN\ F$

➢ **Memoria activa**

$M_0 = <A, B, D, Y, E, (H)>$

En verde se representan los datos de los que dispone la memoria activa, que son aquellos que se han demostrado como ciertos.

En azul y entre paréntesis se representan las hipótesis sin confirmar, es decir, aquellas que queremos demostrar o conocer su resultado.

Ciclo 1)

Disponemos de los datos *A* y *B* en memoria activa, por lo que se activa la regla R4. Añadimos R4 al conjunto de reglas ejecutadas y agregamos su resultado a la memoria activa. Como la hipótesis *H* aún no forma parte de la memoria activa, debemos seguir activando reglas en ciclos posteriores.

$R1:\ IF\ (X \text{ And } Y)\ THEN\ Z$
$R2:\ IF\ (C \text{ And } D)\ THEN\ G$
$R3:\ IF\ (E \text{ And } V)\ THEN\ H$
$\mathbf{R4:\ IF\ (A \text{ And } B)\ THEN\ C}$
$R5:\ IF\ (F \text{ Or } G)\ THEN\ X$
$R6:\ IF\ (Z \text{ And } B)\ THEN\ V$
$R7:\ IF\ (E \text{ And } C)\ THEN\ F$

Reglas activadas		Reglas ejecutadas	
R4	Ciclo 1	R4	Ciclo 1

$M_1 = <A, B, D, Y, E, (H), C>$

Ciclo 2)

Disponemos de *C* y *D* en memoria, por lo que se activa la regla R2, pero también tenemos *E* y *C*, por lo que también se activa la regla R7. Por seguir un orden, ejecutaremos las reglas de menor a mayor, por lo que en este caso ejecutaremos la regla R2, añadiendo así a la memoria activa el dato *G*. Dado que la hipótesis aún no forma parte de la memoria activa, continuamos activando reglas en ciclos posteriores.

$R1:\ IF\ (X\ And\ Y)\ THEN\ Z$
$\mathbf{R2:\ IF\ (C\ And\ D)\ THEN\ G}$
$R3:\ IF\ (E\ And\ V)\ THEN\ H$
$R4:\ IF\ (A\ And\ B)\ THEN\ C$
$R5:\ IF\ (F\ Or\ G)\ THEN\ X$
$R6:\ IF\ (Z\ And\ B)\ THEN\ V$
$\mathbf{R7:\ IF\ (E\ And\ C)\ THEN\ F}$

Reglas activadas		**Reglas ejecutadas**	
R2	Ciclo 2	R2	Ciclo 2
R7		R4	Ciclo 1
R4	Ciclo 1		

$M_2 = <A, B, D, Y, E, (H), C, G>$

Ciclo 3)

Disponemos de *G* en memoria, por lo que se activa la regla R5. Esto añade a memoria activa el dato *X*, y como la hipótesis aún no está en memoria activa, seguiremos activando reglas.

$R1:\ IF\ (X\ And\ Y)\ THEN\ Z$
$R2:\ IF\ (C\ And\ D)\ THEN\ G$
$R3:\ IF\ (E\ And\ V)\ THEN\ H$
$R4:\ IF\ (A\ And\ B)\ THEN\ C$
$\mathbf{R5:\ IF\ (F\ Or\ G)\ THEN\ X}$
$R6:\ IF\ (Z\ And\ B)\ THEN\ V$
$R7:\ IF\ (E\ And\ C)\ THEN\ F$

Reglas activadas		**Reglas ejecutadas**	
R5	Ciclo 3	R5	Ciclo 3
R2	Ciclo 2	R2	Ciclo 2
R7		R4	Ciclo 1
R4	Ciclo 1		

$M_3 = <A, B, D, Y, E, (H), C, G, X>$

Ciclo 4)

Disponemos de *X* e *Y* en memoria activa, por lo que se activa la regla R1 y obtenemos *Z*.

$\mathbf{R1:\ IF\ (X\ And\ Y)\ THEN\ Z}$
$R2:\ IF\ (C\ And\ D)\ THEN\ G$
$R3:\ IF\ (E\ And\ V)\ THEN\ H$
$R4:\ IF\ (A\ And\ B)\ THEN\ C$
$R5:\ IF\ (F\ Or\ G)\ THEN\ X$
$R6:\ IF\ (Z\ And\ B)\ THEN\ V$
$R7:\ IF\ (E\ And\ C)\ THEN\ F$

Reglas activadas		**Reglas ejecutadas**	
R1	Ciclo 4	R1	Ciclo 4
R5	Ciclo 3	R5	Ciclo 3
R2	Ciclo 2	R2	Ciclo 2
R7		R4	Ciclo 1
R4	Ciclo 1		

$M_4 = <A, B, D, Y, E, (H), C, G, X, Z>$

Ciclo 5)

Disponemos de *Z* y *B* en memoria, por lo que se activa la regla R6. Esto agrega *V* a la memoria.

$R1:\ IF\ (X\ And\ Y)\ THEN\ Z$
$R2:\ IF\ (C\ And\ D)\ THEN\ G$
$R3:\ IF\ (E\ And\ V)\ THEN\ H$
$R4:\ IF\ (A\ And\ B)\ THEN\ C$
$R5:\ IF\ (F\ Or\ G)\ THEN\ X$
$\mathbf{R6:\ IF\ (Z\ And\ B)\ THEN\ V}$
$R7:\ IF\ (E\ And\ C)\ THEN\ F$

Reglas activadas		**Reglas ejecutadas**	
R6	Ciclo 5	R6	Ciclo 5
R1	Ciclo 4	R1	Ciclo 4
R5	Ciclo 3	R5	Ciclo 3
R2	Ciclo 2	R2	Ciclo 2
R7		R4	Ciclo 1
R4	Ciclo 1		

$M_5 = <A, B, D, Y, E, (H), C, G, X, Z, V>$

Ciclo 6)

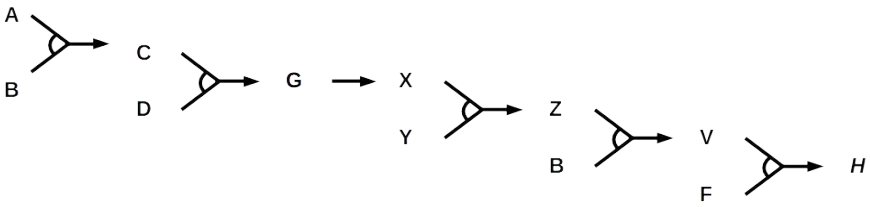
Disponemos de *E* y *V* en memoria, por lo que se activa la regla R3. Esto agrega *H* a la memoria, de modo que ya tenemos la hipótesis como resultado.

$R1:\ IF\ (X \text{ And } Y)\ THEN\ Z$
$R2:\ IF\ (C \text{ And } D)\ THEN\ G$
$\mathbf{R3:\ IF\ (E \text{ And } V)\ THEN\ H}$
$R4:\ IF\ (A \text{ And } B)\ THEN\ C$
$R5:\ IF\ (F \text{ Or } G)\ THEN\ X$
$R6:\ IF\ (Z \text{ And } B)\ THEN\ V$
$R7:\ IF\ (E \text{ And } C)\ THEN\ F$

Reglas activadas		**Reglas ejecutadas**	
R3	Ciclo 6	R3	Ciclo 6
R6	Ciclo 5	R6	Ciclo 5
R1	Ciclo 4	R1	Ciclo 4
R5	Ciclo 3	R5	Ciclo 3
R2	Ciclo 2	R2	Ciclo 2
R7		R4	Ciclo 1
R4	Ciclo 1		

$M_6 = <A, B, D, Y, E, (H), C, G, X, Z, V, >$

El motor de inferencias ha necesitado 6 ciclos. El **circuito inferencia**l generado es el siguiente:



EJEMPLO DE CICLO EN UN SISTEMA DIRIGIDO POR OBJETIVOS

Supongamos que nuestro motor de inferencia tiene las siguientes **restricciones** aplicadas:

- Empleará un **encadenamiento regresivo**, es decir, sigue una inferencia dirigida por datos.
- Activará todas aquellas reglas que emparejen con la memoria activa.
- Realiza una búsqueda en anchura en el espacio de estados.
- No ejecutará dos veces la misma regla.
- La inferencia terminará cuando la hipótesis (*H*) sea un hecho demostrado y se haya agregado a la memoria activa.

Su **funcionamiento** es el siguiente: el emparejador examina las conclusiones de las reglas (“hacia atrás”) y seleccionará aquellas que se correspondan con la hipótesis de la memoria activa. Se irán generando sucesivas subhipótesis que irán agregándose a la memoria activa, lo que se conoce como **retropropagación**. Una vez este proceso haya acabado, comenzará la fase de ejecución: las conclusiones se infieren “hacia adelante”, por lo que la hipótesis inicial puede ser verificada, lo cual ocurre cuando una regla es directamente ejecutable y pueda establecer una conclusión. Los datos iniciales de nuestro ejemplo son los mismos que el ejemplo anterior.

Ciclo 1)

Debemos buscar aquella regla cuya conclusión sea la hipótesis *H*, de modo que seleccionamos la regla R3: ésta tiene como antecedentes a *E* y *V*, pero como solo tenemos a *E* en la memoria activa, necesitamos activar otra regla que nos aporte a *V* como resultado (regla R6). Añadimos *V* a la columna de hipótesis porque la necesitaremos para resolver el problema planteado. Como *H* aún no está como un dato real en la memoria activa, seguiremos activando reglas. Aquí comienza el proceso “hacia atrás”.

R1:	IF (X And Y) THEN Z
R2:	IF (C And D) THEN G
R3:	**IF (E And V) THEN H**
R4:	IF (A And B) THEN C
R5:	IF (F Or G) THEN X
R6:	IF (Z And B) THEN V
R7:	IF (E And C) THEN F

Reglas ejecutadas	Reglas seleccionadas		Hipótesis	
	R3	Ciclo 1	V	Ciclo 1
			H	Ciclo 0

$M_1 = <A, B, D, Y, E, (H), (V),>$

Ciclo 2)

Para poder obtener *V* como resultado debe activarse la regla R6: ésta tiene como antecedentes a *Z* y *B*, pero como en la memoria activa solo tenemos a *B*, debemos activar aquella regla que nos aporte como resultado a *Z* (R1). Añadimos *Z* a la hipótesis porque la necesitaremos, agregamos *V* como dato, y dado que *H* aún no está en la memoria activa, seguiremos activando reglas.

R1:	IF (X And Y) THEN Z
R2:	IF (C And D) THEN G
R3:	IF (E And V) THEN H ◀
R4:	IF (A And B) THEN C
R5:	IF (F Or G) THEN X
R6:	**IF (Z And B) THEN V**
R7:	IF (E And C) THEN F

Reglas ejecutadas	Reglas seleccionadas		Hipótesis	
	R6	Ciclo 2	Z	Ciclo 2
	R3	Ciclo 1	V	Ciclo 1
			H	Ciclo 0

$M_2 = <A, B, D, Y, E, (H), (V), (Z),>$

Ciclo 3)

Para poder obtener *Z* como resultado debe activarse la regla R1: de sus antecedentes *X* e *Y* nos falta *X* en la memoria activa, necesitando activar la regla R5 para obtener *X* como resultado. Añadiremos *X* a la hipótesis porque la necesitaremos, agregamos *Z* como dato, y seguimos activando reglas porque *H* no forma parte de la memoria activa.

R1:	**IF (X And Y) THEN Z**
R2:	IF (C And D) THEN G
R3:	IF (E And V) THEN H ◀
R4:	IF (A And B) THEN C
R5:	IF (F Or G) THEN X
R6:	IF (Z And B) THEN V ◀
R7:	IF (E And C) THEN F

Reglas ejecutadas	Reglas seleccionadas		Hipótesis	
	R1	Ciclo 3	X	Ciclo 3
	R6	Ciclo 2	Z	Ciclo 2
	R3	Ciclo 1	V	Ciclo 1
			H	Ciclo 0

$M_3 = <A, B, D, Y, E, (H), (V), (Z), (X)>$

Ciclo 4)

Para obtener *X* necesitamos activar la regla R5: sus antecedentes son *F* o *G*, de los cuales no tenemos ninguno en memoria, así que necesitaremos activar las reglas R2 y R7 para obtenerlas. Agregamos *F* y *G* a la hipótesis porque necesitaremos alguna de ellas para poder activar R5, agregamos *X* como dato, y seguimos activando reglas porque *H* no forma parte de la memoria activa.

R1:	IF (X And Y) THEN Z ◀
R2:	IF (C And D) THEN G
R3:	IF (E And V) THEN H ◀
R4:	IF (A And B) THEN C
R5:	**IF (F Or G) THEN X**
R6:	IF (Z And B) THEN V ◀
R7:	IF (E And C) THEN F

Reglas ejecutadas	Reglas seleccionadas		Hipótesis	
	R5	Ciclo 4	F | G	Ciclo 4
	R1	Ciclo 3	X	Ciclo 3
	R6	Ciclo 2	Z	Ciclo 2
	R3	Ciclo 1	V	Ciclo 1
			H	Ciclo 0

$M_4 = <A, B, D, Y, E, (H), (V), (Z), (X), (F), (G)>$

Ciclo 5)

Disponemos de *Z* y *B* en memoria, por lo que se activa la regla R6. Esto agrega *V* a la memoria. Aún no tenemos la hipótesis en memoria, por lo que seguimos activando reglas. Por un lado, para obtener *G* (R2) necesitamos tener *C* y *D* en memoria, pero solo tenemos *D*; por el otro lado, para obtener *F* necesitamos tener *E* y *C*, pero solo tenemos *E*. Esto significa que necesitaremos obtener *C* para ambos casos, para lo cual necesitaremos activar la regla R4. Agregamos *C* a la hipótesis porque la necesitaremos posteriormente, agregamos *F* y *G* a los datos (porque obtendríamos ambos resultados a la vez), y seguimos buscando porque *H* no forma parte de la memoria activa.

R1: IF (X And Y) THEN Z ◀
R2: IF (C And D) THEN G
R3: IF (E And V) THEN H ◀
R4: IF (A And B) THEN C
R5: IF (F Or G) THEN X ◀
R6: IF (Z And B) THEN V ◀
R7: IF (E And C) THEN F

M_5 = <*A*, *B*, *D*, *Y*, *E*, (*H*), (*V*), (*Z*), (*X*), (*F*), (*G*), (*C*)>

Reglas ejecutadas	Reglas seleccionadas		Hipótesis	
	R2 | R7	Ciclo 5	C	Ciclo 5
	R5	Ciclo 4	F | G	Ciclo 4
	R1	Ciclo 3	X	Ciclo 3
	R6	Ciclo 2	Z	Ciclo 2
	R3	Ciclo 1	V	Ciclo 1
			H	Ciclo 0

Ciclo 6)

A partir de ahora comienza el proceso “hacia adelante”. La hipótesis actual que tenemos que demostrar es la última añadida a la columna de Hipótesis, la cual nos dice que tenemos que obtener *C*, para lo cual necesitamos ejecutar la regla R4, cuyos antecedentes son *A* y *B*, datos que ya están en memoria activa, por lo que ya podemos ejecutar la regla R4 y obtener *C* como dato real en memoria.

R1: IF (X And Y) THEN Z ◀
R2: IF (C And D) THEN G ◀
R3: IF (E And V) THEN H ◀
R4: IF (A And B) THEN C
R5: IF (F Or G) THEN X ◀
R6: IF (Z And B) THEN V ◀
R7: IF (E And C) THEN F ◀

M_6 = <*A*, *B*, *D*, *Y*, *E*, (*H*), (*V*), (*Z*), (*X*), (*F*), (*G*), *C*>

Reglas ejecutadas		Reglas seleccionadas		Hipótesis	
R4	Ciclo 6	R2 | R7	Ciclo 5	F | G	Ciclo 4
		R5	Ciclo 4	X	Ciclo 3
		R1	Ciclo 3	Z	Ciclo 2
		R6	Ciclo 2	V	Ciclo 1
		R3	Ciclo 1	H	Ciclo 0

Ciclo 7)

La hipótesis que ahora debemos resolver es la que nos dice que debemos obtener *F* o *G*, por lo que o bien podemos ejecutar R2 o bien podemos ejecutar R7. Para evitar que *F* o *G* permanezcan en la memoria activa como hipótesis, ejecutaremos R2 y R7 a la vez, para así obtener sus dos resultados, a pesar de que solamente necesitamos uno de ellos (porque R5 es un OR). Tras ejecutar R2 y R7 obtendremos *G* y *F* como datos reales en la memoria activa.

R1: IF (X And Y) THEN Z ◀
R2: IF (C And D) THEN G ◀
R3: IF (E And V) THEN H ◀
R4: IF (A And B) THEN C
R5: IF (F Or G) THEN X ◀
R6: IF (Z And B) THEN V ◀
R7: IF (E And C) THEN F ◀

Reglas ejecutadas		Reglas seleccionadas		Hipótesis	
R2 | R7	Ciclo 7	R5	Ciclo 4	X	Ciclo 3
R4	Ciclo 6	R1	Ciclo 3	Z	Ciclo 2
		R6	Ciclo 2	V	Ciclo 1
		R3	Ciclo 1	H	Ciclo 0

$M_7 = <A, B, D, Y, E, (H), (V), (Z), (X), \underline{F}, \underline{G}, C>$

Ciclo 8)

La siguiente hipótesis nos dice que debemos obtener *X*, para lo que necesitamos ejecutar la regla R5. De este modo obtendremos *X* como un dato en la memoria real.

R1: IF (X And Y) THEN Z ◀
R2: IF (C And D) THEN G
R3: IF (E And V) THEN H ◀
R4: IF (A And B) THEN C
R5: IF (F Or G) THEN X ◀
R6: IF (Z And B) THEN V ◀
R7: IF (E And C) THEN F

Reglas ejecutadas		Reglas seleccionadas		Hipótesis	
R5	Ciclo 8	R1	Ciclo 3	Z	Ciclo 2
R2 | R7	Ciclo 7	R6	Ciclo 2	V	Ciclo 1
R4	Ciclo 6	R3	Ciclo 1	H	Ciclo 0

$M_8 = <A, B, D, Y, E, (H), (V), (Z), \underline{X}, F, G, C>$

Ciclo 9)

La siguiente hipótesis nos dice que debemos obtener *Z*, para lo que necesitamos ejecutar la regla R1. De este modo obtendremos *Z* como un dato en la memoria real.

R1: IF (X And Y) THEN Z ◀
R2: IF (C And D) THEN G
R3: IF (E And V) THEN H ◀
R4: IF (A And B) THEN C
R5: IF (F Or G) THEN X
R6: IF (Z And B) THEN V ◀
R7: IF (E And C) THEN F

Reglas ejecutadas		Reglas seleccionadas		Hipótesis	
R1	Ciclo 9	R6	Ciclo 2	V	Ciclo 1
R5	Ciclo 8	R3	Ciclo 1	H	Ciclo 0
R2 | R7	Ciclo 7				
R4	Ciclo 6				

$M_9 = <A, B, D, Y, E, (H), (V), \underline{Z}, X, F, G, C>$

Ciclo 10)

La siguiente hipótesis nos dice que debemos obtener *V*, para lo que necesitamos ejecutar la regla R6. De este modo obtendremos *X* como un dato en la memoria real.

R1: IF (X And Y) THEN Z
R2: IF (C And D) THEN G
R3: IF (E And V) THEN H ◀
R4: IF (A And B) THEN C
R5: IF (F Or G) THEN X
R6: IF (Z And B) THEN V ◀
R7: IF (E And C) THEN F

Reglas ejecutadas		Reglas seleccionadas		Hipótesis	
R6	Ciclo 10	R3	Ciclo 1	H	Ciclo 0
R1	Ciclo 9				
R5	Ciclo 8				
R2 | R7	Ciclo 7				
R4	Ciclo 6				

$M_{10} = <A, B, D, Y, E, (H), \underline{V}, Z, X, F, G, C>$

Ciclo 11)

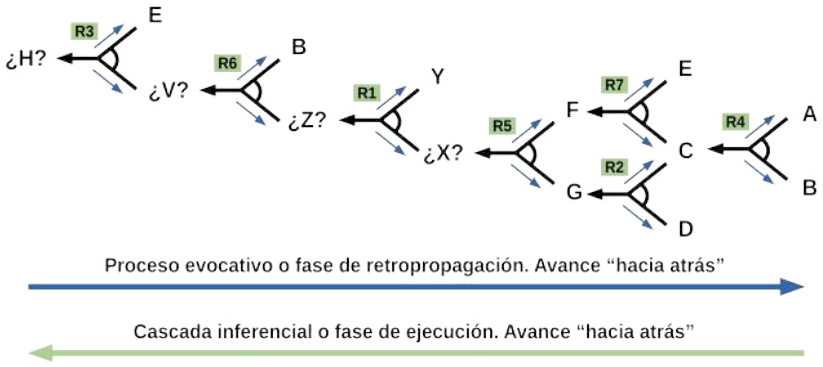
La siguiente hipótesis es la que nos planteamos al principio del problema y es la que nos dice que debemos obtener *H*, para lo que necesitamos ejecutar la regla R6. De este modo obtendremos *H* como un dato en la memoria real, finalizando la inferencia.

$R1:\ IF\ (X \text{ And } Y)\ THEN\ Z$
$R2:\ IF\ (C \text{ And } D)\ THEN\ G$
$R3:\ IF\ (E \text{ And } V)\ THEN\ H$ ◀
$R4:\ IF\ (A \text{ And } B)\ THEN\ C$
$R5:\ IF\ (F \text{ Or } G)\ THEN\ X$
$R6:\ IF\ (Z \text{ And } B)\ THEN\ V$
$R7:\ IF\ (E \text{ And } C)\ THEN\ F$

Reglas ejecutadas		Reglas seleccionadas	Hipótesis
R6	Ciclo 10		
R1	Ciclo 9		
R5	Ciclo 8		
R2 | R7	Ciclo 7		
R4	Ciclo 6		

$M_{11} = <A, B, D, Y, E, (H), \underline{V}, Z, X, F, G, C>$

El motor de inferencias ha necesitado 11 ciclos. El **circuito inferencia**l generado es el siguiente:



FASES DE DECISIÓN Y EJECUCIÓN

Como hemos visto, un ciclo está compuesto de una **fase de decisión** en la que se selecciona una de las reglas activadas, y una **fase de ejecución**, en la que se ejecuta y obtiene los resultados de ejecutar dicha regla. A la hora de ejecutar la fase de decisión deben tenerse en cuenta 3 factores:

- **Restricción**: debe simplificar el proceso de comparación entre reglas. Han de eliminarse del motor de inferencias aquellas reglas que no tienen nada que ver con el estado actual (representado por los datos que almacena la memoria activa en cada ciclo).
- **Equiparación**: se aplica al emparejamiento y trata de identificar qué reglas son relevantes dentro del contexto actual para poder aplicarlas. El resultado se denomina **conjunto conflicto** e incluye todas las reglas útiles para resolver el problema planteado.
- **Resolución de conflictos**: la decisión de que regla ejecutar está condicionada por la estrategia de búsqueda que se emplea para seleccionar las reglas que se activan.

TEMA 5

MODELOS DE RAZONAMIENTO

• **Modelos categóricos:**
  o Dominios de naturaleza marcadamente simbólica, con soluciones que pueden establecerse con total seguridad.

• **Modelos probabilísticos:**
  o Dominios de naturaleza estadística, soluciones no obtenibles de forma unívoca.

• **Modelos de razonamiento bajo incertidumbre:**
  o Dominios con incertidumbre, inherente a datos o a los propios mecanismos inferenciales.

• **Modelos de razonamiento basado en conjuntos difusos:**
  o Dominios en los que los elementos diferenciales incluyen matices lingüísticos.

Modelo Categórico :

• Formalmente:
  o X = {manifestaciones} = {x1, x2, …, xn}
  o Y = {interpretaciones} = {y1, y2, …, ym}

• Las relaciones causales entre manifestaciones e interpretaciones se formalizan a través de la función de conocimiento E.
  o E = E (X, Y)

• Problema lógico.
  o Dadas unas manifestaciones caracterizadas por una función -f-, encontrar la función -g- que satisface
    ▪ E: (f → g)
    ▪ E: (▲g → ▲f)
    ▪ E = E (x1, …, xn, y1, …, ym)

• Procedimiento sistemático para el modelo categórico:
  1. Identificación de M.
  2. Identificación de I.
  3. Construcción de E.
  4. Construcción del conjunto completo de complejos de manifestaciones.
  5. Construcción del conjunto completo de complejos de interpretaciones.
  6. Construcción del conjunto completo de complejos manifestación-interpretación.

m(1)	0	0	1	1
m(2)	0	1	0	1
	m1	m2	m3	m4

i(1)	0	0	1	1
i(2)	0	1	0	1
	i1	i2	i3	i4

M = { m1, m2, m3, m4 } **I** = { i1, i2, i3, i4 }

• Construcción del conjunto completo de complejos manifestación-interpretación:

o Base Lógica Expandida
- BLE = M x I = { m1i1, m1i2, m1i3, m1i4,
  m2i1, m2i2, m2i3, m2i4,
  m3i1, m3i2, m3i3, m3i4,
  m4i1, m4i2, m4i3, m4i4}

o La solución a cualquier problema está en BLE, pero hay muchas combinaciones absurdas.

o El papel del conocimiento -E- es eliminarlas y pasar a una base lógica reducida E: (BLE → BLR).

	m1	m2	m3	m4	m1	m2	m3	m4	m1	m2	m3	m4	m1	m2	m3	m4
m (1)	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
m (2)	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
i (1)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
i (2)	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	i1	i1	i1	i1	i2	i2	i2	i2	i3	i3	i3	i3	i4	i4	i4	i4

- BLR = {m1i1, m3i2, m2i3, m4i3, m3i4, m4i4}
- IF: (1) El conocimiento es completo
- And: (2) El dominio está bien descrito
- Then: (3) La solución a cualquier problema está en BLR.

Bayesiano:

- Las interpretaciones categóricas son poco frecuentes en el mundo real.
  - La existencia de una determinada causa no siempre conlleva la presencia de una manifestación.
  - Ante una manifestación dada, ¿podemos afirmar siempre y de forma categórica que existe una determinada causa?
- Probabilidad total y probabilidad condicional.
- La probabilidad condicional se parece a la total, pero puede ser definida como la probabilidad de las causas.
- En la probabilidad condicional aparecen involucrados dos sucesos, en donde la ocurrencia del segundo depende de la ocurrencia del primero.
- Bayes introduce una forma de razonamiento a posteriori.
  - Dadas dos posibles acciones A y B, y ante un caso tratado con una de tales acciones ¿Cuál es la probabilidad de que la acción haya sido A, si la respuesta del sistema ha sido E?

$$P(A/E)=\frac{P(E/A)P(A)}{P(E)}$$

- Obtención de la ecuación elemental del teorema de Bayes:
  - Sea una población sobre parte de cuyos elementos ha sido efectuada una acción A de un número de posibles acciones.
  - Todos los elementos de la población fueron tratados, con A o con cualquier otra acción, registrándose un cierto número de respuestas E.

- Por definición de probabilidad condicional:

$$P(E/A)=\frac{n(A\cap E)}{n(A)}=\frac{n(A\cap E)/N}{n(A)/N}=\frac{P(A\cap E)}{P(A)}\rightarrow P(A\cap E)=P(A)P(E/A)$$

- Análogamente:

$$P(A/E)=\frac{n(A\cap E)}{n(E)}=\frac{n(A\cap E)/N}{n(E)/N}=\frac{P(A\cap E)}{P(E)}\rightarrow P(A\cap E)=P(E)P(A/E)$$

$$P(A)P(E/A)=P(E)P(A/E)$$

• Obtención de una ecuación generalizada del teorema de Bayes.

- Sea una característica cualquiera X.
- Sea A el número de casos de una población estadística en ellos que X está presente.
- Sea ¬A el número de casos de la misma población estadística en los que X está ausente.
- Sea P una prueba potencialmente resolutiva para investigar la característica X.
- Sea ¬E el número de casos en los que la prueba da resultados positivos.

- Sea ¬E el número de casos en los que la prueba da resultados negativos.

	A	¬A	TOTAL
E	a	b	a + b
¬E	c	d	c + d
TOTAL	a + c	b + d	N

- a = positivos reales
- b = falsos positivos
- c = falsos negativos
- d = negativos reales

- Relaciones
  - P (E / A) = a / (a + c) *sensibilidad*
  - P (¬ E / A) = c / (a + c)
  - P (A / E) = a / (a + b)
  - P (¬ A / E) = b / (a + b)
  - P (E / ¬ A) = b / (b + d)
  - P (¬ E / ¬ A) = d / (b + d) *especificidad*
  - P (A / ¬ E) = c / (c + d)
  - P (¬ A / ¬ E) = d / (c + d)

- Prevalencias
  - P (E) = (a + b) / N P (A) = (a + c) / N
  - P (¬ E) = (c + d) / N P (¬ A) = (b + d) / N

• Tras conocer las probabilidades a priori ¿cómo se plantea la probabilidad condicional a posteriori de la característica X dado un resultado positivo de la prueba?

$$Bayes \rightarrow P(A/E) = \frac{P(E/A)P(A)}{P(E)} : P(E) = \frac{(a+b)}{N}$$
$$a = (a+c)P(E/A) : b = (b+d)P(E/\neg A)$$
$$(a+c) = N \times P(A) : (b+d) = N \times P(\neg A)$$
$$a = N \times P(A) \times P(E/A) : b = N \times P(\neg A) \times P(E/\neg A)$$
$$P(E) = P(A)P(E/A) + P(\neg A)P(E/\neg A)$$
$$P(A/E) = \frac{P(E/A)P(A)}{P(E/A)P(A) + P(E/\neg A)P(\neg A)}$$

- La ecuación $P(A/E) = \frac{P(E/A)P(A)}{P(E/A)P(A) + P(E/\neg A)P(\neg A)}$

  es directamente generalizable. Así, si consideramos todos los Ai posibles obtenemos que:

$$P(A0/E) = \frac{P(E/A0)P(A0)}{\sum_i P(E/Ai)P(Ai)}$$

Que es una expresión general del teorema de Bayes

Tema 6:

SISTEMAS CONEXIONISTAS ALIMENTADOS HACIA ADELANTE

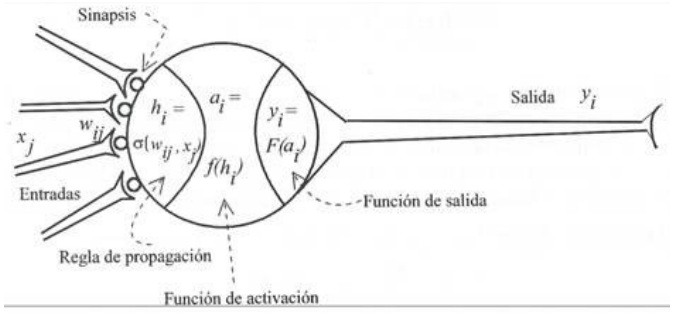
Fundamentos de Intelixencia Artificial

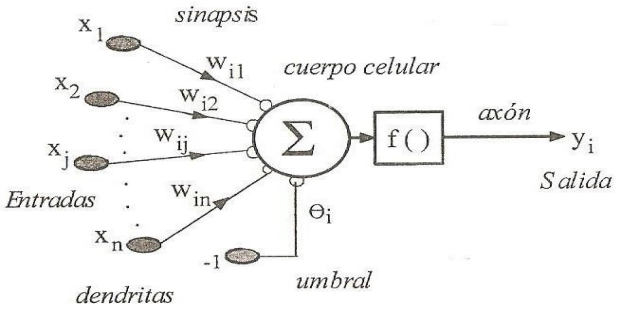
Vicente Moret, Bertha Guijarro, Eduardo Mosqueira, Mariano Cabrero, Amparo Alonso

NEURONAS Y REDES NEURONALES

NEURONAS ARTIFICIALES

Una **neurona biológica** consta de varias dendritas por las cuales recibe impulsos eléctricos de otras neuronas (las sinapsis). Con los valores que recibe por las entradas, los procesa en función de qué neurona haya enviado dicha señal y emite una respuesta a través de su axón hacia otras neuronas.





Una **neurona artificial *i*** tiene la misma estructura: consta de un conjunto de ***n* entradas** x_j , a través de las cuales recibe los datos, y la **bías** b , una entrada adicional con valor siempre 1 que indica la importancia de la neurona. Cada entrada *j* de la neurona *i* tiene un un **peso** w_{ij} asignado que permite modular la importancia de dicha entrada, a excepción de la bías, que tiene su propio peso θ_i . Cada neurona dispone de una **regla de propagación** h_i encargada de realizar la suma ponderada (peso de cada entrada multiplicado por su valor de entrada) de todas las entradas y sumar la bías multiplicada por su peso. A ese resultado se le aplica una **función de activación o transferencia** y_i para modificar el valor de salida de la neurona *i* según nos interese. De este modo:

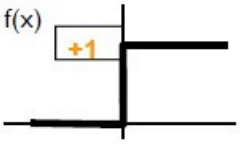
$$h_i = \sum_{j=1}^{n} (w_{ij} \cdot x_j) + b \cdot \theta_i \qquad y_i = f(h_i) = f\left(\sum_{j=1}^{n} (w_{ij} \cdot x_j) + \theta_i \cdot b\right)$$

La neurona artificial también se conoce como **Elemento de Procesado (EP)** y es la unidad básica de una red neuronal. Los valores de entrada y salida son números reales, es decir, $x_j, y_i \in \Re$, pero que suelen estar dentro de un intervalo, generalmente $[-1,1]$ o $[0,1]$. Las conexiones se encargan de conectar los EP y son las que realmente codifican el conocimiento de la red neuronal mediante el peso que tenga asignado cada conexión: si $w_{ij}>0$ significa que dicha entrada *j* adquiere importancia, si $w_{ij}=0$ significa que dicha entrada *j* no tiene importancia, y si $w_{ij}<0$ significa que dicha entrada *j* pierde importancia. La bías de cada neurona indica cómo de dispuesta está la neurona a activarse, es decir, si la neurona es importante o no; pueden existir neuronas que no tengan bías.

EJEMPLOS DE PUERTAS LÓGICAS

Supongamos que queremos construir algo tan sencillo como una puerta lógica empleando una neurona. El valor de sus entradas será 0 (`FALSE`) o 1 (`TRUE`), es decir: $x_j=0 \vee x_j=1$. Como el resultado también es `TRUE` o `FALSE`, la función de activación solo podrá devolver 0 o 1, de modo que usaremos la **función de activación escalón**:

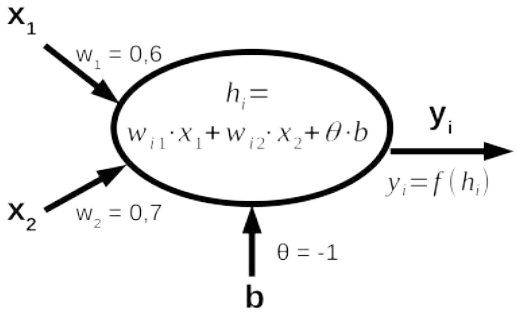
$$f(h_i) = \begin{cases} 1 & si \quad h_i > 0 \\ 0 & si \quad h_i \leq 0 \end{cases}$$



Recordemos que h_i era la suma ponderada de todas las entradas más la suma ponderada de la bías. Para cada puerta lógica buscaremos aquellos pesos que devuelvan el resultado correcto a la salida.

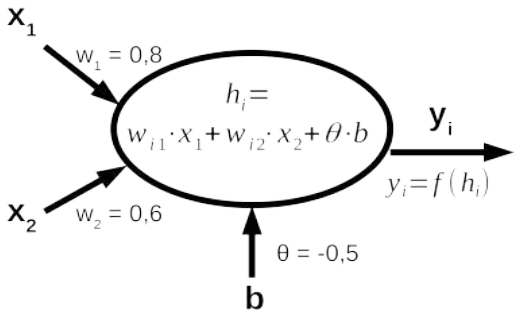
- **Puerta lógica `AND`**: devuelve 1 solo cuando sus dos entradas valen 1.

x_1	x_2	y_i
0	**0**	$f(0{,}6\cdot\mathbf{0} + 0{,}7\cdot\mathbf{0} + (-1)\cdot 1) = f(-1) = 0$
0	**1**	$f(0{,}6\cdot\mathbf{0} + 0{,}7\cdot\mathbf{1} + (-1)\cdot 1) = f(-0{,}3) = 0$
1	**0**	$f(0{,}6\cdot\mathbf{1} + 0{,}7\cdot\mathbf{0} + (-1)\cdot 1) = f(-0{,}4) = 0$
1	**1**	$f(0{,}6\cdot\mathbf{1} + 0{,}7\cdot\mathbf{1} + (-1)\cdot 1) = f(0{,}3) = 1$



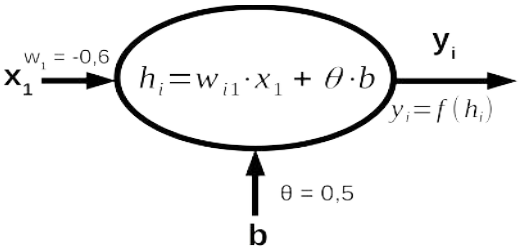
- **Puerta lógica `OR`**: devuelve 1 cuando alguna (o ambas) de sus dos entradas valgan 1.

x_1	x_2	y_i
0	**0**	$f(0{,}8\cdot\mathbf{0} + 0{,}6\cdot\mathbf{0} + (-0{,}5)\cdot 1) = f(-0{,}5) = 0$
0	**1**	$f(0{,}8\cdot\mathbf{0} + 0{,}6\cdot\mathbf{1} + (-0{,}5)\cdot 1) = f(0{,}1) = 1$
1	**0**	$f(0{,}8\cdot\mathbf{1} + 0{,}6\cdot\mathbf{0} + (-0{,}5)\cdot 1) = f(0{,}3) = 1$
1	**1**	$f(0{,}8\cdot\mathbf{1} + 0{,}6\cdot\mathbf{1} + (-0{,}5)\cdot 1) = f(0{,}9) = 1$

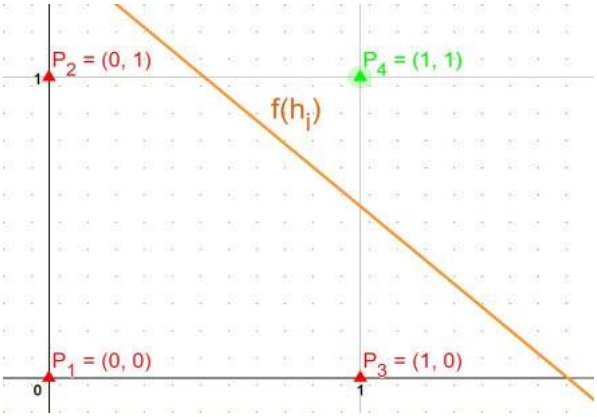


- **Puerta lógica `NOT`**: devuelve el valor opuesto al que recibe en su entrada.

x_1	y_i
0	$f((-0{,}6)\cdot\mathbf{0} + 0{,}5\cdot 1) = f(0{,}5) = 1$
1	$f((-0{,}6)\cdot\mathbf{1} + 0{,}5\cdot 1) = f(-0{,}1) = 0$



Se denomina **patrón** a cada combinación posible de los valores de entrada $(x_1, \ldots, x_n)$ y una forma fácil de visualizar cómo obtiene su salida una neurona es representar dichos patrones en una gráfica. Fijémonos que la función de activación se parece a la ecuación de una recta. Para el ejemplo de la puerta `AND` (gráfica derecha), todo patrón cuya función de activación sea mayor que la recta devolverá un 1 en la salida, y 0 en caso contrario.



Como observamos, cambiando los pesos obtenemos distintos resultados, incluso con entradas iguales. Hay que resaltar que **neuronas iguales con pesos distintos generan resultados distintos**.

La pregunta importante es: ¿cómo podemos hacer que una neurona emita la salida que nos interese? La neurona debe pasar por un proceso de **entrenamiento** en el cual aprenderá qué salida debe emitir a partir de un conjunto de patrones de entrada que le demos. Como vimos, la forma más efectiva es modificar los pesos de cada entrada y la bías. Para ello debemos pasarle una serie de patrones de entrada (**conjunto de entrenamiento**) y decirle qué salida debe generar cada patrón, y a partir de ahí la propia neurona modificará sus pesos para emitir las salidas adecuadas, lo cual se conoce como Adaline.

ADALINE

Adaline procede de ***ADAptative LINear Element*** y es el modelo más básico de red neuronal artificial, creado en 1960. Se trata de una única neurona que emplea una función de activación lineal, $f(h_i) = m \cdot h_i + n$. Para modificar los pesos de sus entradas se emplea un algoritmo **aprendizaje supervisado con corrección de error**: aprendizaje porque es capaz de modificar por sí sola los valores de los pesos; es supervisado porque necesita que un "supervisor" le diga qué error está cometiendo en cada salida generada (previamente debe conocerse la salida correcta y luego calcular el error); corrección de error porque es capaz de lograr minimizar el error que comete con cada patrón en cada iteración.

CORRECCIÓN DEL ERROR

Sea el conjunto de entrenamiento (X, D) formado por:

- $\boldsymbol{X}$: conjunto de L vectores de dimensión n, donde cada vector es un patrón.
- $\boldsymbol{D}$: conjunto de L vectores de dimensión 1, donde cada vector guarda la salida deseada.

Para cada patrón k , la red neuronal emitirá una salida y_k de la forma $y_k = \sum_{j=1}^{n} w_j \cdot x_j$, donde el **error** $\boldsymbol{E_k}$ cometido será la diferencia entre el valor deseado d_k y el valor obtenido y_k : $\boldsymbol{E_k = d_k - y_k}$

■ Error Cuadrático Medio (ECM, *Least Mean Square* o *LMS*)

El ECM para un patrón k se corresponde con la siguiente fórmula:

$$\boldsymbol{E_k = \frac{(d_k - y_k)^2}{2}}$$

El ECM para todos los patrones es la suma del ECM de cada patrón:

$$\boldsymbol{E = \sum_{k=1}^{L} E_k = \frac{(d_k - y_k)^2}{2}}$$

■ Descenso del gradiente o Regla Delta

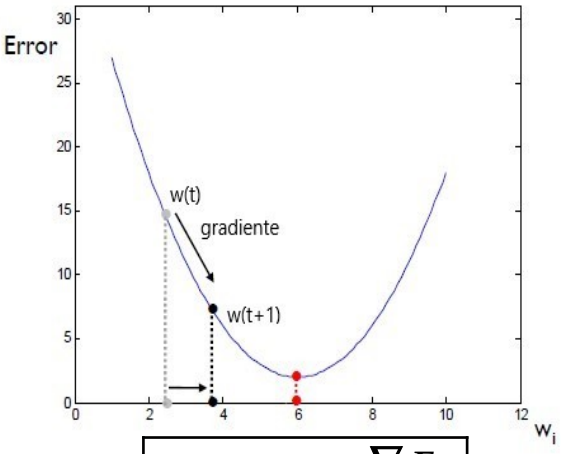
Para minimizar el error cometido según el conjunto total de pesos de todas las entradas (W), debemos derivar respecto a dicho conjunto para conocer cuánto cambia el error. Este algoritmo se basa en el conocimiento local que cometemos y podemos medir con el ECM: no conocemos qué forma tiene la gráfica que representa al error exactamente, pero si podemos conocer qué error se comete en cada punto gracias a su derivada, lo cual nos ayuda a guiarnos hacia un mínimo de dicha función (donde el error es mínimo). Empleamos el concepto de **gradiente** (∇) , que consiste en un vector que indica en qué dirección varía más rápido una función en cada punto (en 2 dimensiones equivale al vector director de la recta tangente a la función en un punto y que apunta al sentido en el que la función crece o decrece).

Un ejemplo sencillo de entender sería el caso de que estuviésemos en una montaña con los ojos vendados y no conociésemos la forma de la montaña. Sin embargo, con nuestros pies podemos dar pequeños pasos y conocer si subimos o bajamos dicha montaña con cada paso. Buscamos el mínimo error posible, por lo que intentaremos **dar pasos en la dirección en la que sintamos que descendemos**. Dicha dirección nos la indica el gradiente. Si en un paso notamos que descendemos muy rápido significa que estamos lejos del mínimo, por lo que debemos dar pasos más grandes para acercarnos más rápido, pero puede pasar que diésemos pasos demasiado grandes y nos quedemos en la misma situación en la que estábamos (o peor) pero en otra zona distinta de la montaña: debemos **adaptar el tamaño de cada paso**. El descenso del gradiente sigue el mismo procedimiento: para la iteración $t+1$, el valor cada peso, $w_j(t+1)$, será su valor actual $w_j(t)$ más la suma de la variación $\Delta w_j(t)$ que nos indique su gradiente.

$$\boxed{\boldsymbol{w_j(t+1) = w_j(t) + \Delta w_j(t)}}$$

El **algoritmo del descenso del gradiente** es el siguiente:

1. Inicializar el conjunto de pesos W aleatoriamente y calcular cuánto error cometen esos valores.
2. Calcular el gradiente descendiente de mayor tamaño (el que apunta al valor mínimo). Para saber la dirección de avance obtenemos la pendiente (derivada) entre el punto anterior y el actual:
   - **2.1.** Si **pendiente es positiva** significa que nos hemos alejado del mínimo (estaríamos subiendo la montaña), por lo que debemos avanzar en sentido contrario (debemos descender).
   - **2.2.** Si **pendiente es negativa** significa que nos acercamos al mínimo (bajamos la montaña).
3. Modificar los pesos para obtener un nuevo valor. Equivale a realizar un salto en la gráfica.
4. Repetir el proceso hasta que cometamos el mínimo error (llegar al mínimo de la gráfica).



La idea principal es cambiar el valor de cada peso $w_j \in W$ de forma proporcional a la derivada del error, medida para el patrón actual k . Es decir, hay que realizar un salto proporcional (μ) al valor del gradiente (la derivada) y cambiar de sentido si es necesario (cuando la pendiente es positiva; cuando la pendiente es negativa, menos por menos es positivo, por lo que se avanza en la misma dirección que indica el gradiente):

$$\boldsymbol{\Delta_k w_j = -\mu \cdot \frac{\nabla E_k}{\nabla w_j}}$$

El aprendizaje del Adaline se basa en aplicar el descenso del gradiente para disminuir el error que comete en cada iteración. La demostración de es simple y solo necesita recordar aplicar la regla de la cadena al derivar.

$$\frac{\nabla E_k}{\nabla w_j} = \frac{\partial\left(\frac{(d_k - y_k)^2}{2}\right)}{\partial w_j} = \frac{1}{2}\cdot\frac{\partial((d_k - y_k)^2)}{\partial w_j} = \frac{1}{2}\cdot 2\cdot(d_k - y_k)\cdot\frac{\partial(d_k - y_k)}{\partial w_j} = \frac{\partial(-y_k)}{\partial w_j}\cdot(d_k - y_k) =$$

$$-\frac{\partial\left(\sum_i^n w_i \cdot x_{ik}\right)}{\partial w_j}\cdot(d_k - y_k) = -\frac{\partial(w_j \cdot x_{jk})}{\partial w_j}\cdot(d_k - y_k) \rightarrow \boxed{\boldsymbol{\frac{\nabla E_k}{\nabla w_j} = -x_{jk}\cdot(d_k - y_k)}}$$

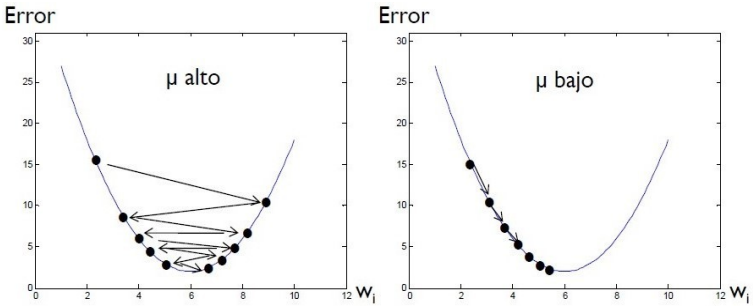
De aquí deducimos que si un peso w_j se mueve en la dirección en la que crece $(y_k - d_k)\cdot x_{jk}$, la cual obtenemos del gradiente $\frac{\nabla E_k}{\nabla w_j}$, entonces el error se incrementa, porque sería equivalente a subir la montaña cuando buscamos el mínimo, y por lo tanto hay que moverse en la dirección contraria, por eso el signo negativo. Si combinamos las dos ecuaciones obtenidas anteriormente obtenemos que la variación que hay que realizar a cada uno de los pesos es la siguiente:

$$\left.\begin{aligned} \nabla_k w_j &= -\mu\cdot\frac{\nabla E_k}{\nabla w_j} \\ \frac{\nabla E_k}{\nabla w_j} &= -x_{jk}\cdot(d_k - y_k) \end{aligned}\right\} \rightarrow \boxed{\boldsymbol{\nabla_k w_j = \mu\cdot x_{jk}\cdot(d_k - y_k)}}$$

$$\boxed{\boldsymbol{w_j(t+1) = w_j(t) + \mu\cdot x_{jk}\cdot(d_k - y_k)}}$$

APRENDIZAJE

El término μ representa la **tasa de aprendizaje**, que es la que controla cuánto de grande debe ser la variación del peso para que converja en el mínimo de la función de error. En el ejemplo de la escalada sería la distancia que debemos andar con cada paso cuando nos movemos por la montaña. El Adaline aprenderá a obtener mejores resultados cuando sea capaz de adaptar correctamente su tasa de aprendizaje en cada iteración que ejecute, dado que esto le ayudará a acercarse cada vez más al resultado adecuado, cometiendo el menor error. El algoritmo de aprendizaje se explica a continuación.



1. Inicializar los pesos de las entradas aleatoriamente.
2. Aplicarle un patrón de entrada a la red cuyo resultado sea conocido de antemano.
3. Calcular la salida generada por dicho patrón.
4. Calcular el error cometido a la salida por dicho patrón.
5. Aplicar el algoritmo de descenso del gradiente para corregir los pesos de las entradas.
6. Repetir los pasos del 2 al 5 para todos los patrones del entrenamiento.
7. Si el ECM toma un valor aceptable, la red se considera entrenada. Sino, volver al paso 2.

PERCEPTRÓN

El Perceptrón es un modelo de red neuronal artificial previo al Adaline (creado en 1958). Su principal ventaja está en que es capaz de aprender a clasificar patrones concretos. Su estructura es igual que Adaline: una capa de entrada y un único elemento en la capa de salida, pero sus diferencias principales están en la función de transferencia y en el algoritmo de entrenamiento que emplea.

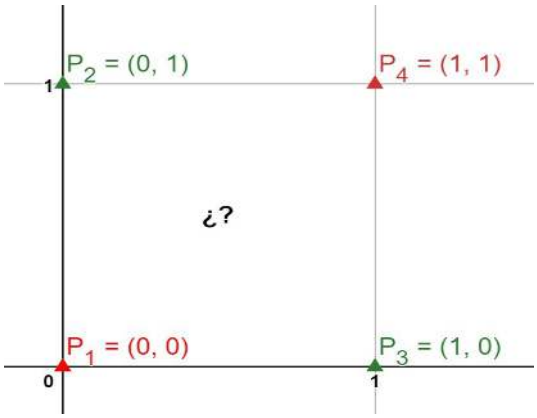
SEPARABILIDAD LINEAL

El Perceptrón tiene una capacidad de representación bastante limitada a causa de que solo hay un único elemento de procesado en su capa de salida. Esto solo le permite realizar una separación lineal de los elementos a la hora de clasificarlos, es decir, separar los elementos mediante ecuaciones lineales. Un conjunto de entradas es **linealmente separable** cuando su representación gráfica puede separarse mediante una única función lineal (recta, plano, hiperplano, etc).

Un ejemplo de separabilidad lineal lo habíamos visto previamente con la puerta `AND` donde la recta $f(h_i)$ separaba los patrones en `TRUE` (por encima) o `FALSE` (por debajo). Si cada patrón del conjunto de entrada tiene solo 2 elementos (bidimensional), la separabilidad lineal se realiza mediante una recta; si el conjunto es tridimensional, se realizaría mediante un plano, y así sucesivamente. De este modo, para una **entrada de *n* dimensiones, la separabilidad lineal emplea un hiperplano de *n-1* dimensiones**.

■ **Problema de la puerta XOR**

Hay gran variedad de problemas que no son linealmente separables, el caso más simple es la puerta lógica XOR: devuelve 1 sólo si todas las entradas son distintas. Si representamos gráficamente sus resultados para entradas bidimensionales (como hicimos con la puerta AND) veríamos que no hay ninguna figura lineal **única** que separe los resultados del mismo tipo en una única región:

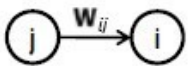


PERCEPTRÓN MULTICAPA

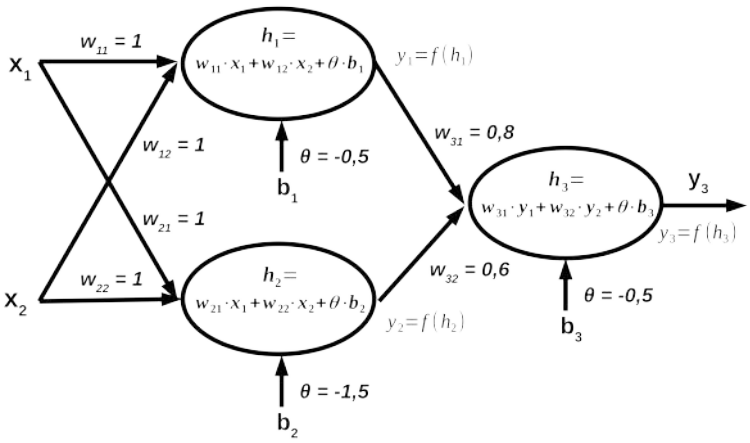
La solución para los problemas no separables linealmente está en añadir más capas al Perceptrón, pasando a tener un **Perceptrón Multicapa (MLP)**. La forma más sencilla de obtenerlo es conectar más neuronas a la salida de otra neurona, de modo que se organizan en capas (verticalmente).

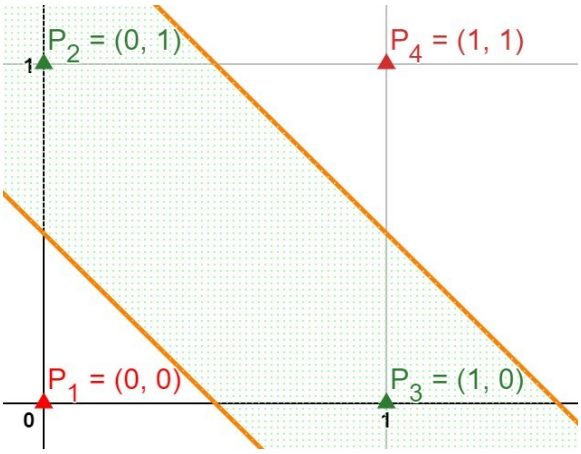
Para resolver el problema de la puerta XOR necesitamos separar sus salidas en 3 regiones: dos regiones para los resultados FALSE y una para los resultados TRUE, de modo que necesitamos 2 rectas, que podemos obtener empleando 2 neuronas. Además, para poder representar ambas rectas dentro del mismo plano añadimos una neurona a la capa de salida que combine sus resultados; las entradas de esta neurona son las salidas de las otras 2 neuronas.

Antes de continuar necesitamos definir la **notación** para distinguir las entradas y los pesos de cada neurona, dado que ahora hay más de una neurona y tenemos más capas. Se denota como w_{ij} al peso que conecta la salida de la neurona *j* con la neurona *i*.



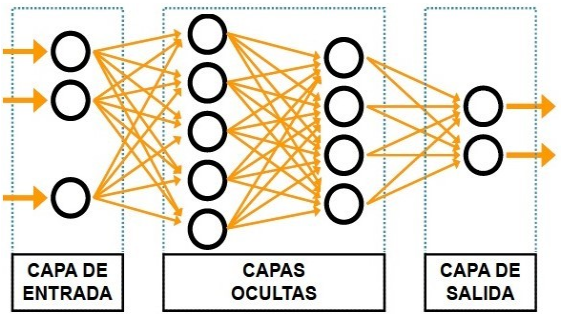
x_1	x_2	h_1	y_1	h_2	y_2	h_3	y_3
0	**0**	$1\cdot\mathbf{0}+1\cdot\mathbf{0}+(-0{,}5)\cdot 1=-0{,}5$	0	$1\cdot\mathbf{0}+1\cdot\mathbf{0}+(-1{,}5)\cdot 1=-0{,}5$	0	$1\cdot\mathbf{0}+(-1{,}5)\cdot\mathbf{0}+(-0{,}5)\cdot 1=-0{,}5$	**0**
0	**1**	$1\cdot\mathbf{0}+1\cdot\mathbf{1}+(-0{,}5)\cdot 1=0{,}5$	1	$1\cdot\mathbf{0}+1\cdot\mathbf{1}+(-1{,}5)\cdot 1=-0{,}5$	0	$1\cdot\mathbf{1}+(-1{,}5)\cdot\mathbf{0}+(-0{,}5)\cdot 1=0{,}5$	**1**
1	**0**	$1\cdot\mathbf{1}+1\cdot\mathbf{0}+(-0{,}5)\cdot 1=0{,}5$	1	$1\cdot\mathbf{1}+1\cdot\mathbf{0}+(-1{,}5)\cdot 1=-1{,}5$	0	$1\cdot\mathbf{1}+(-1{,}5)\cdot\mathbf{0}+(-0{,}5)\cdot 1=0{,}5$	**1**
1	**1**	$1\cdot\mathbf{1}+1\cdot\mathbf{1}+(-0{,}5)\cdot 1=1{,}5$	1	$1\cdot\mathbf{1}+1\cdot\mathbf{1}+(-1{,}5)\cdot 1=0{,}5$	1	$1\cdot\mathbf{1}+(-1{,}5)\cdot\mathbf{1}+(-0{,}5)\cdot 1=-1$	**0**





ARQUITECTURA GENERAL

Como ya dijimos, un Perceptrón distribuye todas sus neuronas en capas, cada una conectada con la siguiente:



- **Capa de entrada**: se encargan de recibir los datos de entrada (patrones) y transmitírselos al resto de neuronas, de modo que no realizan ningún cálculo (de hecho, no tienen bías). Generalmente hay una neurona por cada entrada.
- **Capa de salida**: son las neuronas que generan la salida de la red neuronal. Por lo general hay una neurona por cada salida deseada, y estas neuronas tampoco tienen bías.
- **Capas ocultas**: son las neuronas que realizan los cálculos tal y como vimos hasta ahora. La única forma de conocer cuántas capas ocultas (columnas) hay es mediante ensayo y error; asimismo tampoco sabemos cuántas neuronas debe tener cada capa oculta.

La característica principal del Perceptrón está en el **conocimiento distribuido**: la información no se almacena en una sola neurona, sino que depende de los pesos de las conexiones y las bías de cada neurona. Esto nos aporta una **gran tolerancia a fallos**, dado que si al implementar la red neuronal (en hardware o software) una, o más, de esas neuronas en la capa oculta fallase, se podría reajustar el peso y la bías del resto de neuronas para que la red neuronal siguiese funcionando correctamente. Sin embargo, resulta **imposible explicar el funcionamiento de la red neuronal** dado que necesitaría analizar todos los pesos y bías de todas las neuronas para cada patrón de entrada posible y así conocer cómo la red es capaz de entrenarse, lo cual resulta extremadamente complejo a medida que aumenta el número de neuronas.

El MLP puede aplicarse para resolver los siguientes problemas:

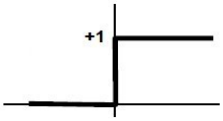
- **Problemas de ajuste de curvas**: buscan obtener una función matemática desconocida. Los patrones de entrada serían pares donde el primer elemento será la entrada y el segundo sería el valor de dicha función. Por ejemplo, el patrón k sería: $((x_{k1}, x_{k2}, \ldots, x_{kn}; y_{k1}, y_{k2}, \ldots, y_{km}), y_{ki}\ valor\ real)$.
- **Problemas de clasificación**: buscan agrupar los patrones en clases en función de sus características. Los patrones de entrada son pares donde el primer elemento es un valor y el segundo par será la clase a la que pertenece dicho patrón: $((x_{k1}, x_{k2}, \ldots, x_{kn}; y_k), y_k\, clase)$
- **Problemas de regresión**: buscan ver si un determinado patrón tiene alguna relación con el resto de patrones del conjunto de entrenamiento. Los patrones de entrada son pares donde el primer elemento es la entrada y el segundo elemento es su valor:. $((x_{k1}, x_{k2}, \ldots, x_{kn}; y_{k1}, y_{k2}, \ldots, y_{km}), y_{ki}\ valor\ real)$

FUNCIONES DE ACTIVACIÓN

Como ya hemos visto, la salida de cada neurona se calcula mediante una función de activación (o transferencia). Dicha función es la que se aplica al resultado de calcular la combinación lineal de sus entradas h_i (multiplicar entradas por sus pesos y sumar las bías). Por lo general, las neuronas dentro de una misma capa usan la misma función de activación. Esta función no suele ser una función lineal, porque si encadenamos varias funciones lineales (una capa propaga su resultado a la siguiente, y así en todas las capas) el resultado también sería una función lineal, e incluso podría llegar a ser nula para determinadas combinaciones de pesos de las neuronas (igual el *kernel* de un espacio vectorial en Álgebra). Es importante destacar que las funciones de activación deben ser continuas en todo su dominio, sino pueden existir valores que no generen ninguna salida. Generalmente se suelen emplear las siguientes funciones:

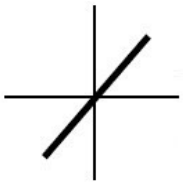
- **Función escalón**: se usa en problemas de clasificación porque solo aporta dos salidas distintas: 1 si el patrón pertenece a una clase o 0 (o -1) si el patrón no pertenece a una clase.

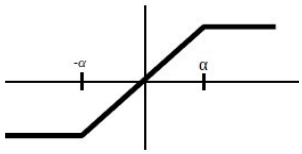
$$f(x)=\begin{cases}1 & si\ \ x>0\\ 0 & si\ \ x\le 0\end{cases}$$



- **Función lineal**: se emplea en problemas de regresión porque tiene forma de recta. También puede obtenerse la **función mixta** al combinar la función escalón con la función lineal.

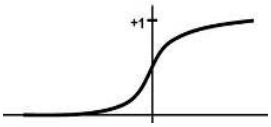
$$f(x)=m\cdot x \qquad f(x)=\begin{cases}-1 & si \quad x\geq\alpha \\ m\cdot x & si \quad -\alpha<x<\alpha \\ 1 & si \quad x\leq-\alpha\end{cases}$$





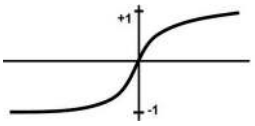
- **Función sigmoidal**: se usa en problemas de clasificación, pero puede extenderse a cualquier tipo de problema. Su salida no es lineal, sino que es una curva acotada en el intervalo (0, 1)

$$f(x)=\frac{1}{1+e^{-x}}$$



- **Función sigmoidal hiperbólica**: es igual que la función sigmoidal pero en el intervalo (-1, 1).

$$f(x)=\frac{2}{1+e^{-2x}}-1$$



REDES NEURONALES

ALGORITMO DE RETROPROPAGACIÓN (*BACKPROPAGATION*)

A la hora de entrenar el MLP no podemos emplear el descenso del gradiente porque ese algoritmo está pensado para redes neuronales que no tienen capas ocultas (y porque absolutamente todas las neuronas tienen pesos de entrada y bías, y aquí solo tienen las neuronas de las capas ocultas). Podemos seguir con el razonamiento de calcular la salida (capa de salida) y medir el error respecto al valor esperado, pero no sabremos exactamente qué neuronas cometen el error, ni en que capa están.

El **algoritmo de *backpropagation* es supervisado con corrección de error** y se diseñó para generalizar el descenso del gradiente a todas las neuronas de todas las capas ocultas, siempre y cuando sus **funciones de activación sean no lineales y derivables** (después veremos por qué). Sin embargo, este algoritmo tiene la capacidad de generalizarse para ajustarse a cualquier tipo de problema. Dado que ahora tenemos muchas más neuronas que antes y divididas en capas, necesitamos volver a definir la notación que emplearemos, actualizando también las fórmulas que empleábamos:

- n : nº de entradas de cada patrón.
- h : capa oculta.
- o : capa de salida.
- p : patrón aplicado a la entrada.
- j : neurona dentro de la capa h .
- k : neurona dentro de la capa de salida o .
- w_{ji}^{h} : peso de la conexión de la neurona i (capa $h-1$) con la neurona j (capa h).
- i_{pj}^{h} : salida de la neurona j (capa h) para el patrón p
- o_{pj} : salida de la neurona j para el patrón p .

◆ **Regla de propagación** σ_{pj} : es la suma ponderada de las entradas más la bías de la neurona j para el patrón de entrada p . Se aplica tanto en una capa oculta h como en la de salida o .

◆ **Función de activación** i_{pj}^{h} **:** es el resultado de aplicar la función de activación f al resultado de la regla de propagación a la neurona j en la capa oculta h para el patrón de entrada p . Solo se aplica a neuronas dentro de capas ocultas.

◆ **Función de activación de salida** o_{pj} **:** es la salida que genera la red neuronal y es el resultado de aplicar una función de activación f_k^o al resultado de la regla de propagación a la neurona j en la capa de salida o para el patrón de entrada p . Solo se aplica a neuronas en la capa de salida.

$$\sigma_{pj}^{h} = \sum_{i=0}^{n} w_{ji}^{h} \cdot x_{pi} \qquad i_{pj}^{h} = f_{j}^{h}(\sigma_{ph}^{h}) \qquad o_{pk} = f_{k}^{o}(\sigma_{pk}^{o})$$

Cabe destacar que ahora la red neuronal es más compleja que antes, así que es muy común que en la capa de salida haya más de una neurona, así que no basta con calcular un único valor para el error cometido por la red. En cambio, para calcular el error debemos obtener el error cuadrático medio de todas las neuronas de la capa de salida; es decir, calcular la suma de los cuadrados de los errores cometidos por todas las neuronas de la capa de salida. Denotaremos como $\boldsymbol{\delta_{pk}^{o}}$ al ECM de una neurona k en la capa de salida o para un patrón p , y como $\boldsymbol{E_p}$ al ECM de todas las neuronas de la capa de salida:

$$\delta_{pk} = (y_{pk} - o_{pk}) \qquad E_p = \frac{\sum_{k=1}^{n} (\delta_{pk})^2}{2} = \frac{\sum_{k=1}^{n} (y_{pk} - o_{pk})^2}{2}$$

Igual que hacíamos antes, lo que queremos minimizar es el error, por lo que debemos derivar esta función. A continuación mostraremos el resultado para una neurona de la capa de salida, de modo que la derivada del error $\boldsymbol{E_p}$ en la capa de salida será la suma de las derivadas de todos los errores cometidos por las *n* neuronas de la capa de salida (obligando a que las funciones de activación sean derivables):

$$\nabla E_p = \sum_{k=1}^{n} \frac{\partial \delta_{pk}}{\partial w_{jk}^{o}} = \frac{\partial \left(\frac{(y_{pk} - o_{pk})^2}{2} \right)}{\partial w_{jk}^{o}} = \frac{1}{2} \cdot 2 \cdot (y_{pk} - o_{pk}) \cdot \frac{\partial (y_{pk} - o_{pk})}{\partial w_{jk}^{o}} =$$

$$(y_{pk} - o_{pk}) \cdot \underbrace{\frac{\partial (y_{pk} - f_k^o(\sigma_{pk}^o))}{\partial w_{jk}^{o}} \cdot \frac{\partial (\sigma_{pk}^{o})}{\partial \sigma_{pk}^{o}}}_{\text{regla de la cadena}} \rightarrow \boxed{\nabla \boldsymbol{E_p} = (\boldsymbol{y_{pk} - o_{pk}}) \cdot \underbrace{\frac{\partial (-f_k^o(\sigma_{pk}^o))}{\partial \sigma_{pk}^{o}} \cdot \frac{\partial (\sigma_{pk}^{o})}{\partial w_{jk}^{o}}}_{\textbf{intercambiar diferenciales}}}$$

Se han intercambiado los diferenciales en las derivadas porque el último término nos indica que tenemos que calcular la derivada de la regla de propagación de la neurona respecto al peso, es decir: calcular la derivada de la suma ponderada de las entradas por sus pesos respecto a sus pesos, lo cual se calcula fácil (es como derivar el producto de una variable por una constante: obtenemos la constante). Si la última capa es o , la penúltima capa será la $o-1$; de este modo, las entradas de las neuronas de la última capa se corresponden con las salidas de las neuronas de la penúltima capa, obteniendo $x_{pk} = i_{pj}^{o-1}$:

$$\frac{\partial \sigma_{pk}^{o}}{\partial w_{jk}^{o}} = \frac{\partial \left(\sum_{j} w_{jk}^{o} \cdot i_{pk}^{o-1} \right)}{\partial w_{jk}^{o}} \rightarrow \boxed{\frac{\partial \boldsymbol{\sigma_{pk}^{o}}}{\partial \boldsymbol{w_{jk}^{o}}} = \boldsymbol{i_{pk}^{o-1}}}$$

$$\frac{\partial \delta_{pk}}{\partial w_{jk}^{o}} = -(y_{pk} - o_{pk}) \frac{\cdot \partial (f_k^o(\sigma_{pk}^o))}{\partial \sigma_{pk}^{o}} \cdot \frac{\partial (\sigma_{pk}^{o})}{\partial w_{jk}^{o}} = -(y_{pk} - o_{pk}) \frac{\cdot \partial (f_k^o(\sigma_{pk}^o))}{\partial \sigma_{pk}^{o}} \cdot i_{pk}^{o-1} \rightarrow$$

$$\boxed{\nabla \boldsymbol{E_p} = (\boldsymbol{y_{pk} - o_{pk}}) \cdot \boldsymbol{i_{pk}^{o-1}} \cdot \frac{\partial (f_k^o(\sigma_{pk}^o))}{\partial \sigma_{pk}^{o}}}$$

La ecuación obtenida es la que permite que el algoritmo de *backpropagation* funcione. Si nos fijamos, tiene una forma parecida a la del descenso del gradiente: para una neurona den la capa de salida, el error depende de la diferencia entre el valor deseado y el valor obtenido multiplicado por la entrada recibida, ¡pero multiplicado por la derivada de la función de activación! Esta derivada es la que indica a cada neurona "cuánta responsabilidad" tiene en el error cometido al final, permitiendo así que **cualquier neurona (dentro de cualquier capa) sea capaz de corregir (minimizar) el error que está cometiendo**.

Al igual que hicimos con el descenso del gradiente, ahora debemos calcular cuánto modificar los pesos para las entradas de cada neurona. Para una neurona k ubicada en la capa de salida o, el nuevo valor de su peso $w_{kj}^{o}(t+1)$ será su valor actual $w_{kj}^{o}(t)$ más la suma de su valor actual multiplicado por su correspondiente corrección Δ_p (calculada con el algoritmo de *backpropagation*), donde μ representaba a la **tasa de aprendizaje**: $\boxed{w_{kj}^{o}(t+1) = w_{jk}^{o}(t) + \Delta_p \cdot w_{jk}^{o}(t)}$

$$\left.\begin{aligned} w_{kj}^{o}(t+1) &= w_{jk}^{o}(t) + \Delta_p \cdot w_{jk}^{o}(t) \\ \Delta_p \cdot w_{jk}^{o}(t) &= \mu \cdot (y_{pk} - o_{pk}) \cdot i_{pk}^{o-1} \cdot \frac{\partial (f_k^o(\sigma_{pk}^o))}{\partial \sigma_{pk}^o} \end{aligned}\right\} \rightarrow \boxed{w_{kj}^{o}(t+1) = w_{jk}^{o}(t) + \mu \cdot (y_{pk} - o_{pk}) \cdot i_{pk}^{o-1} \cdot \frac{\partial (f_k^o(\sigma_{pk}^o))}{\partial \sigma_{pk}^o}}$$

Si nos quedamos solamente con el error cometido para una neurona de la capa de salida δ_{pk}^{o} y el resultado obtenido con *backpropagation* tenemos la **variación de los pesos de cada neurona** resultante:

$$\left.\begin{aligned} w_{kj}^{o}(t+1) &= w_{jk}^{o}(t) + \mu \cdot (y_{pk} - o_{pk}) \cdot i_{pk}^{o-1} \cdot \frac{\partial (f_k^o(\sigma_{pk}^o))}{\partial \sigma_{pk}^o} \\ \delta_{pk}^{o} &= \underbrace{(y_{pk} - o_{pk})}_{\delta_{pk}} \cdot \frac{\partial (f_{pk}^o(\sigma_{pk}^o))}{\partial \sigma_{pk}^o} \equiv \delta_{pk} \cdot \frac{\partial (f_{pk}^o(\sigma_{pk}^o))}{\partial \sigma_{pk}^o} \end{aligned}\right\} \rightarrow \boxed{w_{kj}^{o}(t+1) = w_{jk}^{o}(t) + \mu \cdot \delta_{pk}^{o} \cdot i_{pj}^{o-1}}$$

■ **Aplicación del algoritmo de *backpropagation***

Acabamos de ver que somos capaces de modificar los pesos de las neuronas en la capa de salida (conexiones que van de la capa $o-1$ a la capa o) porque en la capa de salida es donde la red neuronal genera la salida deseada. En las capas ocultas no hay ningún valor de salida deseado porque estas neuronas son las que se encargan de ir generando los resultados intermedios, los cuales no conocemos (si los conociésemos no necesitaríamos usar redes neuronales); al no conocerlos, no podemos calcular el error que cometen, por lo que tampoco podríamos modificar sus pesos.

El algoritmo de *backpropagation* soluciona este problema de una forma muy sencilla: **propagar el error cometido hacia las capas anteriores recursivamente**. El error en la capa de salida E_p está relacionado con la salida de las neuronas de su capa anterior ($o-1$), porque las salidas de esta capa son las entradas de la capa de salida. Si nos fijamos en la arquitectura de la red, las **entradas de una capa son las salidas generadas por la capa anterior**, de modo que si corregimos el error en la salida de la capa anterior, la capa actual será capaz de generar mejores resultados; gracias a la recursividad logramos que toda la red sea capaz de generar mejores resultados, porque TODAS las neuronas corregirán su error.

La demostración es muy sencilla: son los mismos cálculos que hicimos anteriormente, pero teniendo en cuenta la regla de la cadena a la hora de calcular las entradas de una neurona como la salida de las neuronas de su capa anterior, por lo que no la escribiremos. Si queremos generalizar la actualización de los pesos de cada neurona para cualquier capa de la red, la fórmula es la misma, pero aplicada a la capa *h*:

$$\boxed{w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \mu \cdot \delta_{pj}^{h} \cdot i_{i}^{h-1}}$$

De este modo, *backpropagation* realiza el mismo proceso de propagación (de ahí su nombre) del error desde la capa h a su capa anterior $h-1$, hasta llegar a la capa de entrada. El error cometido por las neuronas en las capas ocultas se calcula antes de que se lleguen a modificar los pesos de las neuronas en la capa de salida, generando así mejores salidas en cada iteración del algoritmo. La velocidad con la que la red se autocorrige el error que comete depende de la tasa de aprendizaje μ.

Finalmente obtenemos que el **algoritmo de *backpropagation*** consiste en:

1. Inicializar los pesos de las entradas aleatoriamente.
2. Aplicar todos los patrones de entrada a la red cuyos resultados sean conocidos de antemano.
3. Calcular los errores cometidos para todos los patrones anteriores.
4. Modificar los pesos de las neuronas de las capas anteriores en función del error global cometido por la red (*backpropagation*).
5. Repetir este proceso durante un número de ciclos (***épocas***) hasta cumplir una condición de parada:
   - **5.1.** Alcanzar un error de entrenamiento dentro de un rango de valores aceptable.
   - **5.2.** Llegar al número de épocas establecido como límite.
   - **5.3.** La red no es capaz de mejorar el error cometido tras un número fijo de épocas.

ENTRENAMIENTO DE UNA RED NEURONAL

A la hora de entrenar la red neuronal podemos emplear todos los datos que tengamos disponibles, pero en realidad solo necesitamos un **subconjunto de datos que sean representativos**, es decir, que esos datos sean los más parecidos posibles al resto de datos que vaya a recibir la red, dado que esos datos serán con los que la red aprenderá a clasificar los patrones de entrada. Si la red se entrena con datos no representativos (o insuficiente cantidad de datos), las salidas no serán adecuadas, es decir, la red no será capaz de generalizar.

Como vimos anteriormente, lo que se hace al entrenar la red es dividir en regiones a las entradas de la red para indicar si pertenecen a alguna clase (clasificación), siguen algún patrón concreto (ajuste de curvas) o si están relacionados (regresión). Con el entrenamiento se reduce el error, de modo que los planos que dividen las regiones de las entradas se ajustan mucho a los patrones de entrada. Esto puede causar problemas: ¿cómo clasificamos una entrada que no se parece en nada al conjunto de patrones de entrenamiento? Para ello, todo conjunto de entrenamiento debe cumplir 3 propiedades:

- **Ser significativo**: debe tener una cantidad suficiente de patrones para que la red sea capaz de corregir correctamente los errores de sus neuronas.
- **Ser representativo**: debe tener ejemplos de patrones que se adapten a los datos que vaya a clasificar la red en el futuro.
- **Evitar sobreentrenamiento (*overfitting*)**: si el conjunto de entrenamiento tiene más ejemplos de un tipo concreto que los demás, la red se especializará en clasificar los patrones para dicho tipo. El conjunto de entrenamiento debe seleccionarse con cuidado y adaptarse al futuro uso de la red.

■ **Normalización de los datos de entrenamiento**

Dado que la red neuronal trabaja con funciones matemáticas (combinaciones lineales, funciones de activación, etc), es importante que las entradas sean valores numéricos, que pueden ser números enteros o números reales, por lo todas las entradas deben pasar por un proceso de **normalización**: ajustar los valores de entrada para que estén dentro del rango $(0,1)$ o $(-1,1)$. Puede hacerse de 2 maneras:

Normalización entre máximo y mínimo

$$entrada = \frac{valor - minimo}{maximo - minimo}$$

Se usa cuando todos los valores están acotados.

Normalización estadística

$$entrada = \frac{valor - \mu}{\sigma}$$

Asegura que todos los valores sean homogéneos.

Al obtener la salida es importante que los valores también estén normalizados. Para averiguar el valor original que tenía la salida o la entrada se desnormaliza el valor obtenido (despejar en las ecuaciones).

■ **Convergencia en la tasa de aprendizaje μ**

Hemos visto que la red es capaz de corregir cuánto variar su error mediante la tasa de aprendizaje, que es la que se encargaba de determinar qué distancia saltar dentro de la función del error para llegar a su mínimo. Recordemos que si la tasa de aprendizaje era alta, podríamos no llegar nunca a alcanzar el mínimo error, mientras que con una tasa baja, necesitaremos muchas iteraciones para llegar al mínimo.

Saber elegir el valor adecuado de la tasa de aprendizaje ayudará a optimizar cómo de rápido corrige los errores nuestra red neuronal. Generalmente se suele comenzar con un valor $\mu \in (0.05, 0.25)$ para asegurar que se ajustan los pesos correctamente; en función de la pendiente que indique el gradiente, la tasa de aprendizaje se incrementa o reduce para ayudar a llegar al mínimo de la función de error.

Para ayudar a incrementar al velocidad de aprendizaje de la red se puede emplear otro parámetro denominado **Momento**, basándose en que las variaciones anteriores de los pesos deben influir a la hora de calcular el nuevo valor del peso, teniendo su propia ponderación α : si antes la tasa de aprendizaje era muy alta y aún está lejos del mínimo, debe aumentarse más la tasa; sino, debe reducirse porque ya estaremos más cerca del mínimo. Actualizando la ecuación para recalcular el nuevo valor de los pesos:

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \mu \cdot \delta_{pj}^{h} \cdot i_{i}^{h-1} + \underbrace{\alpha \cdot \Delta_{p} w_{ji}^{o}(t-1)}_{Momento}$$

El gran problema que tenemos hasta ahora es que **no conocemos la función de error**, de modo que al tratar de buscar un mínimo en ella con los algoritmos anteriores no sabemos si estamos avanzando hacia un mínimo local o hacia un mínimo global. Una vez que nuestra red entra en un mínimo, su aprendizaje se detiene, incluso cuando el error cometido sea demasiado alto. En la actualizad existen muchos algoritmos para solucionar este problema y tratar de encontrar el mínimo global, pero no los estudiaremos aquí. La solución más sencilla es volver a iniciar el entrenamiento seleccionando un valor aleatorio para los pesos (como hacíamos anteriormente) y a partir de ahí volver a entrenar la red, porque al cambiar los pesos estaremos ubicados en una zona distinta de la función de error.

■ **Conjuntos de Entrenamiento y Test**

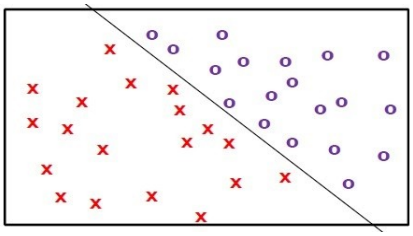
Como vimos, a partir de todos los patrones de entrada disponibles, y de los cuales conocemos las salidas, debemos generar un **conjunto de Entrenamiento** a partir de aquellos datos que sean más significativos para el problema que queremos resolver y que sean lo más parecidos posibles a los datos que recibirá la red neuronal cuando sea usada. El proceso de entrenamiento se encargará de ajustar los pesos de las neuronas para minimizar el error que cometen para los patrones que reciba (conjunto de Entrenamiento). De esta forma sabremos que el error para esos patrones será bajo, pero lo que no sabremos es cómo se comportará la red cuando reciba otros patrones que no pertenezcan a ese conjunto, es decir, ¿nuestra red es capaz de generalizar?

Para saberlo necesitamos tomar otro conjunto de valores disponibles inicialmente y formar el **conjunto de Test**, que se empleará para ver cómo la red es capaz de clasificar los datos y ver qué salidas genera, comprobando así si cumple con su cometido o si está mal entrenada. Debemos destacar que ningún patrón del conjunto de Test debe pertenecer al conjunto de Entrenamiento, puesto que si coincidiesen, la red ya estaría optimizada para clasificar dichos valores y no nos resultaría útil. **El error cometido con el conjunto de Test será el que realmente indique cómo de bien está entrenada la red**.

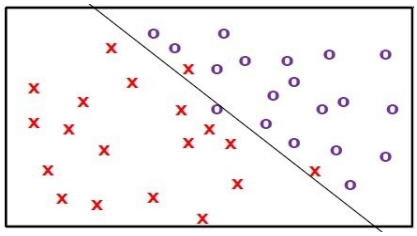
SOBREENTRENAMIENTO (*OVERFITTING*)

Hemos dicho que el conjunto de entrenamiento debe seleccionarse cuidadosamente o podría producirse un **sobreentrenamiento (*overfitting*)**. Esto se produce cuando la red clasifica con mucha precisión las entradas del conjunto de Entrenamiento, pero al recibir datos distintos (conjunto de Test) no es capaz de clasificarlos igual de bien y comete un error más alto: la **red no es capaz de generalizar porque ha aprendido a memorizar patrones y no a clasificarlos**.

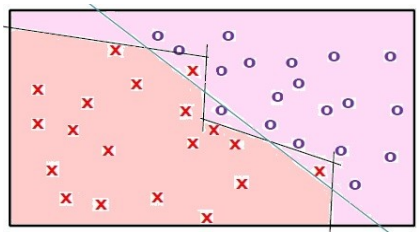
Lo ideal es ser capaz de representar gráficamente los patrones, pero esto resulta imposible cuando las entradas tienen muchos valores (necesitaríamos hiperplanos y al ojo humano ya es difícil de imaginar espacios más allá de las 3 dimensiones). Otro problema es que los patrones de entrada tienen un **ruido** generado a la hora de tomar las mediciones de los valores, al aproximar o incluso al normalizar, por lo que las salidas de la red se calcularán teniendo en cuenta dicho ruido. Generalmente el ruido provocará que al clasificar un patrón en una clase conocida pueda llegar a ser clasificado como otra clase distinta. Este error de clasificación es el principal causante del *overfitting*, dado que la red se especializará en clasificar los valores con sus errores con tanta precisión que, al recibir otra entrada que jamas ha visto (con su correspondiente error), podría llegar a clasificarla donde no debe, y así sucesivamente.



Patrones sin ruido



Patrones con ruido

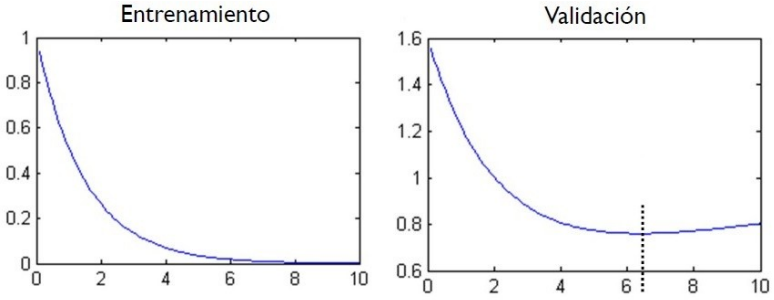


Sobreentrenamiento

Para **evitar el sobreentrenamiento** debemos emplear un **conjunto de Validación** que no interfiera en el entrenamiento pero que lo supervise, e incluso llegue a detener el entrenamiento si es necesario. Generalmente este conjunto se obtiene como una partición del conjunto de Entrenamiento. A medida que se entrena la red, se va evaluando su rendimiento usando el conjunto de Validación, realizando una estimación de la capacidad para generalizar de la red: si el error cometido en el conjunto de validación aumenta significa que la red comienza a sufrir *overfitting*, porque está aumentando considerablemente el error al evaluar patrones que no pertenecen al conjunto de Entrenamiento (porque son conjuntos disjuntos así que no tienen elementos comunes).

Tan pronto se detecte *overfitting* y el error de validación comience a crecer, se debe detener el entrenamiento, porque la red ya no generalizará los patrones correctamente, pero ¿cuándo parar el entrenamiento? Hay varias formas:

- Establecer un número de ciclos fijo sin que se detecte mejora en la validación: si tras superar continuamente dicho número de ciclos y el error de validación no mejora (disminuye) pero tampoco empeora (aumenta), se puede considerar que la red está entrenada.
- Entrenar la red por un número fijo de ciclos muy alto y en cada época recordar cual fue la red que mejores resultados aportó para el conjunto de validación. De este modo, tras finalizar el entrenamiento se devolverá dicha red.



Conjunto de Entrenamiento

- Se encarga de corregir los errores que cometen las neuronas.

Conjunto de Validación

- Supervisa el entrenamiento y lo detiene si es necesario.
- Determina cual es la red óptima de entre todos los entrenamientos.
- Detecta el *overfitting*.

Conjunto de Test

- Evalúa si la red ha sido entrenada correctamente, midiendo el error que comete al generalizar.
- Representa los valores con los que en un futuro trabajará la red neuronal tras su entrenamiento.

VENTAJAS Y DESVENTAJAS DE LAS REDES NEURONALES

■ **Ventajas de las redes neuronales**

Debido a su estructura formada por conexiones entre neuronas y a su capacidad para aprender, las redes neuronales se asemejan al cerebro gracias a que son capaces de aplicar sus conocimientos para generalizar y clasificar patrones concretos, llegando a hacerlo incluso mejor que un cerebro. Además, se fundamentan en conceptos matemáticos muy básicos como la combinación lineal o la derivación. Ofrecen un **aprendizaje adaptativo** puesto que aprenden a realizar tareas tanto básicas como muy complejas, necesitando previamente un proceso de entrenamiento.

Además, las redes neuronales están **autoorganizadas**, es decir, ellas mismas son capaces de repartir el conocimiento que obtienen en el entrenamiento a todas las neuronas, por lo que son capaces de presentar soluciones (incluso aunque sean erróneas) para cualquier conjunto de entradas que se le presente, siempre que esté relacionado con el problema que están destinadas a resolver. Este conocimiento distribuido les aporta una **gran tolerancia a fallos** dado que si un subconjunto de neuronas fallase, el resto de neuronas sería capaz de adaptarse para seguir realizando su trabajo correctamente.

Como ya dijimos, al basarse en conceptos matemáticos sencillos, una red neuronal puede llegar a **implementarse en casi cualquier hardware**, y a día de hoy existen múltiples librerías de código que permiten implementar rápidamente una librería (como `TensorFlow` para *Python*). Sin embargo, dado que su conjunto de operaciones es reducido, puede llegar a desarrollarse hardware concreto para resolver dichas operaciones más eficientemente e implementar las redes neuronales en ordenadores que usen dicho hardware. Por ejemplo, las tarjetas gráficas están compuestas por miles de núcleos especializados en trabajar con matrices y gradientes, por lo que resultan óptimas para trabajar con redes neuronales. Esto también es otra ventaja porque permite **ejecutar redes neuronales en paralelo** en varios ordenadores, acelerando la forma de entrenar y resolver los problemas necesarios.

■ **Desventajas de las redes neuronales**

Si ejecutamos una red neuronal en un ordenador secuencial solamente ejecutará una única instrucción a la vez, por lo que si queremos **paralelizar las redes neuronales** debemos diseñar cuidadosamente cómo se realizará el paso de datos, gestionar la concurrencia y la sincronización de las neuronas.

Por otro lado, hemos visto que añadiendo capas ocultas podemos obtener mejores resultados, pero no existen algoritmos que permitan **calcular cuántas capas ocultas debemos añadir, ni cuántas neuronas** debe tener cada capa. Estos parámetros debemos determinarlos nosotros mediante prueba-error. Además, la existencia de múltiples neuronas y el conocimiento distribuido nos dificulta poder **explicar cómo funciona la red neuronal** en su conjunto, pues la red nos genera los resultados deseados pero no nos explica cómo lo hace, por lo que habrá determinados ámbitos (como la medicina) en los que resulta útil por su efectividad y eficiencia, pero no por cómo logra obtener los resultados. Otra desventaja es que para entrenar la red neuronal necesitamos **disponer de antemano con todos los patrones**, obligándonos a recolectarlos y a clasificarlos posteriormente en los conjuntos de Entrenamiento, Test y Validación.

TEMA 7

OTROS MODELOS DE SISTEMAS CONEXIONISTAS

Redes Autoorganizativas:

Introducción:

- La autoorganización en Biología: proceso a través del cual el comportamiento global de un sistema se obtiene solamente como resultado de la interacción local entre los componentes que integran el sistema.
- La **autoorganización** en Sistemas Conexionistas consiste en la modificación repetida de los pesos de las conexiones en respuesta a modos de activación y siguiendo unas reglas preestablecidas, hasta el desarrollo final de la estructura o sistema.
    - No existen observadores globales, no existe un "jefe" que determine el comportamiento, no se cuenta con la *salida deseada*.
    - Comportamiento emergente: características surgen de forma inesperada a partir de la interacción entre los componentes del sistema.
    - La red crea su propia representación de la información que recibe mediante la etapa de aprendizaje.
    - La información relevante debe ser localizada en los propios datos de entrada (redundancia), sin redundancia sería imposible encontrar características en los datos.

Problemas a resolver:

- El reconocimiento automático, descripción y agrupamiento de **patrones** son actividades importantes en una gran variedad de disciplinas científicas, como biología, psicología, medicina, inteligencia artificial, etc.
- **Patrón**: entidad representada por un conjunto de propiedades y las relaciones entre ellas (vector de características). Por ejemplo, un patrón puede ser:
    - Una señal sonora y su vector de características, el conjunto de coeficientes espectrales extraídos de ella (espectrograma).
    - Una imagen de una cara humana, de la cual se extrae el vector de características formado por un conjunto de valores numéricos calculados a partir de la misma.
- Es importante conocer qué tipo de proceso puede realizar la red autoorganizativa, qué representa la salida de la red y qué problemas puede resolver.

SOM – Mapas autoorganizativos:

- La estructura anatómica e histológica cerebral revela que la **ubicación espacial** de las células nerviosas tiene gran importancia.

    - En la corteza auditiva primaria las neuronas se distribuyen en un mapa según la frecuencia temporal o la tonalidad a la que responden.
    - En la corteza visual las neuronas se organizan en columnas, y en cada columna presentan características similares: selectividad ante la orientación del estímulo.
- Una de las propiedades importantes del cerebro es el significativo **orden de sus unidades de proceso**.
    - Este orden hace que células estructuralmente idénticas rengan una diferente funcionalidad debida a parámetros internos que evolucionan de forma diferente según dicha ordenación.
    - Propiedad fundamental para la representación de imágenes visuales, abstracciones, etc.
- La posibilidad de que la representación del conocimiento en una categoría particular pueda **asumir la forma de un mapa de características** organizado geométricamente sobre la corteza del cerebro, ha motivado una serie de investigaciones que han dado lugar a nuevos modelos de redes neuronales artificiales.
- **La localización** de la neurona en la red, especifica un orden topológico que describe la relación de similitud entre los patrones de entrada.
- **"***El supuesto del procesado inteligente de la información puede ser visto como la creación de imágenes simplificadas del mundo real con diferentes niveles de abstracción, en relación a un subconjunto particular de datos observables" [T. Kohonen]*.
- El científico finlandés Teuvo Kohonen diseñó en 1982 un modelo llamado **mapa autoorganizativo de características** que consiste en una red de neuronas de 2 capas: capa de entrada y capa de competición (salida).
- Modelo de red neuronal con capacidad para formar mapas de características, simulando los mapas topológicos de los fenómenos sensoriales existentes en el cerebro, a través de una organización matricial de neuronas artificiales, que permite:
    - Conseguir un modelo simplificado de los datos de entrada.
    - Obtener un mapa que muestra gráficamente las relaciones existentes entre los datos, preservando su topología.
    - Proyectar datos altamente dimensionales a un esquema de representación de baja dimensión: representar conjuntos de datos de gran número de atributos en mapas 2D.
    - Encontrar similitudes en los datos: visualmente podemos detectar de forma rápida cómo quedan agrupados patrones con valores próximos entre sí (Ej. Color rojo).
- Características principales:
    - Son redes no supervisadas.
    - **Aprendizaje competitivo:** las células compiten por aprender (por modificar sus pesos), solo hay una célula ganadora:
        - Se actualizan los pesos de la neurona ganadora para acercarlos al patrón de entrada.

        - Así se van asociando a cada neurona de la capa competitiva, un grupo de patrones de entrada similares, generando *clusters* o grupos.
        - Los pesos de las neuronas serán los prototipos, centros o centroides de los *clusters*.
    - Presentan una topología predefinida de neuronas en el mapa, el aprendizaje conserva la **relación topológica** (orden topográfico).
    - La **vecindad** entre neuronas preserva las relaciones topológicas: las neuronas cercanas responden ante patrones similares.
        - Se puede definir para cada célula de la capa competitiva el conjunto de células próximas que serán sus vecinas (según arquitectura uni, bi o n-dimensional de la capa).

SOM – Mapas autoorganizativos de Kohonen:

- Se trata de categorizar los datos de entrada, agrupar los datos similares:
    - Se persigue que patrones parecidos hagan reaccionar a las mismas neuronas.
    - Cada neurona se especializa en determinados patrones de entrada.
    - Cada grupo de patrones estará representado por un prototipo (pesos de la neurona).
    - El sistema debe relacionar cada prototipo con los patrones de entrada que representa.
    - Cada prototipo será utilizado, una vez entrenada la red, para clasificar patrones de datos nuevos y desconocidos.
- En el aprendizaje competitivo:
    - Si un patrón nuevo pertenece a una categoría reconocida previamente, entonces la inclusión de este nuevo patrón a esta categoría o clase matizará la representación de la misma.
    - Si el nuevo patrón no pertenece a ninguna de las categorías reconocidas anteriormente, entonces la estructura y los pesos de la red neuronal serán ajustados para reconocer a la nueva categoría.

- La capa de entrada recibe la señal de entrada a la red.
    - La **dimensión** de la entrada depende del número de atributos que tengan los patrones de entrada, si *n* atributos, entonces: $e = \{e_1,\ldots,e_n\}$.
    - Cada neurona de la capa de entrada **se conecta con todas** las neuronas de la capa de competición (no hay conexiones entre las neuronas de competición).
    - Si hay *m* neuronas en la capa de competición, los pesos de las conexiones entre las dos capas se definen mediante una matriz (n, m).
    - El **vector de pesos** de cada neurona de la capa de competición tendrá el mismo nº de componentes que el vector de entrada.
    - Al tener la misma dimensión, se pueden **comparar entre sí** los dos vectores, definiendo una función de distancia entre ellos.

- Los mapas de Kohonen utilizan usualmetne la **<u>distancia euclídea</u>**, la salida de las neuronas de la capa de competición se calcula aplicando dicha función de distacia:

$$\tau_j = \sqrt{\sum_{i=1}^{n} (e_i - \mu_{ij})^2}$$

τ_j – salida de la neurona j
e_i – entrada i
μ_{ij} - peso de ei a neurona j

- **Funcionamiento del modelo – FASE APRENDIZAJE:**
  - Objetivo: acercar los pesos a los vectores de entrada.
  - Los pesos de las conexiones se inicializan aleatoriamente al principio de la fase de aprendizaje.
  - Se introducen todos los patrones de entrenamiento un número suficiente de veces.
  - Cada vez que la red recibe un patrón o vector de entrada, se calcula la salida de todas las células de la capa de competición (cálculo de distancias).
  - Gana la célula de **menor distancia** con ese vector de entrada. Dicha célula tendrá salida 1 y el resto salida 0.
  - Por tanto, los **pesos de la neurona ganadora** se modifican siguiendo la ecuación siguiente, donde α es la tasa de aprendizaje, que habitualmente decrece con el tiempo (primero tasa elevada y luego baja) hasta llegar a 0 o al final de las iteraciones de entrenamiento (finaliza el aprendizaje):

$$\mu_{ij}(t+1) = \mu_{ij}(t) + \Delta\mu_{ij}$$
$$\Delta\mu_{ij} = \alpha(e_i - \mu_{ij})$$

- **Funcionamiento del modelo – FASE APRENDIZAJE (vecindad):**
  - **Vecindad:** indica qué otras células aprenden cuando se activa una neurona.
  - Los pesos de las conexiones de las células vecinas se modifican en función de la distancia, con lo que α se dividirá por una <u>función</u> *distancia de vecindario* cuando se aplique a las vecinas la modificación de los pesos, siendo c_i la célula ganadora y c_j la vecina:

$$\Delta\mu_{ij} = \frac{\alpha}{d(c_i, c_j)}(e_i - \mu_{ij})$$

  - La **vecindad** también se modifica con el tiempo o el número de iteraciones, inicialmente es mayor y se va reduciendo para estabilizar el aprendizaje.
  - La **vecindad** se modifica con el tiempo o el número de iteraciones, inicialmente es mayor y se va reduciendo para estabilizar el aprendizaje.
  - **Vecindad**: regulada por 3 parámetros:
    - <u>Radio de vecindad</u>: amplitud del alcance de las neuronas afectadas por vecindad.
    - <u>Topología de la vecindad</u>: neuronas que se consideran vecinas inmediatas.

        - Función de vecindad: cuantificación del grado de vecindad proporcional a distancia a la ganadora (centro).
    - Las neuronas pertenecientes a la región de vecindad o interés del ganador son también modificadas para conseguir que la red cree **regiones** que respondan a valores muy próximos al del patrón de entrenamiento.
    - Como consecuencia, patrones de entrada que no se hayan usado para el entrenamiento serán similares a algún vector de pesos de una región y serán correctamente agrupadas -> **demuestra la Generalidad de esta estructura**.
    - 2 ideas centrales en las que se basa Kohonen para desarrollar sus estructuras usando aprendizaje autoorganizativo y competitivo: "*El proceso de adaptación de pesos y el concepto de geometría topológica de elementos de proceso*".
- **Funcionamiento del modelo – FASE DE APRENDIZAJE (algoritmo).** *Tendencia del proceso de aprendizaje: cada neurona tiende a colocarse en el centroide de aquellos grupos de ejemplos de entrada para los cuales es la neurona ganadora.*
    1. Inicializar pesos.
        - Asignar a los pesos valores pequeños aleatorios.
    2. Presentar una entrada.
        - El conjunto de aprendizaje (ξ) se presenta repetidas veces hasta llegar a la convergencia de la red.
        - Actualizar α (ir reduciendo su valor).
    3. Propagar el patrón de entrada hasta la capa de competición.
        - Obtener los valores de salida (distancias) de las neuronas de dicha capa.
    4. Seleccionar la neurona ganadora (BMU – *best-matching unit*).
        - La de menor distancia al patrón.
    5. Actualizar conexiones entre capa de entrada y la BMU.
        - Actualizar también los pesos de sus vecinas según el grado de vecindad.
        - 1. Fase Ordenación: valores altos de α y radio vecindario. 2. Fase Convergencia: valores bajos.
    6. Si α se mantiene por encima del umbral de parada (o no se ha llegado a T iteraciones), volver a 2, en caso contrario FIN.

- **Funcionamiento del modelo – Validar calidad del SOM.**
    - Se repite el proceso de entrenamiento N veces, cada una de ellas con una configuración inicial diferente de los vectores de pesos sinápticos.
    - Para determinar la calidad del SOM y ayudar en la selección de sus parámetros de tamaño y aprendizaje adecuados se emplean el ***error de cuantización medio*** y ***medidas de preservación de la topología***.
        - Se calcula el error de representación para cada uno de los mapas obtenidos y se selecciona el de menor error.
        - Este error de cuantización medio, evalúa el grado de adaptación del mapa SOM a los datos de entrada, siendo mejor el que menor

error obtenga entre las conexiones de las BMU y los vectores (patrones de entrada) a los que representa.

- Una vez entrenada la red, se vuelven a procesar todos los patrones de entrenamiento, obteniendo para cada uno de ellos la neurona BMU, y se calcula el error de cuantificación medio como:

$$E = \frac{1}{N}\sum_{i=1}^{N}\left\|ei - w_{bmu}\right\|^2$$ siendo N el número de vectores de entrada.

- Además, se emplean ***Medidas de preservación de la topología***:
  - El error de cuantización indicado, no tiene en cuenta la estructura del mapa de la capa competitiva de la SOM, la información sobre las relaciones existentes en los patrones de entrenamiento.
    - Por lo que también es necesario medir la adaptación de la topología de la red a las características topológicas del espacio de datos de entrenamiento.
  - Cuando una red SOM converge, la distribución de la estructura regular definida en la capa de salida se suele distorsionar, pudiendo aparecer efectos adversos (neuronas que representan a datos de entrada similares pero que están alejadas, etc.).
    - En este caso el error de cuantización puede ser bueno, pero a pesar de ellos existen problemas.
  - Es necesario emplear además **medidas de preservación de la topología para validad la calidad** de la red entrenada que permitan comprobar que:
    - Los prototipos de las neuronas que se encuentran próximas en el mapa también se encuentren próximos en el espacio de entrada de datos.
    - Los datos de entrada similares quedan representados por neuronas próximas en el mapa.

- **Funcionamiento del modelo – MODO OPERACIÓN:**
  - Se introduce un patrón de entrada para conocer su prototipo.
  - Cada neurona de salida calcula la similitud entre el vector de entrada y su vector de pesos.
  - Vence la neurona con mayor similitud: será la categoría seleccionada por la red para agrupar ese patrón de entrada o el prototipo seleccionado.

<u>Mapas auto-organizativos:</u>

- Una demostración <u>visual</u> del funcionamiento de la red la ofrecen los *mapas topológicos **bidimensionales** de Kohonen.*
  - Distribuyen los elementos de la red (puntos que representan a los pesos) a lo largo de un conjunto de patrones de entrada.
  - Cuando las entradas son de 2 atributos o 3 (2D y 3D), los pesos representados por puntos coordenadas en 2D se van aproximando a los

datos de entrada que están ahí representados, en un cuadrado 2D, por ejemplo.
    - Se representan los puntos (pesos de las neuronas) conectados con los de sus vecinas, no es una conexión física, sino una relación entre neuronas a efectos de vecindario (conexión topológica).
        - Grado de vecindad: número de células que habrá que recorrer para llegar de una célula a otra.
- *Mapas topológicos bidimensionales de Kohonen:*
    - A medida que se introducen los patrones de entrada, la red los va aprendiendo, varía el valor de sus conexiones, trasladándose hacia las entradas.
    - Donde hay mayor densidad de entradas, habrá más densidad de "puntos conexiones – neuronas".
    - Se comprueba si la red está comportándose correctamente si no se cruzan las líneas.
- **Aplicaciones**: Tipos de problemas que pueden resolver los SOM, condicionará cómo se interprete lo que representa la salida de un SOM.
    - **Agrupamiento de patrones (clustering):** A partir de un conjunto de entrada se desea determinar si se puede dividir ese conjunto en diferentes grupos o *clusters* con un centro o patrón del *cluster*.
        - Permite separar datos en grupos (a priori no sabemos cuántos existen).
        - Facilita indicar a qué grupo pertenece cada dato de entrada determinando su neurona ganadora.
        - Permite caracterizar cada grupo mediante los pesos de la neurona que representa ese grupo.
    - **Prototipado:** Similar al anterior, en lugar de interés por los grupos, interesa obtener un prototipo del grupo al que pertenece cada patrón de entrada.
        - Usa los pesos de la ganadora para determinar ese prototipo.
    - **Análisis de componentes principales:** Se trata de detectar qué elementos/atributos del conjunto de entrada caracterizan en mayor grado ese conjunto de datos. Seleccionar las entradas de la red realmente necesarias, sin pérdida de información – supone reducción de dimensionalidad.
        - Las demás entradas podrán eliminarse sin una pérdida significativa de información.
    - **Extracción y relación de características:** Se pretende organizar los vectores de entrada en un mapa topológico.
        - A partir de la red entrenada, patrones parecidos producirán respuestas similares en neuronas cercanas.
        - Si existe una organización global de patrones de entrada, se verá reflejada en la salida de la red.
        - Se ubican entradas parecidas y/o relacionadas en zonas próximas de la red.

    - Estas categorías no son disjuntas. Ej.: componentes principales para reducir la dimensionalidad y después aplicar clustering.
- **Ventajas de SOM:**
    - Transparentes a los datos de entrada (se limitan a comparar vectores).
    - Adaptación local de los vectores de referencia a la densidad de probabilidad de los datos.
    - Facilidad de visualización gracias a la malla que conforma la topología de la capa de salida.
    - Facilidad de integración con otras técnicas.
- **Limitaciones o inconvenientes de SOM:**
    - Necesidad de determinar la arquitectura exacta de la red antes de someterla al entrenamiento, así como la imposibilidad de modificar su diseño durante el mismo tiempo.
        - Dimensión de la red (se tiene que fijar a priori – más neuronas > tiempo entrenamiento).
        - Velocidad y vecindad del aprendizaje.
    - Algunas neuronas pueden no ser entrenadas: pueden existir vectores de pesos muy distanciados de las entradas, con lo cual, nunca ganarán (este tipo de neurona es común cuando se aplica el algoritmo a redes y conjuntos de datos muy grandes).
        - Patrones cercanos activan neuronas distantes.
        - Neuronas vecinas pueden ser activadas por patrones distantes.
    - No se puede medir hasta que punto es buena la neurona vencedora, la neurona que gana es la más cercana a la entrada, pero no podemos cuantificar si está cerca o lejos de los pesos adecuados.
    - Importantísimo en estos sistemas: la inicialización de pesos.
    - Son caros computacionalmente cuando se incrementa la dimensión de los datos.

Otros modelos autoorganizativos: Crecimiento de Redes.

- A pesar de las interesantes características del SOM de Kohonen, en la década de los años 90 se representaron nuevos modelos de mapas autoorganizativos que pretendían solucionar los inconvenientes.
- Al trabajar con SOM hay que definir previamente la estructura de la red.
    - Esta definición suele ser bastante compleja por la falta de información que normalmente se tiene sobre el espacio de entrada.
    - La topología estática de los SOM limita a veces su capacidad para ajustarse a los datos.
- Para solucionar este problema aparece una nueva filosofía a la hora de aplicar redes de neuronas a problemas de agrupamiento: **Crecimiento de Redes.**
    - Se trata de iniciar el aprendizaje con un conjunto pequeño de elementos de proceso (EP) y que sea la propia red la que determine de forma autónoma su estructura, a partir de los datos de entrenamiento se generan los EP necesarios y las conexiones entre ellos.

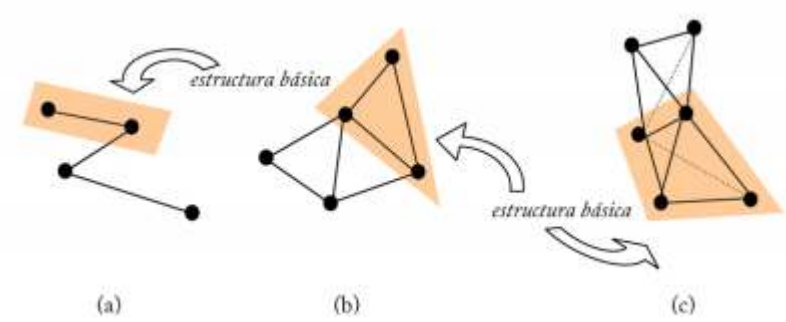
- Esta característica permite obtener modelos simplificados de los datos **con mejores medidas de la preservación de la topología** que las redes de Kohonen.
- Se construyen partiendo de estructuras básicas (*incluso hipertetraedros de dimensión k*).
- Para llegar a la estructura final de la red se realiza un proceso donde se van añadiendo y borrando EP y conexiones, al tiempo que se sigue un proceso de autoorganización similar al que se produce en el modelo propuesto por Kohonen.
- Incluso pueden existir mallas de vecindad separadas en la capa de salida.
- Mantienen la capacidad de ofrecer un mapa para visualizar las relaciones existentes entre los patrones de entrada.

- Inserción de Neuronas:
  - Para poder determinar dónde y cómo se ha de incluir una nueva neurona se introduce un nuevo concepto, **valor *resource*** (error de cuantización): valor asignado a cada neurona que medirá lo lejos que se encuentra de la estructura ideal (se obtiene del error con respecto a los patrones de entrada).
    - Este valor irá variando a medida que la red vaya evolucionando.
  - Siempre después de un nº constante de "modificaciones de pesos" sobre la red actual, se analizará la necesidad o no de añadir nuevas neuronas.
    - Añadir un nuevo elemento a la red conlleva el adecuar las conexiones entre los elementos vecinos para que la nueva estructura sea consistente.
  - Nueva neurona: tanto su vector de pesos como su valor *resource*, se calcularán a partir de los parámetros de sus neuronas vecinas.
- Borrado de Neuronas:
  - Cuando el conjunto de patrones o vectores de entrada que pueden activar una determinada neurona tienen una probabilidad de aparición muy baja, podemos estimar que el coste de mantener esa neurona dentro de la estructura de la red no compensa y, por lo tanto, se elimina.

Growing Cell Structures (GCS):

- Entre los modelos de crecimiento de redes, se creó el modelo conocido como Growing Cell Structures (GCS), planteado por B. Fritzke en 1994.
- La red GCS ofrece como principal ventaja frente al modelo de Kohonen la dinamicidad de la arquitectura de la capa de salida de la red durante la fase de entrenamiento de la misma.
  - Orientada a cuantificación vectorial – clustering (no supervisada).
  - La adición de neuronas (EP) se realiza en las regiones con más patrones de entrenamiento.
  - Su estructura de la capa de salida es simple y está formada por **estructuras k-dimensionales básicas**.

    - El número de parámetros de entrenamiento y la complejidad de su configuración son normalmente bajos, aunque a veces difíciles de interpretar (inconveniente).
- El objetivo es conseguir una estructura cuyos vectores de entrenamiento se distribuyan de acuerdo a la distribución (desconocida) de los vectores de entrada.
    - Patrones/vectores de entrada similares se agruparán en neuronas topológicamente próximas.
    - Neuronas vecinas agruparán vectores de entrada similares.
    - Regiones del espacio de entrada con una densidad alta serán representadas por un número elevado de neuronas y viceversa.
- Durante el proceso de adaptación de pesos de la red, se insertan y eliminan neuronas/conexiones de vecindad asegurando que la capa de salida queda compuesta por **estructuras k-dimensionales básicas**.
- Se muestran estructuras básicas para k igual a 1 (a – segmento), 2 (b – triángulo) y 3 (c – tetraedro).
- k dimensiones (hipertetraedros). Existirán: k+1 vértices y (k+1)*k/2 aristas.



- Exceptuando la topología de la capa de salida, el resto de la arquitectura de una red GCS es idéntica a la del modelo de Kohonen.
- respecto a la dinámica de procesamiento de vectores, el entrenamiento es competitivo, pero cada neurona tiene además de vecinas y vector de pesos, un **contador** que almacena el **error de cuantización**: cuadrado de la distancia euclídea al patrón de entrada cuando dicha neurona es ganadora (BMU):
    - Se utilizan **2 tasas de aprendizaje**: una para modificar los pesos de la BMU y otra para los de sus vecinas inmediatas.

      $\Delta w_{bmu} = \alpha_1(e - w_{bmu}); \Delta w_n = \alpha_2(e - w_n)$

    - No se hace uso de una función de vecindad que varíe según avanza el entrenamiento, solo se modifican los pesos de las vecinas inmediatas.

Growing Neural Gas (GNG):

- En 1995 Fritzke planteón el modelo Growing Neural Gas (GNG) Gas Neuronal Creciente.
- Es otro modelo de red autoorganizativa dinámica, pero a diferencia de la GCS no mantiene en el proceso de inserción y eliminación de neuronas ninguna estructura k-dimensional estricta o fija en la capa de salida de la red.

    - Se adapta a diferentes dimensiones y puede ir variando la dimensión de la estructura dimensional que emplea en función de los datos de entrada.
    - A diferencia de la GCS, la GNG trabaja más con **aristas** que con triángulos.
    - Con la inserción y borrado de aristas la red trata de adaptarse a los vectores de entrada.
    - Los parámetros introducidos son constantes en el aprendizaje.
    - En los mapas de Kohonen, las conexiones de vecindad son laterales formando una cruz en cada unidad, en GNG, una unidad puede tener mucho más de cuatro vecinos, generando diversas figuras geométricas.
    - Se trata, por tanto, de una red con mayor capacidad de aprendizaje.
- Se adapta a diferentes dimensiones y puede ir variando la dimensión de la estructura dimensional que emplea en función de los datos de entrada.

TEMA 8

COMPUTACIÓN EVOLUTIVA

¿Qué es la evolución?

- Proceso de optimización natural.
- Historia:
    - Lamarck
        - Herencia de caracteres adquiridos.
    - Darwin “El origen de las especies”
        - Selección natural.
    - Mendel
        - Mecanismo de herencia.
    - De Vries
        - Mutación. Cambios abruptos.
    - Neo-darwinismo
        - Suma de las anteriores.
    - Teoría Neutralista de evolución molecular
        - Deriva genética.

Ideas desde la Naturaleza.

- Aplicaciones desde la naturaleza:
    - Leonardo da Vinci
        - Vuelo de pájaros y desarrollo posterior de aviones (año 1500).
- Computación evolutiva (CE).
    - Definir un problema sin solución algorítmica.
    - Población de individuos o posibles soluciones.
    - Operadores genéticos:
        - Selección.
        - Cruce y mutación.
        - Sustitución.

Orígenes:

Basada en la teoría de la evolución de Darwin.

- John Holland, “planes reproductivos”: técnica que incorpora la selección natural en un programa de computadora.
- Su **objetivo**: que las computadoras aprendieran por sí mismas.
- John Holland se preguntaba cómo lograba la naturaleza crear seres cada vez más perfectos.
  A principios de los 60 aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje, y tomó la decisión de aplicar estas ideas para desarrollar programas bien adaptados para un fin determinado.
- David Goldberg conoció a Holland y se convirtió en su estudiante. Fue uno de los primeros que trató de aplicar los algoritmos genéticos a problemas industriales.

Investigadores e innovadores:

- 1948 Turing: propuso la búsqueda por medios evolutivos o genéticos.
- 1962 Bremermann: propone la optimización a través de la recombinación y la evolución.

1964 Rechenberg: introdujo los conceptos de estrategias de evolución.

- 1965 L. Fogel, Owens and Walsh: introdujo la programación evolutiva.
- 1975 Holland: Realmente es el creador de la teoría en 1975 y reedita su libro en 1992.
- 1989 D. Goldberg: Promueve el uso de algoritmos genéticos para aprendizaje y optimización.
- 1992 Koza: introduce la programación genética.

Orígenes: ¿Qué es un Algoritmo Genético?

- ¿Podemos crear un algoritmo con la misma filosofía que emplea la naturaleza?
    - En respuesta a esto nacieron los algoritmos genéticos.
- Los algoritmos genéticos establecen una analogía entre el conjunto de posibles soluciones de un problema y el conjunto de individuos de una población natural.

Orígenes: Fundamentos de los AG:

- La naturaleza utiliza potentes medios para impulsar la evolución satisfactoria de los organismos, el proceso de selección natural.
- Los organismos que son poco aptos para un determinado ambiente mueren, en tanto que los que están bien adaptados para vivir, **se reproducen**, y transmiten sus caracteres.
- Ocasionalmente se producen **mutaciones** al azar, y aunque implican la pronta muerte del individuo mutado, algunas mutaciones dan como resultado nuevas y satisfactorias especies.

Algoritmos Genéticos.

Los algoritmos Genéticos (AG) son algoritmos de búsqueda inspirados en procesos de selección natural.

Se aplican principalmente en problemas de optimización. Se comportan de un modo muy eficaz en problemas de superficie compleja, con múltiples mínimos locales y grandes espacios de búsqueda.

Está justificado su uso en aquellos problemas cuya complejidad no permita una solución directa. Por ejemplo los no resolubles polinomicamente (NP-duros).

Componentes de un AG:

Generalmente se acepta que un Algoritmo Genético debe tener 5 componentes:

1. Una representación genética de las soluciones del problema, es decir, representación de las variables que intervienen en cadenas de bits/caracteres.

2. Una forma de crear una población inicial de soluciones.
3. Una función de evaluación en términos de conveniencia o adaptación de la solución evaluada.
4. Operadores genéticos que cambien la composición de los descendientes.
5. Valores para los parámetros utilizados por los Algoritmos Genéticos (N, Pc, Pm, …).

Población y Evaluación de individuos:

- Se genera aleatoriamente la población inicial.
- A cada uno de los individuos o cromosomas generados se le aplicará la **función de aptitud o evaluación**.
- Esta función que debe ser capaz de "castigar" a las malas soluciones, y de "premiar" a las buenas, de forma que sean estas últimas las que se propaguen con mayor rapidez.
- Base para determinar qué soluciones tienen mayor o menor probabilidad de sobrevivir.

Ejercicios

El conjunto de todas las FBDs que se pueden construir legalmente a partir de alfabetos de denomina...

A. Fórmula bien definida.
B. Conjunto inferencial.
C. Constante.
D. Elemento gramatical.
E. Lenguaje formal.

En lógica formal, la ingeniería del conocimiento supone:

A. Identificar y comprender el conocimiento relevante.
B. Formalizar enunciados.
C. Fragmentar los enunciados en sus partes constituyentes.
D. Establecer la simbología adecuada para representar elementos y relaciones.
E. Construir las FBDs.

El proceso de resolución por refutación en lógica formal…

A. Trata de establecer la verdad de una fórmula compleja mediante la reducción sucesiva, desde dentro hacia fuera, usando la tabla de la verdad.
B. Trabaja sobre declaraciones previamente normalizadas.
C. No siempre está definido.
D. Intenta encontrar que la negación de una declaración produce una contradicción axiomática.
E. Obliga a realizar inferencias particulares sobre el conjunto de axiomas establecido.

El objetico fundamental de la llamada Fórmula Normalizada Conjunta de Davis es…

A. Simplificar las FBDs separando los cuantificadores del resto de la fórmula.
B. Eliminar implicaciones.
C. Normalizar variables para que cada cuantificador esté ligado a una única variable.
D. Elimina cuantificadores existenciales.
E. Identificar las cláusulas.

SISTEMAS DE PRODUCCIÓN

Los sistemas de producción…

A. Son sistemas inteligentes basados en reglas.
B. Representan el conocimiento declarativo como reglas, y el procedimental como frames.
C. Incorporan de forma explícita en su estructura los mecanismos de emparejamiento.
D. Representan el conocimiento procedimental como reglas, y el declarativo como frames.
E. Incorporan de forma implícita en su estructura los mecanismos de emparejamiento.

En relación con los sistemas de producción…

A. En los sistemas dirigidos por los datos las inferencias se obtienen cuando los antecedentes de una (o más) de sus reglas se emparejan con (al menos) una parte de los hechos que describen el estado actual.
B. En los sistemas dirigidos por los objetivos las inferencias se obtienen cuando los antecedentes de una (o más) de sus reglas se emparejan con (al menos) una parte de los hechos que describen el estado actual.
C. En los sistemas dirigidos por los datos la meta se alcanza mediante un proceso evocativo en el que, de forma recursiva, se establecen los antecedentes de las metas como submetas de orden inferior.
D. En los sistemas dirigidos por los objetivos la meta se alcanza mediante un proceso evocativo en el que, de forma recursiva, se establecen los antecedentes de las metas como submetas de orden inferior.
E. En los sistemas dirigidos por los objetivos antecedentes y consecuentes son aserciones sobre los datos.

En los sistemas de producción, la entidad que describe e universo de discurso que se quiere modelar se denomina…

A. Base de reglas.
B. Memoria activa.
C. Motor de inferencias.
D. Base de conocimientos.
E. Base de hechos.

El motor de inferencias de un sistema de producción…

A. Interactúa con el mundo exterior a través de la base de la base de hechos.
B. Reconoce y activa las reglas en función de los criterios de activación elegidos.
C. Interactúa con el mundo exterior a través de la base de reglas.
D. Interactúa con el mundo exterior a través de la base de conocimientos.
E. Contiene los mecanismos necesarios para examinar la memoria activa y determinar qué reglas deben ejecutarse.

En los sistemas de producción, las hipótesis de trabajo y las metas o submetas que todavía no han sido establecidas, forman parte de…

A. La base de conocimientos.
B. La base de reglas.
C. La base de hechos.
D. La memoria activa.
E. El motor de inferencias.

Cuando un proceso inferencial se detiene, la memoria activa contiene una descripción del estado final del problema, que incluye…

A. Datos.
B. Hechos.
C. Reglas activas.
D. Reglas ejecutadas.
E. Hipótesis.

En el motor de inferencias, el programa secuencial cuya misión es determinar el siguiente paso que debe ejecutarse de denomina…

A. Emparejador.
B. Elicitador.
C. Intérprete.
D. Cotejador.
E. Compilador.

El ciclo básico de un sistema de producción está constituido por las fases de…

A. Diferenciación.
B. Decisión.
C. Selección de reglas.
D. Acción.
E. Ejecución de reglas.

La Restricción es una tarea de la fase de…

A. Activación.
B. Equiparación.
C. Cotejo.
D. Acción.
E. Decisión.

La ejecución de las reglas seleccionadas, en la fase de acción, concluye con…

A. El proceso inferencia.
B. La verificación de si continuar o no el proceso cíclico.
C. El proceso evocativo.
D. El marcaje de las estructuras utilizadas.
E. La actualización de la memoria activa.

RAZONAMIENTO CATEGÓRICO Y CORRECCIÓN BAYESIANA.

Dado un conjunto X de manifestaciones posibles en un dominio, y un conjunto de Y de interpretaciones posibles, y sean f(x) y g(x) dos funciones booleanas, en un problema concreto…

A. f(xi)=1, si xi no es una manifestación de mi problema.
B. f(xi)=0, si xi no es una interpretación de mi problema.
C. g(yi)=0, si yi no es una manifestación de mi problema.
D. g(yi)=1, si yi es una interpretación de mi problema.
E. f(xi)=0, si xi no es una manifestación de mi problema.

Las relaciones causales entre manifestaciones e interpretaciones se formalizan a través de la función de conocimiento E, E(X, Y) de forma que si X son manifestaciones, si Y son interpretaciones, f(X) y g(Y) son respectivamente funciones booleanas, entonces el problema lógico de la interpretación diferencial es…

A. E: (f→ g).
B. E: (no f→ g).
C. E: (f→ no g).
D. E: (no g→ no f).
E. E→ (f : g).

El conjunto de complejos manifestación-interpretación…

A. Representa el total de situaciones idealmente posibles en el problema.
B. Es un conjunto completo.
C. Se caracteriza porque sus elementos son mutuamente excluyentes.
D. Representa el total de situaciones idealmente posibles en el dominio.
E. Representa el total de situaciones realmente posibles en el domino.

Si el conocimiento sobre un dominio es completo y ha sido aplicado, y el dominio está bien descrito en términos de manifestaciones e interpretaciones, la solución a cualquier problema estará en…

A. La función de conocimiento E.
B. La base lógica expandida.
C. La base lógica reducida.
D. La función booleana de manifestaciones.
E. La función booleana de interpretaciones.

Uno de los problemas del procedimiento sistemático de interpretación diferencial en el modelo categórico…

A. Es identificar la función de conocimiento E.
B. Es crear la base lógica expandida.
C. Es crear la base lógica reducida.
D. Es la explosión combinatoria.
E. Es identificar las manifestaciones y las interpretaciones.

Sea un universo N, y un conjunto de atributos f(x,y,…,z), la expresión P(f) = N(f)/N…

A. Es la probabilidad total de N si N(f) es el nº de elementos del universo que presenta los atributos f.
B. Es la probabilidad total de N(f) si N(f) es el nº de elementos del universo que presenta los atributos f.
C. Es la probabilidad total de f si N(f) es el nº de elementos del universo que presenta los atributos f.
D. Es la probabilidad total de f si N(f) es el nº de elementos del universo que presenta los atributos N.
E. Ninguna de las otras es correcta.

La probabilidad condicional…
A. Es “1” si la probabilidad total es “0”.
B. se caracteriza porque aparecen involucrados dos sucesos, en donde la ocurrencia del mero depende de la ocurrencia del segundo.
C. Es “0” si la probabilidad total es “1”.
D. Es la probabilidad total de f si N(f) es el nº de elementos del universo que presenta los atributos N.
E. Es conocida también como la probabilidad de las causas.

En su forma más sencilla, la ecuación del teorema de Bayes se puede escribir del siguiente modo…
A. P(A/E)={P(E/A)P(A)}/P(E)
B. P(E/A)={P(A/E)P(E)}/P(A)
C. P(A/E)={P(E/A)P(E)}/P(A)
D. P(E/A)={P(A/E)P(A)}/P(E)
E. P(E)={P(E/A)P(A)}/P(A/E)

Señale las expresiones que son ciertas:
A. P(A y E)= P(A)P(E/A)
B. P(A o B)= P(A)P(E/A)
C. P(A y E)= P(A)P(A/E)
D. P(A y E)= P(E)P(E/A)
E. P(A y E)= P(E)P(A/E)

Para que el esquema bayesiano sea aplicable, corrigiéndolo, al modelo categórico de razonamiento…
A. La función de conocimiento E debe poder expresarse como una probabilidad condicional.
B. La función booleana de manifestaciones del problema debe poder expresarse como una probabilidad condicional.
C. La función booleana de manifestaciones del dominio debe poder expresarse como una probabilidad condicional.
D. Manifestaciones e interpretaciones deben ser mutuamente independientes.
E. La función booleana de interpretaciones del problema debe poder expresarse como una probabilidad condicional.

MODELOS CUASI-ESTADÍSTICOS DE RAZONAMIENTO.

Las medidas de confianza y desconfianza crecientes…
A. Valen 0 cuando la relación causal es categórica.
B. Son índices dinámicos que representan incrementos de confianzas asociados a evidencias nuevas.
C. Valen 1 cuando la relación causal es categórica.
D. Si una toma un valor mayor que 0, la otra toma el mismo valor absoluto pero signo negativo.
E. Si una toma un valor mayor que 0, la otra toma el valor 0.

Sean MB(h,e) y MD(h,e) –respectivamente- la medida de confianza creciente y la medida de desconfianza creciente, sea p(h) la confianza previa en h antes de e, sea p(h/e) la confianza en h tras aparecer e, y sea 1 – p(h) la desconfianza en h antes de e…
A. Si p(h/e) > p(h), entonces MB(h,e) > MD(h,e).
B. Si p(h/e) > p(h), entonces MB(h,e) > 0, y MD(h,e) < 0.
C. Si p(h/e) < p(h), entonces MB(h,e) = 0, y MD(h,e) < 0.
D. Si p(h/e) < p(h), entonces MB(h,e) = 0, y MD(h,e) > 0.
E. Si p(h/e) > p(h), entonces MB(h,e) > 0, y MD(h,e) = 0.

Sean MB(h,e) y MD(h,e) –respectivamente- la medida de confianza creciente y la medida de desconfianza creciente, sea p(h) la confianza previa en h antes de e, sea p(h/e) la confianza en h tras aparecer e, y sea 1 –p(h) la desconfianza en h antes de e…

A. Si p(h/e)=p(h) MB(h,e) y MD(h,e) no están definidas.

B. Si p(h/e)=p(h) la nueva evidencia ni aumenta la confianza ni aumenta la desconfianza.

C. Si p(h/e)=p(h) la nueva evidencia es independiente de la hipótesis considerada.

D. Si p(h/e)=p(h), entonces MB(h,e)=MD(h,e)=0.

E. No existe una relación causal de ningún tipo entre e y h.

El factor de certidumbre CF(h,e)…

A. Es una medida del grado de compatibilidad entre evidencias e hipótesis.

B. Sirve como un medio para facilitar la comparación entre potencias evidenciales de hipótesis alternativas en relación con una misma evidencia e.

C. Es igual a MB(h,e) cuando la evidencia apoya a la hipótesis, e igual a MD(h,e) cuando la evidencia va en contra de la hipótesis.

D. Se define como CF(h,e) = MB(h,e) – MD(h,e).

E. Se define como CF(h,e) = MB(h,e) + MD(h,e).

Dada una evidencia e y una hipótesis h…

A. Si h es falsa, MB(h,e)=0, MD(h,e)=1, CF(h,e)=-1.

B. Si h es falsa, MB(h,e)=0, MD(h,e)=1, CF(h,e)=1.

C. Si h es cierta, MB(h,e)=1, MD(h,e)=0, CF(h,e)=1.

D. Si h es falsa, MB(h,e)=-1, MD(h,e)=0, CF(h,e)=-1.

E. Si h es cierta, MB(h,e)=1, MD(h,e)=0, CF(h,e)=-1.

En la teoría de Dempster y Shafer, el marco de discernimiento…

A. Es un conjunto exhaustivo de hipótesis mutuamente excluyentes.

B. Es el conjunto finito de todas las hipótesis que se pueden establecer en el problema del dominio.

C. Es el conjunto de los subconjuntos que se pueden construir en el problema del dominio.

D. Es el conjunto finito de todas las hipótesis que se pueden establecer en el dominio del problema.

E. Es el conjunto de los subconjuntos que se pueden construir en el dominio del problema.

Sea un marco de discernimiento, A un subconjunto de ese marco, y m una función de asignación básica de verosimilitud…

A. m{conjunto vacío}=0

B. En ausencia de toda evidencia, m{marco}=1

C. A es un elemento focal si m(A) es distinto de 0

D. m{conjunto vacío}=1

E. A es un elemento focal si m(A)=0

El grado de conflicto K…

A. Nunca puede ser 1.

B. Puede valer 1 en algún caso.

C. De alguna manera evalúa el grado de compatibilidad d las evidencias que están siendo combinadas.

D. Puede ser 0 en algunos casos.

E. Si vale 1 la combinación de evidencias no está definida.

En la teoría evidencial, el intervalo de confianza…
A. Representa la incertidumbre asociada al elemento focal.
B. Tiene como valor máximo el valor de la credibilidad del elemento focal, y como valor mínimo el correspondiente valor de la plausibilidad.
C. Puede usarse como índice dinámico de confianza.
D. Tiene como valor mínimo el valor de la credibilidad del elemento focal, y como valor máximo el correspondiente valor de la plausibilidad.
E. Representa la incertidumbre asociada al marco de discernimiento.

En el contexto de la teoría evidencial, señale las opciones que no son correctas.
A. $0 \leq Cr(A) \leq 1$
B. $Cr(A) \leq PI(A)$
C. $0 \leq PI(A) \leq 1$
D. Si $Cr(A) = PI(A)$ la incertidumbre asociada a A es total.
E. Si $Cr(A) = PI(A) = 0$ la incertidumbre asociada a A es total.

CONJUNTOS DIFUSOS.
Hablando de conjuntos difusos, Sea A un subconjunto difuso de un referencial cualquiera…
A. Se puede definir una función de pertenencia que le asigna un valor en [0,1] a todo elemento del referencial que pertenece a A.
B. Un elemento del referencial puede pertenecer o no a A.
C. Se puede definir una función de pertenencia que le asigna un valor [0,1] a todo elemento del referencial, según su grado de pertenencia a A.
D. El concepto de referencial no está definido para los conjuntos difusos.
E. Un elemento dado del referencial, o bien pertenece a A, o no pertenece a A, o pertenece en cierto grado a A.

La función de grado de pertenencia de un elemento de un referencial a un subconjunto difuso de ese referencial…
A. Toma valores en [-1,1]
B. Toma valores en (0,1)
C. Toma valores en [0,1]
D. Toma valores en (-1,1)
E. Necesita de un criterio, frecuentemente subjetivo, para ser definida.

Si U es un referencial, x es un elemento del referencial, A es un subconjunto difuso de U, noA es el complementario de A, la ley del tercero excluido dice que:
A. La función de grado de pertenencia de la unión de A y noA es mayor de 0.5 para todo x de U.
B. La función de grado de pertenencia de la unión de A y noA es mayor de 0.5 para todo x de A.
C. La función de grado de pertenencia de la unión de A y noA es mayor o igual que 0.5 para todo x de A.
D. La función de grado de pertenencia de la unión de A y noA es menor o igual que 0.5 para todo x de U.
E. La función de grado de pertenencia de la unión de A y noA es mayor o igual que 0,5 para todo x de U.

Si U es un referencial, x es un elemento del referencial, A es un subconjunto difuso de U, noA es el complementario de A, la ley de no contradicción dice que:
A. La función de grado de pertenencia de la intersección de A y noA es menor o igual que 0.5 para todo x de U.
B. La función de grado de pertenencia de la intersección de A y noA es menor de 0.5 para todo x de U.

C. La función de grado de pertenencia de la intersección de A y noA es menor o igual que 0.5 para todo x de A.
D. La función de grado de pertenencia de la intersección de A y noA es menor de 0.5 para todo x de A.
E. La función de grado de pertenencia de la intersección de A y noA es mayor o igual que 0.5 para todo x de U.

Señale las opciones correctas…
A. Los conjuntos difusos son un caso particular de los conjuntos ordinarios, siempre.
B. Los conjuntos ordinarios son un caso particular de los conjuntos difusos, siempre.
C. Los conjuntos difusos son un caso particular de los conjuntos ordinarios, siempre que la función de grado pertenencia valga 1.
D. Los conjuntos ordinarios son un caso particular de los conjuntos difusos, siempre que la función de grado de pertenencia valga 1.
E. Ninguna de las otras es correcta.

Señala las declaraciones que no son correctas…
A. Los conjuntos difusos no tienen estructura de álgebra de Boole porque no cumplen las leyes de DeMorgan.
B. Los conjuntos difusos no tienen estructura de álgebra de Boole porque no cumplen las leyes distributivas.
C. Los conjuntos difusos no tienen estructura de álgebra de Boole porque no cumplen la ley del tercero excluido.
D. Los conjuntos difusos no tienen estructura de álgebra de Boole porque no cumplen el principio de no contradicción.
E. En los conjuntos difusos la intersección no está definida.

Un conjunto difuso se dice que esta normalizado cuando…
A. Todos los elementos del referencial pertenecen, al menos en cierto grado, al conjunto difuso considerado.
B. Es no vacío.
C. Existe al menos un elemento del referencial que pertenece completamente al conjunto difuso considerado.
D. Existe al menos un elemento del referencial que no pertenece al conjunto difuso considerado.
E. Tiene núcleo.

Señale las declaraciones que son correctas…
A. El núcleo de un conjunto difuso A de U está constituido por todos los elementos de A que pertenecen completamente a U.
B. El núcleo de un conjunto difuso A de U está constituido por todos los elementos de U que pertenecen completamente a A.
C. El conjunto difuso se dice que está normalizado cuando tiene núcleo.
D. El núcleo de un conjunto difuso A de U está constituido por todos los elementos de A que pertenecen al menos en cierto grado a U.
E. El núcleo de un conjunto difuso A de U está constituido por todos los elementos de U que pertenecen al menos en cierto grado a A.

Hablando de lógica difusa… Señale las opciones que crea correctas.
A. En lógica difusa, el razonamiento categórico es un caso particular el razonamiento aproximado.
B. En lógica difusa todo es cuestión de grado.
C. Cualquier sistema lógico convencional puede ser tratado de forma difusa.
D. En lógica difusa el conocimiento no puede ser representado.
E. En lógica difusa los procesos inferenciales no existen.

Si A es un conjunto difuso referencial U, y B es un conjunto difuso del referencia V, con x de U e y de V, la expresión: Si x es A entonces y es B se puede representar del siguiente modo:

A. no A “suma acotada” B en UxV

B. no A “unión” B en UxV

C. no A “intersección” B en UxV

D. no A “producto” B en UxV

E. no A “diferencia absoluta” B en UxV

UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA
Grao en Enxeñería Informática
Sistemas Intelixentes

Boletín de problemas. Aula TGR 2.

Escribir Apellidos, Nombre y grupo de TGR correspondiente

1. *(1 punto)* Para cada una de las siguientes aserciones, decir si es verdadera o falsa justificando la respuesta:

   a) La estrategia de búsqueda que expande el nodo más superficial del conjunto de nodos no explorados, es decir, el que se encuentra al frente de la frontera, es la búsqueda preferente por el mejor.

   b) La búsqueda en profundidad es óptima y completa siempre que el coste de los operadores sea constante

   c) Tanto la complejidad temporal como la complejidad espacial de la búsqueda en amplitud se expresan en función del número de nodos explorados

   d) La complejidad espacial de la búsqueda en amplitud es mayor que en el caso de la búsqueda en profundidad

2. *(1'5 puntos)* Contestar a las siguientes preguntas:

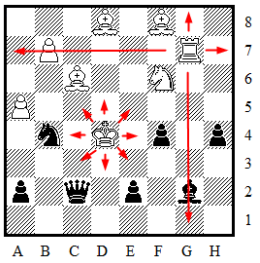
   a) ¿A qué búsqueda es equivalente el algoritmo A* si se utiliza como función de coste g(n)=1 y como función heurística h(n)=0 para cualquier nodo n?

   b) ¿Qué significa que una función heurística h_1 está más informada que otra función heurística h_2? ¿Qué supone para el algoritmo A* utilizar h_1 en lugar de h_2 como función heurística?

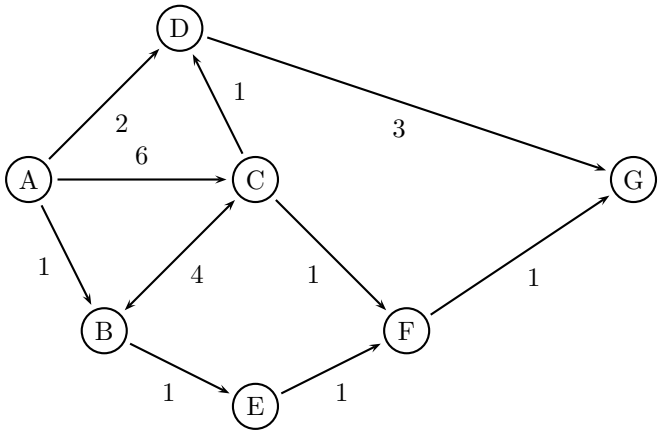
   c) Sea un espacio de estados finito en el que existen varios nodos que contienen estados meta y el coste de los operadores es constante. Sabiendo que utilizaremos el algoritmo de búsqueda general basado en grafo, ¿qué estrategia de búsqueda no informada seleccionaríamos si queremos encontrar una solución? ¿Y si quisiéramos encontrar la solución menos costosa?

3. *(2 puntos)* Consideremos el juego del ajedrez, y dos de sus piezas: la torre y el rey. En cada movimiento, mientras la torre puede desplazarse cualquier número de casillas en línea recta, vertical u horizontalmente, pero nunca saltar sobre otras piezas, el rey sólo puede hacerlo una casilla pero en todas direcciones. En la figura se puede ver un esquema de los movimientos de ambas fichas.

   En este contexto, queremos utilizar un algoritmo de búsqueda informado que encuentre la ruta más corta para los problemas de mover una torre y mover un rey. En cada uno de los dos casos, determinar si la *heurística Manhattan* (medida del número de cuadrículas que atraviesa una pieza realizando movimientos horizontales o verticales) es una heurística admisible y justificar la respuesta.



4. *(3 puntos)* En la figura se muestra un grafo de búsqueda donde cada círculo representa un nodo correspondiente a un estado en el espacio de estados, y los arcos los costes de la acción que permite pasar de un estado a otro. Suponer que el estado inicial se encuentra en el nodo A y la meta en el nodo G. El orden de generación de nuevos nodos será el alfabético, es decir, a partir de A se generan B, C y D en ese orden. Se pide :



a) Escribir la secuencia de generación y expansión de nodos, el camino solución (si se encuentra) y su coste aplicando las estrategias de búsqueda preferente en anchura y en profundidad. Utilizar para ello el algoritmo de búsqueda basada en grafo[1]. La siguiente tabla muestra el formato para presentar los resultados, donde el valor entre paréntesis de cada nodo es el coste del camino:

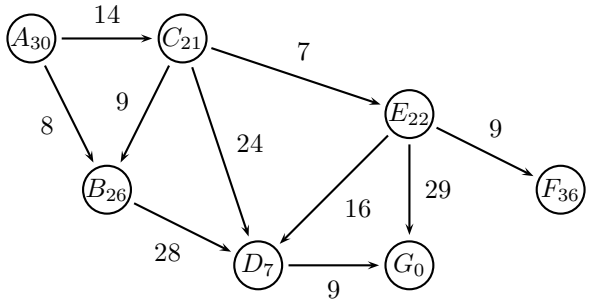
Paso	Frontera	Explorados
1	A(0)	-
2	B(1),C(6),D(2)	A(0)

b) ¿Cuantos nodos genera el algoritmo de búsqueda en anchura? ¿Y profundidad?

c) ¿Qué algoritmo encuentra la solución óptima? ¿Por qué?

5. *(2'5 puntos)* En la siguiente figura se muestra un grafo de búsqueda del espacio de estados de un problema. Las etiquetas de los nodos representan un estado, y el subíndice de dichas etiquetas el valor de la *función heurística h*. Los arcos que interconectan los nodos representan la acción que permite pasar de un estado a otro y la etiqueta el coste de dicha transición. Suponer que el estado inicial se encuentra en el nodo A y la meta en el nodo G. Escribir la secuencia de generación (FRONTERA) y expansión de nodos (EXPLORADOS) y el camino solución (si se encuentra) aplicando el algoritmo de búsqueda A* basado en grafo. Utilizar la siguiente tabla como modelo para presentar los resultados, donde cada nodo de la columna FRONTERA se acompaña del valor de la función de evaluación (el primer sumando entre paréntesis corresponde al valor de g y el segundo al valor de h) y cada nodo de la columna EXPLORADOS se acompaña sólo del valor de g:

Paso	Frontera	Explorados
1	A(0+30)	-
2	B(8+26),C(14+21)	A(0)



Normas de entrega:

- La entrega de los boletines propuestos es OPCIONAL.

[1]La frontera es una cola FIFO en el primer caso y LIFO en el segundo

- La realización será individual.
- La nota asociada a las tareas de Tutorías de Grupo Reducido se mantiene para la oportunidad de JULIO ya que **no existe la posibilidad de una nueva entrega**.
- No se admitirán los trabajos entregados fuera del plazo indicado
- Forma de entrega: electrónico a través del Campus virtual.

Fecha límite de entrega: jueves 13 de marzo de 2014 (23:55 horas).

BOLETÍN DE PROBLEMAS TGR 2

Breixo Camiña Fernández – 2.3.1

Ejercicio 1

Para cada una de las siguientes aserciones, decir si es verdadera o falsa justificando la respuesta:

a) La estrategia de búsqueda que expande el nodo más superficial del conjunto de nodos no explorados, es decir, el que se encuentra al frente de la frontera, es la búsqueda preferente por el mejor.

La aserción es falsa porque el nodo a expandir en la búsqueda preferente por el mejor se selecciona en base al menor (mejor) valor de la función de evaluación f(n).

b) La búsqueda en profundidad es óptima y completa siempre que el coste de los operadores sea constante.

La búsqueda en profundidad es completa si usamos búsqueda basada en grafos en espacios de estado finitos porque expandirá todos los nodos pero no es óptima porque puede encontrar una solución más profunda que otra en una rama no expandida, por lo tanto la sentencia es falsa

c) Tanto la complejidad temporal como la complejidad espacial de la búsqueda en amplitud se expresan en función del número de nodos explorados.

Las complejidades temporal ($O(b^d)$) y espacial ($O(b^d)$) están basados en función del número de nodos expandidos, es decir, que están en la frontera. En complejidad espacial cada nodo generado permanece en memoria por lo que al nivel del nodo meta se almacena la mayor cantidad de nodos $O(b^d)$ para los nodos en la frontera mientras que en la complejidad temporal si la meta se encuentra a una profundidad d, hay que generar todos los nodos de ese nivel.

d) La complejidad espacial de la búsqueda en amplitud es mayor que en el caso de la búsqueda en profundidad.

Verdadero, ya que para la búsqueda en amplitud, en la complejidad espacial cada nodo generado permanece en memoria por lo que hasta el nodo meta se almacena la mayor cantidad de nodos $O(b^d)$ y para la búsqueda en profundidad, el mayor número de nodos almacenados se alcanza en el nodo inferior de más a la izquierda, recorriendo solo un camino por lo que sería O(bm).

Ejercicio 2

Contestar a las siguientes preguntas:

a) **¿A qué búsqueda es equivalente el algoritmo A* si se utiliza como función de coste g(n) = 1 y como función heurística h(n)=0 para cualquier nodo n?**
Si para todo nodo del grafo se cumple h(n)=0 y g(n)=1 la búsqueda sería de coste uniforme, presumiblemente no informada.

b) **¿Qué significa que una función heurística h_1 está más informada que otra función heurística h_2? ¿Qué supone para el algoritmo A* utilizar h_1 en lugar de h_2 como función heurística?**
Que la función heurística h1 realiza su trabajo y alcanza el nodo meta de forma más eficiente que la función h2. Que el número de nodos expandido sería menor, al emplear una función mejor para hallar la meta.

c) **Sea un espacio de estados finito en el que existen varios nodos que contienen estados meta y el coste de los operadores es constante. Sabiendo que utilizaremos el algoritmo de búsqueda general basado en grafo, ¿qué estrategia de búsqueda no informada seleccionaríamos si queremos encontrar una solución? ¿Y si quisiéramos encontrar la solución menos costosa?**
Para hallar una solución cualquiera usaríamos la búsqueda preferente en profundidad porque en esta búsqueda siempre se expande uno de los nodos que se encuentre en lo más profundo del árbol pero si la búsqueda conduce a un callejón sin salida (nodo que no es meta y sin expansión), se revierte la búsqueda y se expanden los nodos menos profundos.

Para hallar la solución menos costosa, emplearíamos la búsqueda en profundidad iterativa porque es equivalente a la búsqueda en anchura pero usa mucha menos memoria: visita los nodos del árbol de búsqueda en el mismo orden que una búsqueda en profundidad pero el orden de visita se corresponde con la búsqueda en anchura, recorriendo por lo tanto un menor número de nodos para hallar la solución.

Ejercicio 3

Consideremos el juego del ajedrez, y dos de sus piezas: la torre y el rey. En cada movimiento, mientras la torre puede desplazarse cualquier número de casillas en línea recta, vertical u horizontalmente, pero nunca saltar sobre otras piezas, el rey sólo puede hacerlo una casilla pero en todas direcciones. En la figura se puede ver un esquema de los movimientos de ambas fichas. En este contexto, queremos utilizar un algoritmo de búsqueda informado que encuentre la ruta más corta para los problemas de mover una torre y mover un rey. En cada uno de los dos casos, determinar si la heurística Manhattan (medida del número de cuadrículas que atraviesa una pieza realizando movimientos horizontales o verticales) es una heurística admisible y justificar la respuesta.

Para el rey la heurística de Manhattan podría ser admisible porque puede atravesar una única cuadrícula por movimiento, aunque si tenemos en cuenta los movimientos en diagonal, tendría que ser dos (la heurística real del Rey sería la de Chebyshev, al igual que la de la reina).

Para la torre, la función heurística no sería correcta porque puede variar según el número de casillas que se mueva la pieza en cada turno y por lo tanto habría que aplicar uno o incluso dos veces dicha función.

Ejercicio 4

En la figura se muestra un grafo de búsqueda donde cada círculo representa un nodo correspondiente a un estado en el espacio de estados, y los arcos los costes de la acción que permite pasar de un estado a otro. Suponer que el estado inicial se encuentra en el nodo A y la meta en el nodo G. El orden de generación de nuevos nodos será el alfabético, es decir, a partir de A se generan B, C y D en ese orden. Se pide:

a) Escribir la secuencia de generación y expansión de nodos, el camino solución (si se encuentra) y su coste aplicando las estrategias de búsqueda preferente en anchura y en profundidad. Utilizar para ello el algoritmo de búsqueda basada en grafo. La siguiente tabla muestra el formato para presentar los resultados, donde el valor entre paréntesis de cada nodo es el coste del camino:

ANCHURA

PASO	FRONTERA	EXPLORADOS
1	A(0)	-
2	B(1), C(6), D(2)	A(0)
3	C(6), D(2), E(2)	A(0), B(1)
4	D(2), E(2), F(7)	A(0), B(1), C(6)
5	E(2), F(7), G(5)	A(0), B(1), C(6), D(2)

Camino solución: A – D – G

PROFUNDIDAD

PASO	FRONTERA	EXPLORADOS
1	A(0)	-
2	B(1), C(6), D(2)	A(0)
3	E(2), C(6), D(2)	A(0), B(1)
4	F(3), C(6), D(2)	A(0), B(1), E(2)
5	G(4), C(6), D(2)	A(0), B(1), E(2), F(3)

Camino solución: A – B – E – F – G

b) ¿Cuantos nodos genera el algoritmo de búsqueda en anchura? ¿Y profundidad?

En anchura se generarían todos los nodos, es decir, el número de nodos generados es 7 mientras que en profundidad también se generarían los 7 nodos.

c) ¿Qué algoritmo encuentra la solución óptima? ¿Por qué?

A pesar de que en la búsqueda en profundidad el coste es menor, la solución óptima sería la búsqueda en anchura porque se recorren menos nodos para llegar a la solución final.

Ejercicio 5

En la siguiente figura se muestra un grafo de búsqueda del espacio de estados de un problema. Las etiquetas de los nodos representan un estado, y el subíndice de dichas etiquetas el valor de la función heurística h. Los arcos que interconectan los nodos representan la acción que permite pasar de un estado a otro y la etiqueta el coste de dicha transición. Suponer que el estado inicial se encuentra en el nodo A y la meta en el nodo G. Escribir la secuencia de generación (FRONTERA) y expansión de nodos (EXPLORADOS) y el camino solución (si se encuentra) aplicando el algoritmo de búsqueda A* basado en grafo. Utilizar la siguiente tabla como modelo para presentar los resultados, donde cada nodo de la columna FRONTERA se acompaña del valor de la función de evaluación (el primer sumando entre paréntesis corresponde al valor de g y el segundo al valor de h) y cada nodo de la columna EXPLORADOS se acompaña sólo del valor de g:

PASO	FRONTERA	EXPLORADOS
1	A(0)	-
2	B(8+26), C(14+21)	A(0)
3	C(14+21), D(36+7)	A(0), B(8)
4	D(36+7), E(21+22) *	A(0), B(8), C(14)
5	E(21+22), G(45+0)	A(0), B(8), C(14), D(36)
6	G(45+0), F(30+36)	A(0), B(8), C(14), D(36), E(21)
7	F(30+36)	A(0), B(8), C(14), D(36), E(21), G(45)

*Elijo el nodo por menor orden alfabético.

Camino solución: A - B - D - G

UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA
Grao en Enxeñería Informática
Sistemas Intelixentes
Ejercicio TGR 3. Marzo 2014

Modelos de razonamiento: razonamiento categórico y corrección bayesiana

1. (*10 puntos*) Dos pacientes acuden a urgencias del hospital la noche del sábado. Uno aquejado de *tos*—$m(3)$—y *diarreas* frecuentes—$m(2)$. El médico de guardia comprueba que no tiene *fiebre alta*—$m(1)$. El otro, manifiesta fiebre alta y tos pero no tiene diarrea. A la vista de únicamente estos síntomas, el médico debe discernir entre dos posibilidades para establecer un tratamiento adecuado: (1) El paciente tiene el virus de la *gripe*—$i(1)$—, que se manifiesta a través de fiebre alta acompañada de tos; o (2) el paciente tiene una *gastroenteritis*—$i(2)$— pero no *gripe*—$i(1)$—, lo cual se manifiesta por diarreas frecuentes. Y además, la manifestación de cualquiera de los tres síntomas, indicaría la presencia de alguno de los diagnósticos posibles. Se pide:

   (a) Aplicar el procedimiento sistemático para el razonamiento categórico y obtener el conjunto de interpretaciones compatible con las manifestaciones para ambos pacientes.

   (b) Si una estadística fiable nos ha aportado la siguiente información:

$$\begin{array}{lll} P(m1/i1) = 1,0 & P(m7/i2) = 0,2 & P(m8/i3) = 0,9 \\ P(m3/i2) = 0,4 & P(m8/i2) = 0,1 & P(m8/i4) = 0,9 \\ P(m4/i2) = 0,3 & P(m6/i3) = 0,1 & P(m6/i4) = 0,1 \end{array}$$

   y también se sabe que en una muestra suficientemente amplia y representativa:

$$\begin{array}{c} \neg I(1) \wedge \neg I(2) \text{ representan el } 45\% \text{ de los casos} \\ \neg I(1) \wedge I(2) \text{ representan el } 30\% \text{ de los casos} \\ I(1) \wedge \neg I(2) \text{ representan el } 20\% \text{ de los casos} \\ I(1) \wedge I(2) \text{ representan el } 5\% \text{ de los casos} \end{array}$$

   aplicar el esquema bayesiano para mejorar los resultados obtenidos a partir del modelo categórico, decidiendo qué interpretación sería más probable en el caso que el paciente padezca fiebre alta y tos.

   En la solución detallar:

   1. El conjunto completo de complejos manifestaciones y de interpretaciones
   2. La función del conocimiento en base a la información proporcionada
   3. La base lógica expandida (BLE)
   4. La tabla de complejos manifestación-interpretación posible o BLR
   5. El conjunto de interpretaciones compatible con las manifestaciones para el primer y segundo paciente

   Para resolver el caso planteado utilizar el siguiente criterio:

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8
m(1)	0	0	0	0	1	1	1	1
m(2)	0	0	1	1	0	0	1	1
m(3)	0	1	0	1	0	1	0	1

	i_1	i_2	i_3	i_4
i(1)	0	0	1	1
i(2)	0	1	0	1

TGR3 - Modelos de razonamiento: razonamiento categórico y corrección bayesiana

Breixo Camiña Fernández – 2.3.1

Ejercicio 1

Dos pacientes acuden a urgencias del hospital la noche del sábado. Uno aquejado de tos (m(3)) y diarreas frecuentes (m(2)). El médico de guardia comprueba que no tiene fiebre alta (m(1)). El otro, manifiesta fiebre alta y tos pero no tiene diarrea. A la vista de únicamente estos síntomas, el médico debe discernir entre dos posibilidades para establecer un tratamiento adecuado: (1) El paciente tiene el virus de la gripe (i(1)), que se manifiesta a través de fiebre alta acompañada de tos; o (2) el paciente tiene una gastroenteritis (i(2) pero no gripe (i(1)), lo cual se manifiesta por diarreas frecuentes. Y además, la manifestación de cualquiera de los tres síntomas, indicaría la presencia de alguno de los diagnósticos posibles. Se pide:

a) **Aplicar el procedimiento sistemático para el razonamiento categórico y obtener el conjunto de interpretaciones compatible con las manifestaciones para ambos pacientes.**

1 – El conjunto completo de complejos manifestaciones y de interpretaciones

Manifestaciones

Fiebre alta: m(1)
Diarreas: m(2)
Tos: m(3)

Interpretaciones

Gripe: i(1)
Gastroenteritis pero no gripe: i(2) Λ ¬ i(1)
Cualquier manifestación: i(1) V i(2)

Pacientes

Paciente 1: m(3) Λ m(2) Λ ¬m(1)
Paciente 2: m(1) Λ m(3) Λ ¬m(2)

Relaciones casuales

Para que haya i(1), debe estar presente m(1) y m(3).
Para que haya i(3), debe estar presente m(2).
Si alguna manifestación (m(1) V m(2) V m(3)) está presente es porque se puede establecer alguna interpretación (i(1) V i(2)).

Reglas

R1: i(1) -> m(1) Λ m(3)
R2: i(2) Λ ¬i(1) -> m(2)
R3: i(1) V i(2) -> m(1) V m(2) V m(3)

2 – La función del conocimiento en base a la información proporcionada

E = {R1, R2, R3}

3 - Base Lógica Expandida (BLE)

{m1i1, m1i2, m1i3, m1i4,
m2i1, m2i2, m2i3, m2i4,
m3i1, m3i2, m3i3, m3i4,
m4i1, m4i2, m4i3, m4i4,
m5i1, m5i2, m5i3, m5i4,
m6i1, m6i2, m6i3, m6i4,
m7i1, m7i2, m7i3, m7i4,
m8i1, m8i2, m8i3, m8i4}

	m1	m2	m3	m4	m5	m6	m7	m8	m1	m2	m3	m4	m5	m6	m7	m8
m(1)	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
m(2)	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
m(3)	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
i(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i(2)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	i1								i2							

	m1	m2	m3	m4	m5	m6	m7	m8	m1	m2	m3	m4	m5	m6	m7	m8
m(1)	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
m(2)	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
m(3)	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
i(1)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
i(2)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	i3								i4							

La regla R1 elimina los complejos m1i3, m2i3, m3i3, m4i3, m5i3, m7i3, m1i4, m2i4, m3i4, m4i4, m5i4 y m7i4 (color azul) porque, con independencia de lo que pueda pasar con i(2), para cumplirse i(1), m(1) y m(3) deben estar presentes.

La regla R2 elimina los complejos m1i2, m2i2, m5i2 y m6i2 (color naranja).

La regla R3 elimina los complejos m2i1, m3i1, m4i1, m5i1, m6i1, m7i1, m8i1, m1i2, m1i3, m1i4 (estos tres últimos ya eliminados previamente) (color verde).

4 – La tabla de complejos manifestación-interpretación posible o BLR

La aplicación del conocimiento del dominio (función E) sobre BLE se traduce en la siguiente Base Lógica Reducida:

BLR = {m1i1, m3i2, m4i2, m7i2, m8i2, m6i3, m8i3, m6i4, m8i4}

5 – El conjunto de interpretaciones compatible con las manifestaciones para el primer y segundo paciente

Para el paciente 1 se cumple el complejo m4i2, teniendo entonces gastroenteritis i(2).

Para el paciente 2 se cumplen los complejos m6i3 y m6i4, por lo tanto puede tener tanto gripe i(1) como gastroenteritis i(2).

b) Si una estadística fiable nos ha aportado la siguiente información:

P(m1/i1) = 1,0	**P(m3/i2) = 0,4**	**P(m4/i2) = 0,3**
P(m7/i2) = 0,2	**P(m8/i2) = 0,1**	**P(m6/i3) = 0,1**
P(m8/i3) = 0,9	**P(m8/i4) = 0,9**	**P(m6/i4) = 0,1**

y también se sabe que en una muestra suficientemente amplia y representativa:

¬I(1) ^ ¬I(2) representan el 45% de los casos
¬I(1) ^ I(2) representan el 30% de los casos
I(1) ^ ¬I(2) representan el 20% de los casos
I(1) ^ I(2) representan el 5% de los casos

aplicar el esquema bayesiano para mejorar los resultados obtenidos a partir del modelo categórico, decidiendo que interpretación sería más probable en el caso que el paciente padezca fiebre alta y tos.

$$P(i3/m6) = \frac{P(m6/i3)*P(i3)}{P(m6/i3)*P(i3)+P(m6/i4)*P(i4)} = \frac{0.1*0.2}{0.1*0.2+0.1*0.05} = 0.8$$

$$P(i4/m6) = \frac{P(m6/i4)*P(i4)}{P(m6/i4)*P(i4)+P(m6/i3)*P(i3)} = \frac{0.1*0.05}{0.1*0.05+0.1*0.2} = 0.2$$

Por lo tanto, como P(i3/m6) es mayor que P(i4/m6), el paciente 2 tiene más posibilidades de tener gripe i(1).