INTELIGENCIA ARTIFICIAL

UNIVERSIDAD DE GRANADA

2022 - 2023

Índice

Carmen Chunyin Fernández Núñez

a
6
4
b
0
4
6
9
f
f
3
5
9
5
8
e
f
4
a
b
8
8
7
a
8
9
8
b
d
5
0
b

Introducción a la Inteligencia Artificial

1. DEFINICIÓN DE INTELIGENCIA

Según la R.A.E hay siete definiciones diferentes de inteligencia:

1. Capacidad de entender o comprender.
2. Capacidad de resolver problemas.
3. Conocimiento, comprensión, acto de entender.
4. Sentido en que se puede tomar una sentencia, un dicho o una expresión.
5. Habilidad, destreza y experiencia.
6. Trato y correspondencia secreta de dos o más personas o naciones entre sí.
7. Sustancia puramente espiritual.

"La inteligencia es la capacidad de ordenar los pensamientos y coordinarlos con las acciones. La inteligencia no es una sola, sino que existen tipos distintos"

Howard Gardner, Universidad de Harvard

1.2 TEORÍA DE LAS INTELIGENCIAS MÚLTIPLES DE H. GARDNER

Es conocido fundamentalmente por su **teoría de las inteligencias múltiples**, que señala que no existe una inteligencia única en el ser humano, sino una diversidad de inteligencias que marcan las potencialidades y acentos significativos de cada individuo, trazados por las fortalezas y debilidades en toda una serie de escenarios de expansión de la inteligencia.

Inteligencia lingüística. En los niños y niñas se aprecia en su facilidad para escribir, leer, contar cuentos o hacer crucigramas.

Inteligencia Lógica-matemática. Se aprecia en los menores por su interés en patrones de medida, categorías y relaciones. Facilidad para la resolución de problemas aritméticos, juegos de estrategia y experimentos.

Inteligencia Corporal y Cinética. Facilidad para procesar el conocimiento a través de las sensaciones corporales. Deportistas, bailarines o manualidades como la costura, los trabajos en madera, etc.

Inteligencia Visual y espacial. Los niños y niñas piensan en imágenes y dibujos. Tienen facilidad para resolver rompecabezas, dedican el tiempo libre a dibujar, prefieren juegos constructivos, etc.

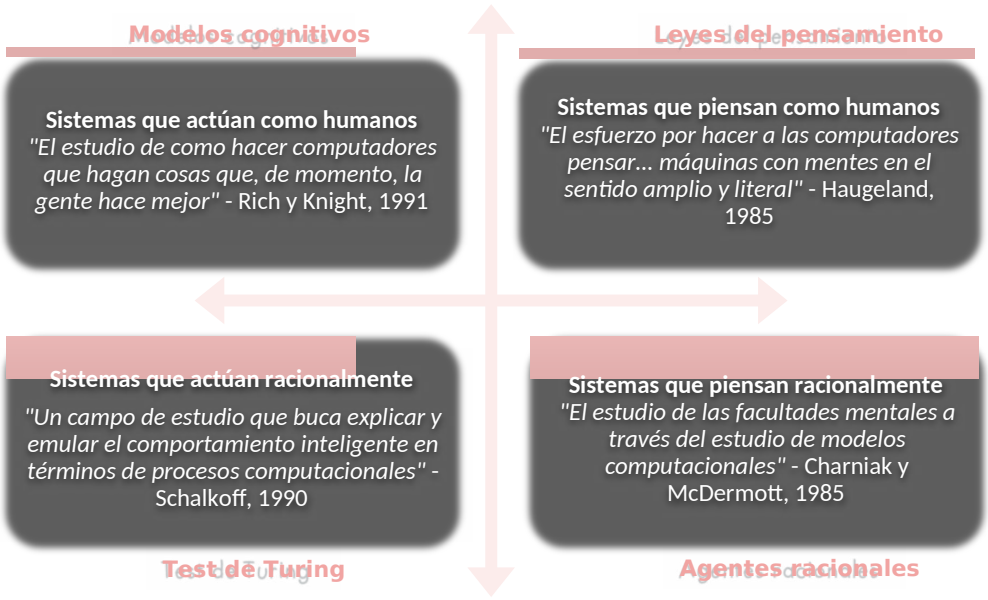
Inteligencia Musical. Los menores se manifiestan frecuentemente con canciones y sonidos. Identifican con facilidad los sonidos.

Inteligencia Interpersonal (inteligencia social). Se comunican bien y son líderes en sus grupos. Entienden bien los sentimientos de los demás y proyectan con facilidad las relaciones interpersonales.

Inteligencia Intrapersonal. Relacionada con la capacidad de un sujeto de conocerse a sí mismo: sus reacciones, emociones y vida interior.

Inteligencia naturalista. Relacionada con la facilidad de comunicación con la naturaleza; que consiste en el entendimiento del entorno natural y la observación científica de la naturaleza como la biología, geología o astronomía

2. DEFINICIÓN DE INTELIGENCIA ARTIFICIAL



SISTEMAS QUE PIENSAS COMO HUMANOS

El modelo es el funcionamiento de la mente humana. Intentamos establecer una teoría sobre el funcionamiento de la mente (experimentación psicológica). A partir de la teoría podemos establecer modelos computacionales **(Ciencias cognitivas)**

SISTEMAS QUE PIENSAN RACIONALMENTE

Las leyes del pensamiento racional se fundamentan en la lógica. La lógica formal está en la base de los programas inteligentes **(Logicismo).** Se presentan dos obstáculos:

- Es muy difícil formalizar el conocimiento.
- Hay un gran salto entre la capacidad teórica de la lógica y su realización práctica.

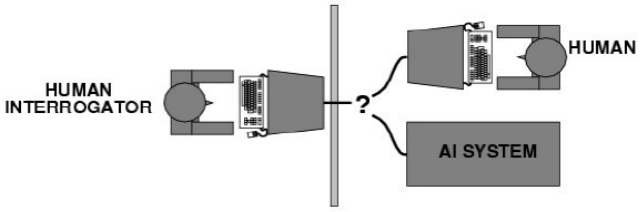
SISTEMAS QUE ACTÚAN COMO HUMANOS

El modelo es el hombre, el objetivo es construir un sistema que pase por humano. Capacidades necesarias:

- Procesamiento del Lenguaje Natural
- Representación del conocimiento
- Razonamiento
- Aprendizaje.

La interacción de programas con personas hace que sea necesario que estos actúen como humanos

Test de Turing Conducta Inteligente: la capacidad de lograr eficiencia a nivel humano en todas las actividades de tipo cognoscitivo, suficiente para engañar a un evaluador



SISTEMAS QUE ACTÚAN RACIONALMENTE

Actuar racionalmente significa conseguir unos objetivos dadas unas creencias. El paradigma es el **agente**. Un agente percibe y actúa, siempre según el entorno en el que está situado. Un **agente racional** actúa de la manera **correcta según la información que posee.** Las capacidades necesarias coinciden con las del **test de Turing**. Su visión es más general, no centrada en el modelo humano.

La Inteligencia Artificial es una rama de la Informática que estudia y resuelve problemas situados en la frontera de la misma. Se basa en dos ideas fundamentales:

- Representación del conocimiento explícita y declarativa
- Resolución de problemas (heurística)

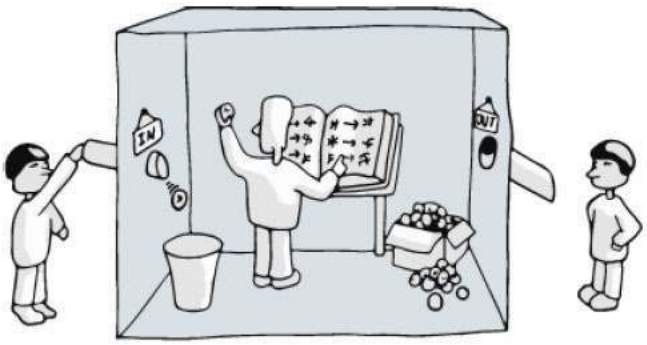
La posibilidad de la inteligencia artificial plantea problemas filosóficos complejos.

- ¿Las maquinas pensantes poseen consciencia?
- ¿Es la inteligencia una propiedad emergente de los elementos biológicos que la producen?
- No hay una conclusión definitiva.

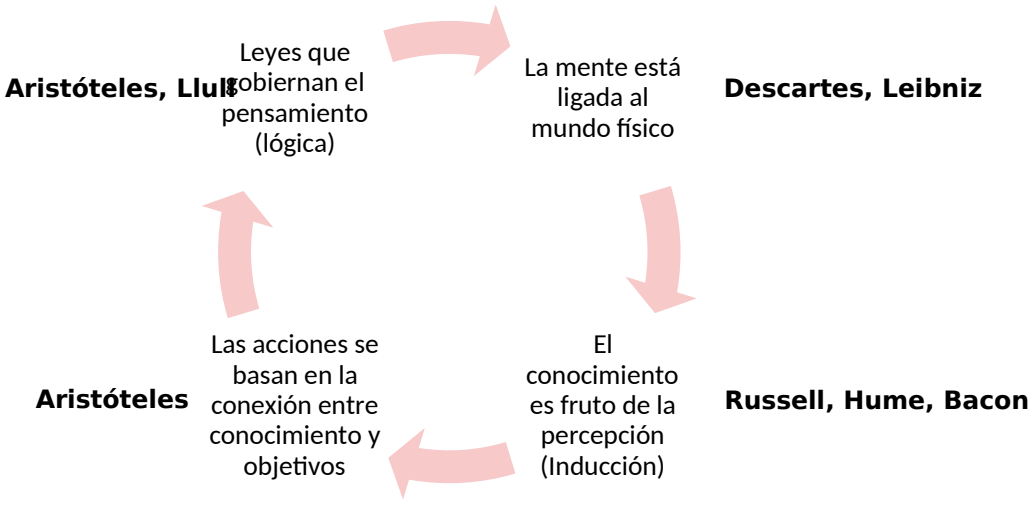
LA HABITACIÓN CHINA DE SEARLE

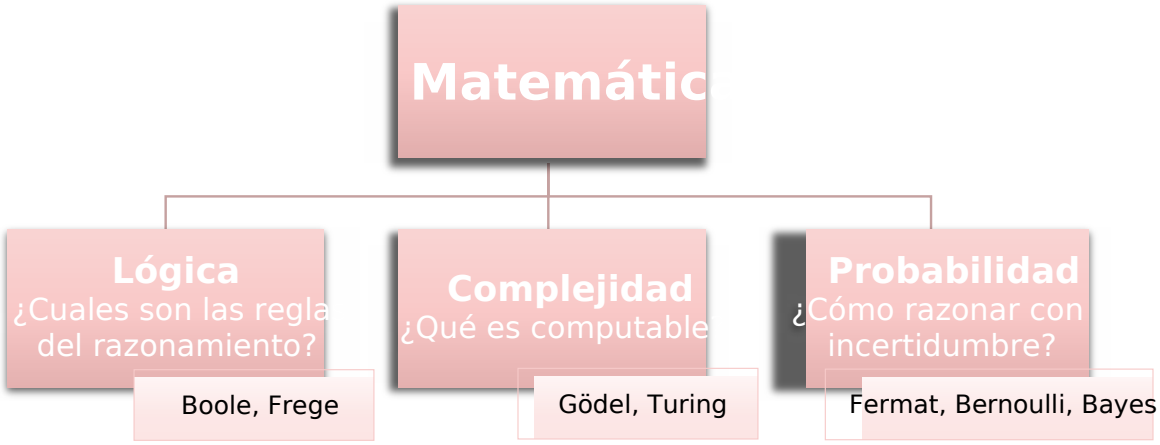
En una habitación cerrada, con un orificio de entrada y uno de salida, se coloca a un sujeto con un diccionario de chino. Cada vez que el sujeto recibe un documento en chino por la entrada, lo traduce y devuelve el documento resultante por la salida.

Para el que no conozca el sistema, este en su conjunto "sabe chino", pero realmente el sujeto sabe chino?



2.1 BASES DE LA INTELIGENCIA ARTIFICAL





Computación

- Para la existencia de la IA es necesario un mecanismo para soportarlo
- (Hardware)
- También son necesarias herramientas para desarrollar programas de IA

Teoría de control/Cibernética

- Construcción de sistemas autónomos

Lingüística

- Chomsky: Representación del conocimiento, gramática de la lengua
- Lingüística computacional

3. HISTORIA DE LA INTELIGENCIA ARTIFICIAL

Período de gestación (1943-1955): Se desarrollan los primeros modelos neuronales artificiales que simulan una neurona biológica (McCulloch y Pitts, 1943).

Nacimiento (1956): Conferencia Dartmouth, se perfila la disciplina de la **Inteligencia Artificial** cuyo objetivo es duplicar facultades humanas como creatividad, automejora, uso del lenguaje, etc.

Entusiasmo inicial, grandes expectativas (1952-1969): General Problem Solver, hipótesis de sistema de símbolos físicos, Geometry Problem Solver, Advice Taker, mundo de los bloques, etc.

Una dosis de realidad (1966-1973): Se encuentran dificultades debido al gran conocimiento general necesario para resolver problemas específicos y la intratabilidad de algunos problemas.

Sistemas Expertos(1969-1986) Se desarrollan los primeros sistemas expertos (DRENDAL para reconocer moléculas, MYCIN para diagnóstico médico, SHRDLU para entender el lenguaje natural, desarrollo de LISP y Prolog, etc.)

I.A. en la industria (1980-actualidad): Control difuso, diseño de chips, interfaces hombre-máquina, algoritmos heurísticos, resolución de problemas de logística, etc.

Nueva era de las redes neuronales artificiales (1986-actualidad)

Razonamiento probabilístico y aprendizaje (1987-actualidad)

Big Data (2011-actualidad)

Deep learning(2011-actualidad)

4. ÉTICA DE LA INTELIGENCIA ARTIFICIAL

Hoy en día, la inteligencia artificial (IA) desempeña un papel en la vida de miles de millones de personas. A veces inadvertida, pero a menudo con profundas consecuencias, transforma nuestras sociedades y desafía lo que significa ser humano.

¿Qué necesitamos?

- Políticas internacionales y nacionales, así como marcos regulatorios para garantizar que estas tecnologías emergentes beneficien a la humanidad en su conjunto.
- Una IA centrada en el ser humano. La IA debe estar al servicio de los intereses de los ciudadanos, y no al revés.

Agentes

1. AGENTES INTELIGENTE

Los humanos resolvemos problemas en todos los ámbitos de la vida. Algunos de esos problemas son de logística, planificación de tareas, resolver puzles. La IA se inspira en la **ciencia cognitiva, la psicología cognitiva y la neurociencia** para saber qué procesos se llevan a cabo en nuestras mentes a la hora de resolver dichos problemas.

Inteligencia Artificial: subcampo de la Informática dedicado a la construcción de agentes que exhiben aspectos del comportamiento inteligente. Los agentes permiten dar una nueva forma de mostrar la Inteligencia Artificial.

Un **Agente Inteligente** es un sistema de ordenador, **situado** en algún entorno, que es capaz de realizar acciones de forma **autónoma** y que es **flexible** para lograr los objetivos planteados.

- **Situación:** el agente recibe entradas sensoriales de un entorno en donde está situado y realiza acciones que cambian dicho entorno.
- **Autonomía:** el sistema es capaz de actuar sin la intervención directa de los humanos y tiene control sobre sus propias acciones y estado interno.

FLEXIBILIDAD

Reactivo: el agente debe percibir el entorno y responder de una forma temporal a los cambios que ocurren en dicho entorno

Pro-activo: los agentes no deben simplemente actuar en respuesta a su entorno, deben de ser capaces de exhibir comportamientos dirigidos a lograr objetivos que sean oportunos, y tomar la iniciativa cuando sea apropiado

Social: los agentes deben de ser capaces de interactuar, cuando sea apropiado, con otros agentes artificiales o humanos para completar su propio proceso de resolución del problema y ayudar a otros con sus actividades

1.1 TIPOS DE ENTORNOS

Es conveniente identificar algunas características de los entornos que determinan el agente apropiado a utilizar, entre ellas tenemos:

- **Completamente observable.** Disponemos de sensores que detectan toda la información relevante en un estado para tomar una decisión. Puzles, Juegos.
- **Parcialmente observable.** Disponemos de información parcial. Un planificador de rutas sin información de carreteras cortadas.
- **Determinista**. El estado siguiente a la ejecución de una acción podemos determinarlo siempre.
- **No determinista**. No podemos predecir con total certeza lo que puede ocurrir después de ejecutar una acción.
- **Estático**. Podemos tener todo el tiempo que queramos para encontrar solución.
- **Dinámico**. Tenemos que tomar una decisión de actuación rápido.
- **Discreto**. Conjunto finito de estados. Acciones en intervalos discretos de tiempo.
- **Continuo**. Estados continuos (velocidad, posición). Acciones continuas (ángulo de giro, velocidad de giro)
- **Conocido**. Conocemos todos los aspectos del mundo y su dinámica
- **(Parcialmente) desconocido.** Desconocemos (todos) los resultados de las acciones. Explorar y aprender.

1.2 SISTEMAS BASADOS EN AGENTES

Un Sistema Basado en Agentes será un sistema en el que la abstracción clave utilizada es precisamente la de agente

Sistemas multi-agente: un sistema diseñado e implementado con varios agentes interactuando

Los sistemas multi-agente son interesantes para representar problemas que tienen

- múltiples formas de ser resueltos
- múltiples perspectivas
- múltiples entidades para resolver el problema

INTERACCIÓN ENTRE AGENTES

Cooperación: trabajar juntos para resolver algo

Coordinación: organizar una actividad para evitar las interacciones perjudiciales y explotar las beneficiosas

Negociación: llegar a un acuerdo que sea aceptable por todas las partes implicadas

1.2.1 SISTEMAS MULTI-AGENTE. INTELIGENCIA ARTIFICIAL DISTRIBUIDA

SMA: una red más o menos unida de resolutores de problemas que trabajan conjuntamente para resolver problemas que están más allá de las capacidades individuales o del conocimiento de cada resolutor del problema

Resoluto≠agente (autónomo y de naturaleza heterogénea)

Características:

- Cada agente tiene información incompleta, o no todas las capacidades para resolver el problema, así cada agente tiene un punto de vista limitado.
- No hay un sistema de control global.
- Los datos no están centralizados.
- La computación es asíncrona.

Cooperación: herramienta fundamental en la formación de equipos

Negociación: coordinación y resolución de conflictos

2. ARQUITECTURA DE AGENTES

2.1 ARQUITECTURAS DELIBERATIVAS

Sistema de símbolos físicos: un conjunto de entidades físicas (símbolos) que pueden combinarse para formar estructuras, y que es capaz de ejecutar procesos que operan con dichos símbolos de acuerdo a conjuntos de instrucciones codificadas simbólicamente

La **hipótesis de sistema de símbolos físicos** dice que tales sistemas son capaces de generar acciones inteligentes

Agente deliberativo: aquel que contiene un **modelo simbólico del mundo explícitamente representado**, cuyas decisiones se realizan a través de un **razonamiento lógico** basado en **emparejamientos de patrones** y **manipulaciones simbólicas**

El problema de trasladar en un tiempo razonable para que sea útil el mundo real en una descripción simbólica precisa y adecuada

El problema de representar simbólicamente la información acerca de entidades y procesos complejos del mundo real, y como conseguir que los agentes razonen con esta información para que los resultados sean útiles

2.2 ARQUITECTURAS REACTIVAS

Una arquitectura reactiva es aquella que no incluye ninguna clase de modelo centralizado de representación simbólica del mundo, y no hace uso de razonamiento complejo.

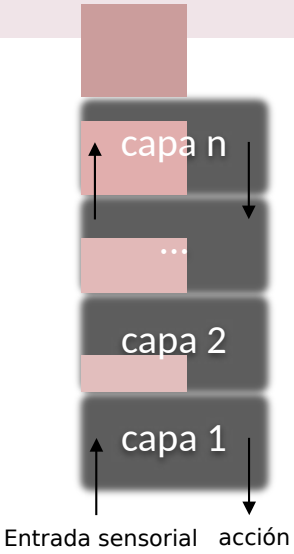
El comportamiento inteligente puede ser generado sin una representación explícita ni un razonamiento abstracto explícito de la clase que la IA simbólica propone.

La inteligencia es una propiedad emergente de ciertos sistemas complejos

El comportamiento "inteligente" surge como el resultado de la interacción del agente con su entorno.

2.3 ARQUITECTURAS HÍBRIDAS

Tienen estructura vertical.



3. AGENTES REACTIVOS

3.1 DISEÑO DE UN AGENTE REACTIVO: ARQUITECTURAS DE AGENTES

3.1.1 PERCEPCIÓN Y ACCIÓN

El agente reactivo percibe su entorno a través de sensores.

Procesa la información percibida y hace una representación interna de la misma.

Escoge una acción, entre las posibles, considerando la información percibida.

Transforma la acción en señales para los actuadores y la realiza.

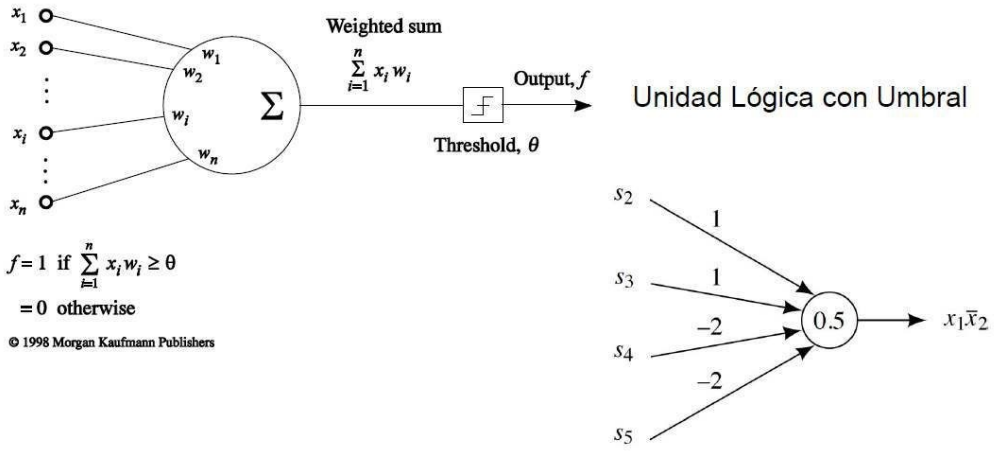
3.2 ARQUITECTURAS DE AGENTES REACTIVOS

3.2.1 SISTEMAS DE PRODUCCIÓN

en donde Ci es una función booleana definida sobre el vect
características, habitualmente una conjunción de literales boolea

$c_1 \rightarrow a_1$
$c_2 \rightarrow a_2$
...
$c_i \rightarrow a_i$
...
$c_m \rightarrow a_m$

3.2.2 REDES

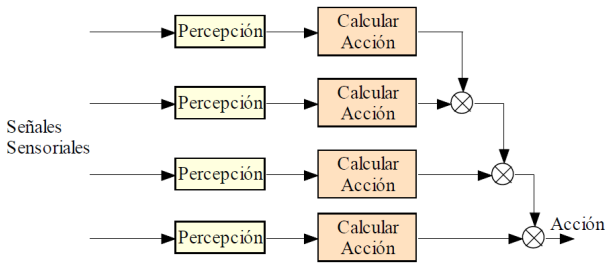


3.2.3 ARQUITECTURA DE SUBSUNCIÓN

La arquitectura de subsunción consiste en agrupar **módulos de comportami**

Cada módulo de comportamiento tie una acción asociada, recibe la percepción directamente y compruet una condición. Si esta se cumple, el módulo devuelve la acción a realizar.

Un módulo se puede subsumir en otro. el módulo superior del esquema se cumple, se ejecuta este en lugar de los módulos inferiores.

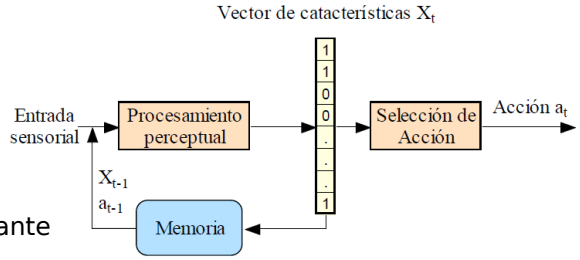


3.3 AGENTES REACTIVOS CON MEMORIA

Limitaciones del sistema sensorial de un agente.

Mejorar la precisión teniendo en cuenta la historia sensorial previa: sistemas con memoria

La representación de un estado en el instante t+1 función de las entradas sensoriales en el inst t+1, la representación del estado en el instar anterior t y la acción seleccionada en el instante anterior t.



Búsqueda en espacios de estados

1. DISEÑO DE UN AGENTE DELIBERATIVO: BÚSQUEDA

El agente dispone de un **modelo del mundo** en el que habita y de un **modelo de los efectos de sus acciones** sobre el mundo. El agente es capaz de **razonar sobre esos modelos** para decidir qué hacer para conseguir un objetivo

Espacio de estados. Representación del conocimiento a través de las acciones del agente

Búsqueda en el espacio de estados. Resolución del problema mediante proyección de las distintas acciones

EL MUNDO DE BLOQUES

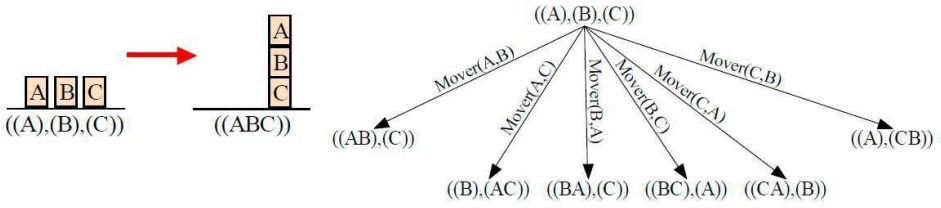
Supongamos un mundo cuadriculado con 3 bloques **A, B, C.** Inicialmente, todos los bloques están en el suelo. El objetivo es apilar los bloques de modo que **A** quede sobre **B, B** quede sobre **C**, y **C** esté en el suelo.

En cada momento, se dispone de la operación **mover(x,y)** para poner **x** sobre **y**, donde **x={A, B, C}** e **y={A, B, C, Suelo}**.

En cada momento, se conoce el estado del sistema. Lo modelamos con una secuencia de listas de objetos sobre objetos. Inicialmente, el estado es **((A), (B), (C))** y se desea llegar al estado **((ABC))**.

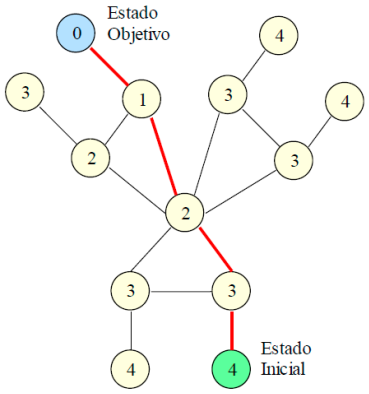
Asumimos que se descartan los **operadores imposibles** mover(A, A), mover(B, B), mover(C, C), etc., para cada estado.

Una estructura de **grafo dirigido** puede ser útil para buscar secuencias de acciones que nos lleven al objetivo final. En esta estructura, **un nodo representa un estado** del sistema y **un arco una posible acción**. La acción, aplicada al estado que representa al nodo origen, producirá el estado del nodo destino. Se denomina **grafo de estados.**



A la secuencia de acciones que lleva al agente desde un **estado inicial** hasta un **estado destino** se denomina plan. La búsqueda de dicha secuencia se denomina **planificación**.

- Grafos explícitos.
- Grafos implícitos.



Estado Inicial: ((ABC))

Acción 1: Mover(A, Suelo)
Acción 2: Mover(B, Suelo)
Acción 3: Mover(A, C)
Acción 4: Mover(B, A)

Estado objetivo alcanzado: ((BAC))

2. SISTEMAS DE BÚSQUEDA Y ESTRATEGIAS

2.1 ESTRATEGIAS IRREVOCABLES

En cada momento, el grafo explícito lo constituye **un único nodo**, que incluye la descripción completa del sistema en ese momento:

- Se selecciona una acción A.
- Se aplica sobre el estado del sistema E, para obtener el nuevo estado E' = A(E).
- Se borra de memoria E y se sustituye por E'.

2.2 ESTRATEGIA RETROACTIVA (BACKTRACKING)

En memoria **sólo guardamos un hijo** de cada estado; esto es, se mantiene el camino desde el estado inicial hasta el actual. El grafo explícito, por tanto, es realmente una lista.

¿Cuándo para el proceso? Cuando hemos llegado al objetivo y no deseamos encontrar más soluciones, o bien no hay más operadores aplicables al nodo raíz.

¿Cuándo se produce una vuelta atrás (o retroceso)?

- Cuando se ha encontrado una solución, pero deseamos encontrar otra solución alternativa.
- Cuando se ha llegado a un límite en el nivel de profundidad explorado o el tiempo de exploración en una misma rama.
- Cuando se ha generado un estado que ya existía en el camino.
- Cuando no existen reglas aplicables al último nodo de la lista (último nodo del grafo explícito).

2.3 BÚSQUEDA EN GRAFOS

En memoria se guardan todos los estados (o nodos generados hasta el momento), de forma que la búsqueda puede proseguir por cualquiera de ellos:

1. Seleccionar un estado E del grafo.
2. Seleccionar un operador A aplicable sobre E.
3. Aplicar A, para obtener un nuevo nodo A(E).
4. Añadir el arco E→ A(E) al grafo
5. Repetir el proceso.

2.4 MEDIDAS DEL COMPORTAMIENTO DE UN SISTEMA DE BÚSQUEDA

Completitud: hay garantía de encontrar la solución si esta existe

Optimalidad: hay garantía de encontrar la solución óptima

Complejidad en tiempo: ¿Cuánto tiempo se requiere para encontrar la solución?

Complejidad en espacio: ¿Cuánta memoria se requiere para realizar la búsqueda?

3. BÚSQUEDA SIN INFORMACIÓN

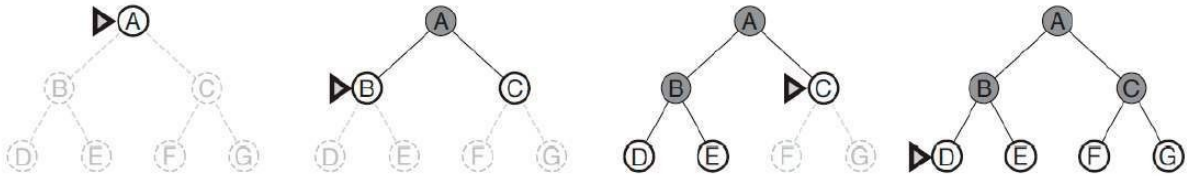
3.1 BÚSQUEDA EN ANCHURA

Características:

- **Completo**: encuentra la solución existe y el factor de ramificación es finito en cada nodo
- **Optimalidad**: si todos los operadores tienen el mismo coste encontrara la solución óptima
- **Eficiencia**:buena si las meta están cercanas

Problema: consume memoria exponenc

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```



BÚSQUEDA CON COSTO UNIFORME

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

3.2 BÚSQUEDA EN PROFUNDIDAD (SOBRE GRAFOS/RETROACTIVA)

Igual que la búsqueda en anchura cambiando FIFO por LIFO

Características:

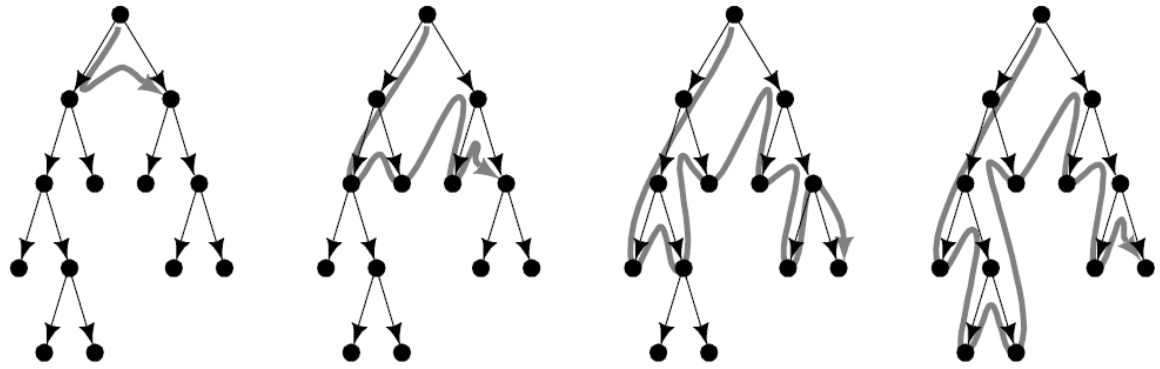
- **Completitud**: no asegu encontrar la solución
- **Optimalidad**: no asegu encontrar la solución óptima
- **Eficiencia**: bueno cuando l metas están alejadas del estado inicial, o hay problemas de memoria

No es bueno cuando hay ciclos

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
    cutoff_occurred? ← false
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      result ← RECURSIVE-DLS(child, problem, limit − 1)
      if result = cutoff then cutoff_occurred? ← true
      else if result ≠ failure then return result
  if cutoff_occurred? then return cutoff else return failure
```

3.3 DESCENSO ITERATIVO




```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

4. BÚSQUEDA CON INFORMACIÓN

4.1 HEURÍSTICAS

Algoritmo exacto. Si se tiene conocimiento perfecto.

Búsqueda sin información. Si no se tiene conocimiento.

En la mayor parte de los problemas que resuelven los humanos, se está en posiciones intermedias

Heurística: (del griego “heurisko" yo encuentro) conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio

Las heurísticas son criterios, métodos o principios para decidir cuál de entre varias acciones promete ser la mejor para alcanzar una determinada meta

En IA, entendemos por heurística un **método para resolver problemas** que en general **no garantiza la solución óptima**, pero que en media **produce resultados satisfactorios** en la resolución de un problema.

Una heurística encapsula el conocimiento específico/experto que se tiene sobre un problema, y sirve de guía para que un algoritmo de búsqueda pueda encontrar una solución válida aceptable.

Eventualmente, una heurística puede devolver siempre soluciones óptimas bajo ciertas condiciones (requiere demonstración).

En IA, implementaremos heurísticas como **funciones que devuelven un valor numérico**, cuya maximización o minimización guiará al proceso de búsqueda a la solución.

4.2 MÉTODOS DE ESCALADA

Si dibujamos las soluciones como puntos en el espacio, una **búsqueda local** consiste en seleccionar la solución mejor en el vecindario de una solución inicial, e ir viajando por las soluciones del espacio hasta encontrar un óptimo (local o global).

4.2.1 ALGORITMO DE ESCALADA SIMPLE

E: Estado activo

```
while (E no sea el objetivo
        y queden nodos por explorar a partir de E) {
        Seleccionar operador A para aplicarlo a E
        Evaluar f(A(E))
        if (f(A(E))>f(E)) {
                E = R(E)
        }
}
```

4.2.2 ALGORITMO DE ESCALADA POR LA MÁXIMA PENDIENTE

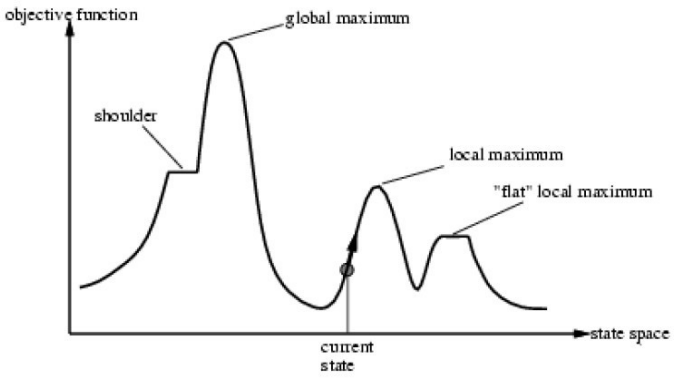
Características:

- **Completitud**: no tiene porqu encuentra la solución
- **Admisibilidad**: no siendo comple aun menos será admisible
- **Eficiencia**: rápido y útil si la función es monótona (de)creciente

E: Estado activo

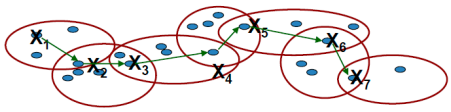
while (E no sea el objetivo
y queden nodos por explorar a partir de E) {
Para todos los operadores A_i, obtener $E_i=A_i(E)$
Evaluar $f(E_i)$ para todos los estados $E_i=A_i(E)$
Seleccionar Emax tal que $f(E_{max}) = \max\{f(E_i)\}$
if $(f(E_{max})>f(E))$ {
$E = E_{max}$
} **else return** E
}

MÉTODOS DE ESCALADA



4.2.3 ALGUNAS VARIACIONES ESTOCÁSTICAS. ENFRIAMIENTO SIMULADO

Es un método de búsqueda local. Se basa en principios de Termodinámica. Al contrario que otros métodos de ascensión de colinas, permite visitar soluciones peores que la actual para evitar óptimos locales.



En el campo de la Termodinámica, en los años 50 se simuló el proceso de enfriamiento en sistemas de partículas hasta que se llegaba a un estado estable.

El proceso simulaba la diferencia de energía del sistema, δE, y se quería verificar que la probabilidad de que el sistema tuviese el cambio δE seguía la siguiente fórmula (tes la temperatura actual del sistema; k es una constante física):

Analogía entre el proceso de enfriamiento y el algoritmo de enfriamiento simulado:

- Los **estados** por los que pasa el sistema físico de partículas equivalen a las **soluciones factibles** del algoritmo.
- La **energía E del estado actual** del sistema es el valor de la **función objetivo de la solución actual.** Ambos tienen que minimizarse.
- Un **cambio de estado** en el sistema equivale a **explorar el entorno de una solución y viajar a una solución vecina.**
- El **estado final estable** (congelado) es la **solución final** del algoritmo.

La **solución inicial** se puede gener
forma aleatoria, por conocimiento
experto, o por medio de otras técni
algorítmicas como greedy

La **actualización de temperatura** tar
es heurística, y hay varios métodos:

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
inputs: problem, a problem
        schedule, a mapping from time to "temperature"

current ← MAKE-NODE(problem.INITIAL-STATE)
for t = 1 to ∞ do
    T ← schedule(t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← next.VALUE – current.VALUE
    if ΔE > 0 then current ← next
    else current ← next only with probability e^(ΔE/T)
```

-
-

Número de vecinos a generar: Fijo N(T)= cte, dependiente de la temperatura N(T)= f(T), etc.

Tanto la temperatura inicial como la temperatura final son parámetros de entrada al algoritmo. Es difícil asignar un valor concreto a , por lo que la condición de parada se suele sustituir por un número específico de iteraciones a realizar.

Ventajas:

- Al ser un método probabilístico, tiene capacidad para salir de óptimos locales.
- Es eficiente.
- Es fácil de implementar.

Inconvenientes:

- Encontrar la temperatura inicial Ti, el método de actualización de temperatura **α**, el número de vecinos a generar en cada estado y el número de iteraciones óptimo es una tarea que requiere de **muchas pruebas de ensayo y error** hasta que ajustamos los parámetros óptimos.

Pese a todo, el algoritmo puede proporcionar soluciones mucho mejores que utilizando algoritmos no probabilísticos.

4.2.4 ALGORITMOS GENÉTICOS

La simulación de procesos naturales es un campo de investigación muy amplio en Inteligencia Artificial. Ejemplos son la **computación evolutiva, biocomputación, algoritmos bioinspirados**, etc.

Ejemplos:

- Algoritmos genéticos.
- Algoritmos basados en Colonias de Hormigas.
- Algoritmos basados en inteligencia de enjambres.

Son algoritmos de optimización basados en el proceso de la evolución natural de Darwin.

En un proceso de evolución, existe una **población de individuos**. Los más adecuados a su entorno **se reproducen y tienen descendencia** (a veces **con mutaciones** que mejoran su idoneidad a entorno). Los más adecuados sobreviven para la siguiente generación.

No necesitan partir de un nodo/estado inicial: ¡Hay toda una población!

Su objetivo es encontrar una solución cuyo valor de función objetivo sea óptimo.

Cromosoma ↔ Vector representación de una solución al problema.

Gen ↔ Característica/Variable/Atributo concreto del vector de representación de una solución

Población ↔ Conjunto de soluciones al problema.

Adecuación al entorno ↔ Valor de función objetivo (fitness).

Selección natural ↔ Operador de selección.

Reproducción sexual ↔ Operador de cruce.

Mutación ↔ Operador de mutación.

Cambio generacional ↔ Operador de reemplazamiento.

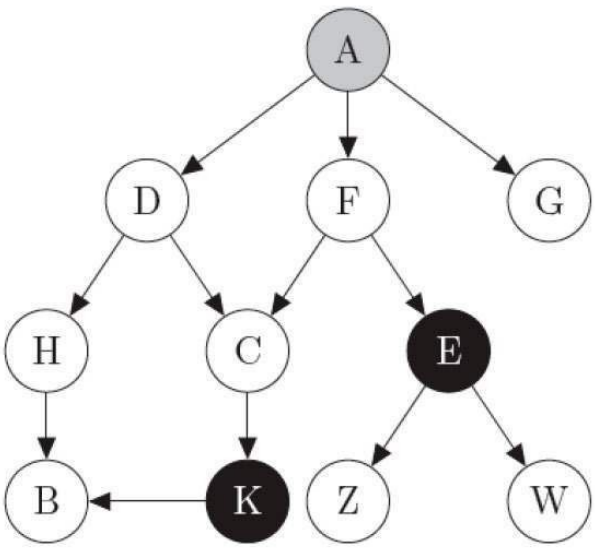
4.3 BÚSQUEDA PRIMERO EL MEJOR

4.3.1 BÚSQUEDA PRIMERO MEJOR GREEDY (BFS)

Solo movimientos en horizontal y vertical.

Heurística: distancia Manhattan. Sur de la distancia vertical y horizontal medida en celdas

	5	S	5	6	
6	4	3		5	
4	3			4	
3				3	
2		G	1	2	
3	2	1			



4.3.2 ALGORITMO A*

Estrategia de búsqueda sobre grafos

Uso de ABIERTOS y CERRADOS

Heurística: f(n) = g(n) + h(n)

Estructura de un nodo:

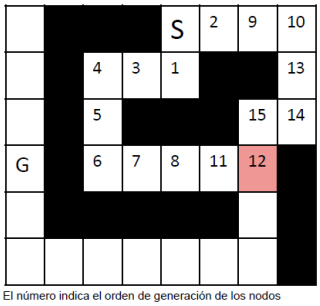
estado	hijos	mejor_padre	g
			h

Búsqueda en grafos donde abiertos es una cola con prioridad ordenada de acuerdo a f(n).

ABIERTOS contiene el nodo inicial, CERRADOS está vacío

Comienza un ciclo que se repite hasta que se encuentra solución o hasta que ABIERTOS queda vacío

- Seleccionar el mejor nodo de ABIERTOS
- Si es un nodo objetivo terminar
- En otro caso se expande dicho nodo
- Para cada uno de los nodos sucesores
  - Si está en ABIERTOS insertarlo manteniendo la información del mejor padre
  - Si está en CERRADOS insertarlo manteniendo información del mejor padre y actualizar la información de los descendientes
  - En otro caso, insertarlo como un nodo nuevo



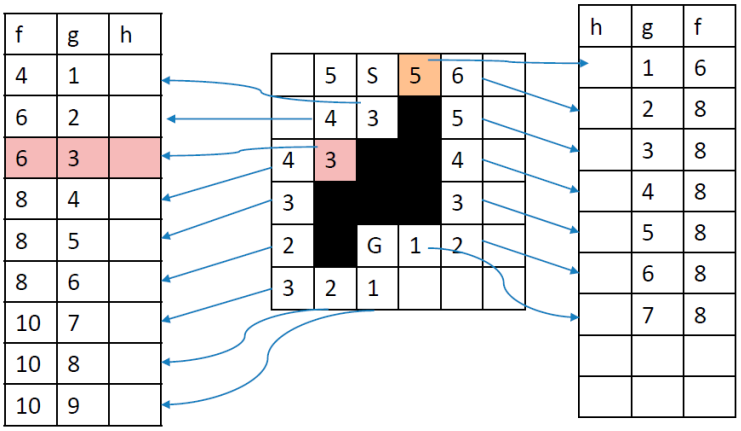
El número indica el orden de generación de los nodos

Nodos repetidos en abiertos: Si un nodo n ya existe en al hay que comprobar si el nuevo camino es mejor que el anter

Nodos repetidos en cerrados:

- **Caso 1:** el nuevo padre de n no es mejor que el padre anterior. FIN
- **Caso 2:** el nuevo padre de n es mejor que el padre anterior.
  - Actualizar enlace al padre, actualizar el nuevo valor de coste del camino. **Propagar la información a los hijos de n**

Para cada nodo n, A* tiene en cuenta la **estimación, h(n)**, al objetivo y el **coste real, g(n)** desde el inicio.



A* detecta que el camino inicial no tiene por qué ser el mejor cuando el nodo rojo genera su hijo y éste es peor que el nodo naranja

Toda la descendencia de naranja no empeora (empatan) respecto al hijo de rojo

Usamos como criterio de desempate el nodo más joven.

CASOS PARTICULARES DEL ALGORITMO A*

Supongamos que usamos el algoritmo A* pero tomamos siempre h(n)=0, entonces

- En el caso que el coste de cada arco sea siempre unidad, el algoritmo se comporta como la búsqueda en anchura.
- En otro caso, el algoritmo se comporta como la búsqueda de coste uniforme.

Supongamos que usamos el algoritmo A* pero tomamos siempre g(n)=0, entonces el algoritmo se comporta como el algoritmo de búsqueda primero el mejor greedy.

Características:

- **Completitud**: si existe solución, la encuentra bajo condiciones muy generales.
- **Admisibilidad**: si hay una solución óptima, bajo unas condiciones muy generales y si la función h(n) es admisible: $h(n) \leq h^*(n)$

4.3.3 BÚSQUEDA DIRIGIDA (BEAM SEARCH)

Una variación del algoritmo A* que **limita el factor de ramificación** en problemas complejos.

Cada vez que se expande un nodo, se generan sus sucesores, se evalúan con la función heurística f, y se eliminan aquellos sucesores con peor valor de la f, quedándonos con un número fijo de sucesores.

Los procesos de percepción no siempre pueden obtener la información necesaria acerca del **estado** del entorno. Las acciones pueden no disponer siempre de modelos de sus **efectos**. Puede haber otros procesos físicos, u **otros agentes**, en el mundo.

En el tiempo que transcurre desde la construcción de un plan, el mundo puede cambiar de tal manera que el plan ya no sea adecuado

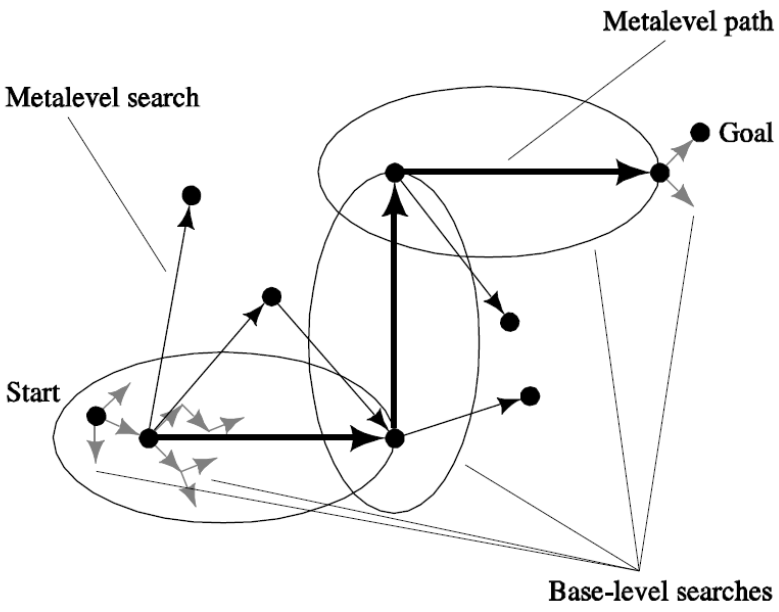
Podría suceder que se le requiriese al agente actuar antes de que pudiese completar una búsqueda de un estado objetivo

Aunque el agente dispusiera de tiempo suficiente, sus recursos de memoria podrían no permitirle realizar la búsqueda de un estado objetivo

4.4 BÚSQUEDA CON HORIZONTE

Se establece una profundidad máxima (horizonte) y se realiza la búsqueda con esa profundidad máxima. A veces es necesario cambiar el criterio de búsqueda del objetivo

4.5 BÚSQUEDA JERÁRQUICA

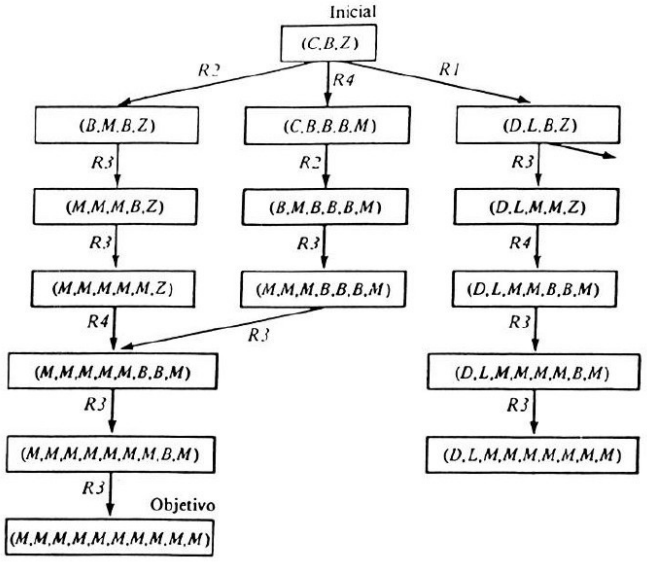


5. PROBLEMAS DESCOMPONIBLES Y BÚSQUEDAS

Base de datos inicial (C, B, Z)

Operadores:

- R1: C→ (D,L)
- R2: C→ (B,M)
- R3: B→ (M,M)
- R4: Z→ (B,B,M)

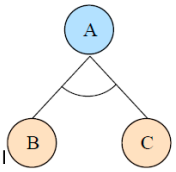


5.1 GRAFO Y/O

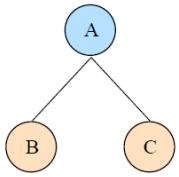
Descomposición de problemas: arcos Y

Resolución de problemas: arcos O

Concepto de solución: subgrafo solución

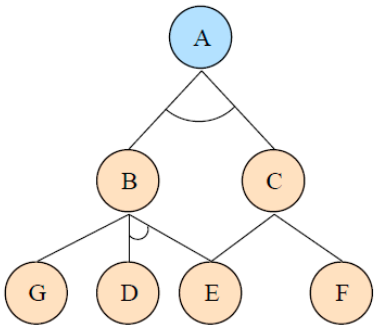


Grafo Y: Para completar el objetivo/tarea A, es necesario terminar antes objetivos/tareas B y C. En el cálculo proposicional, la expresión del grafo Y ante correspondiente sería de la siguiente form



Grafo O: Para completar el objetivo/tarea A, es necesario terminar antes o bien objetivo/tarea B, o bien el objetivo/tarea C. En el cálculo proposicional, la expres del grafo O anterior correspondiente sería de la siguiente for

Grafo Y/O: Combinación de grafos Y y grafos O que indican el orden de consecución de tareas a realizar para alcanzar el objetivo. En el cálculo proposicional, la expresión del grafo Y/O anterior correspondiente sería de la siguiente forma:



Para resolver un grafo Y/O, cada nodo se resuelve d siguiente manera:

- **Si es un nodo Y:** Resolver todos sus hijos. Combinar solución y solucionar el nodo. Devolver su solución.
- **Si es un nodo O:** Resolver un hijo y ver si dev solución. En caso contrario, resolver el siguiente hijo, Cuando ya esté resuelto algún hijo, combinar la soluci en el nodo y devolverla.
- **Si es un nodo terminal:** Resolver subproblema asoc devolverla.

Mejora: Para seleccionar el orden de resolución de nodos hijos, se puede utilizar alguna medida de estimación del coste de resolución

Búsqueda con adversario: juegos

1. JUEGOS BIPERSONALES CON INFORMACIÓN PERFECTA

Estas situaciones se estudian y resuelven utilizando la **Teoría de Juegos**. La teoría matemática de juegos fue inventada como tal por **John von Neumann y** por **OskarMorgensternen** 1944.

Juego. Es cualquier situación de decisión, caracterizada por poseer una interdependencia estratégica, gobernada por un conjunto de reglas y con un resultado bien definido.

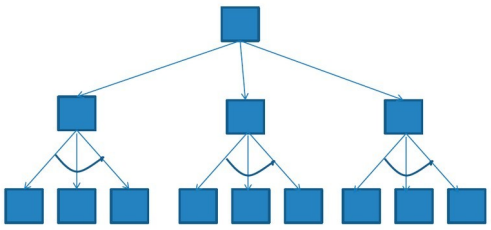
En un juego, cada jugador intenta conseguir el mayor beneficio para sus intereses. La solución de un juego permite indicar a cada jugador qué resultado puede esperar y cómo alcanzarlo.

2. ÁRBOLES DE EXPLORACIÓN DE JUEGOS

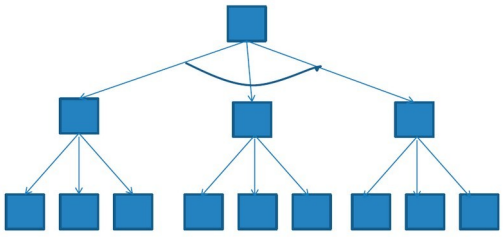
Un árbol del juego es una representación explícita de todas las formas de jugar a un juego

Correspondencia entre árboles de juegos y árboles Y/O:

Árboles de exploración de juegos: para el primer jugador



Árboles de exploración de juegos: para el segundo jugador



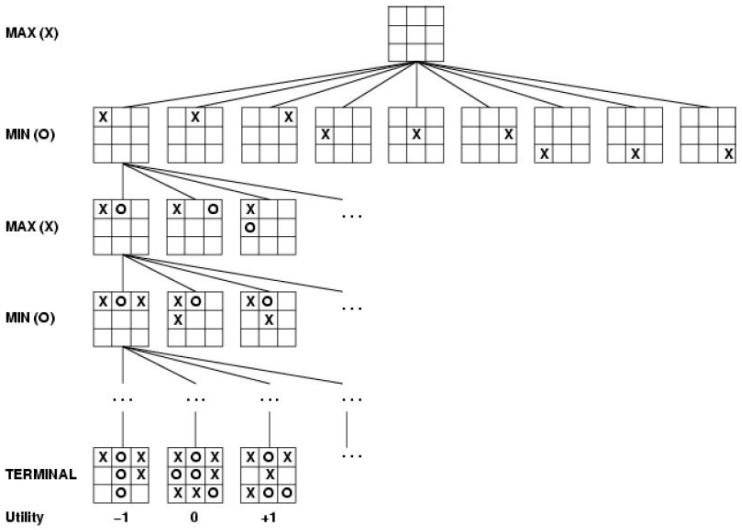
NOTACIÓN MIN-MAX

MAX: primer jugador

MIN: segundo jugador

Nodos MAX y nodos MIN

Los nodos terminales se etiquetan con V, D o E desde el punto de vista de MAX



ALGORITMO STATUS

Si J es un nodo MAX no terminal, entonces STATUS(J)
- V si alguno de los sucesores de J tiene STATUS V
- D si todos los sucesores de J tienen STATUS D
- E en otro caso

Si J es un nodo MIN no terminal, entonces STATUS(J)
- V si todos los sucesores de J tienen STATUS V
- D si alguno de los sucesores de J tiene STATUS D
- E en otro caso

3. EL MODELO BÁSICO

3.1 HEURÍSTICAS

Heurística para el ajedrez del programa de Turing: B/N

Heurística para las damas del programa de Samuel: función lineal de varias características

$$f(n) = w_1 f_1(n) + w_2 f_2(n) + \dots + w_n f_n(n)$$

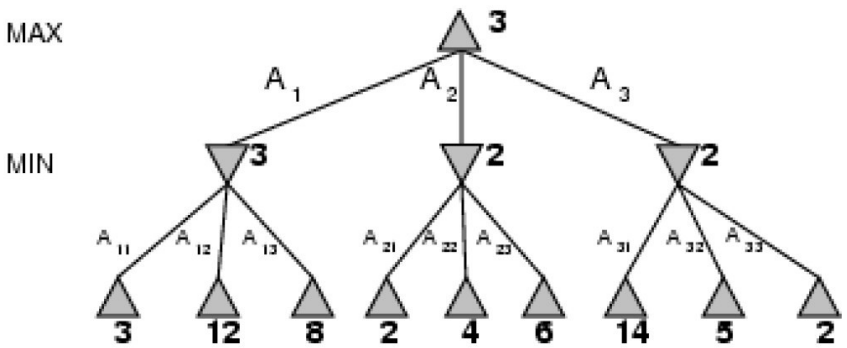
3.2 ALGORITMO MINIMAX

El valor V(J) de un nodo J de la frontera de búsqueda es igual al de su evaluación estática; en otro caso

Si J es un nodo **MAX**, entonces su valor V(J) es igual al máximo de los valores de sus nodos sucesores

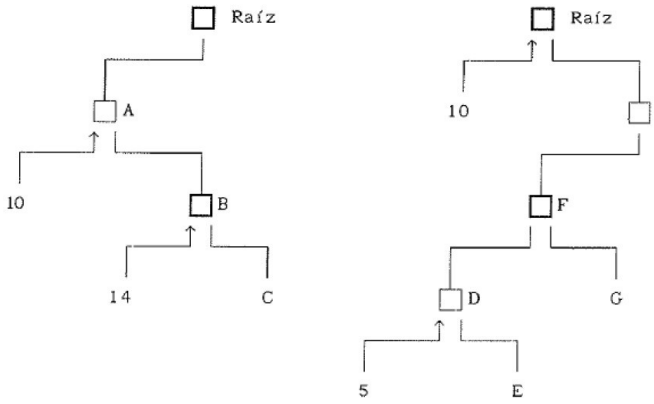
Si J es un nodo **MIN**, entonces su valor V(J) es igual al mínimo de los valores de sus nodos sucesores.

Para determinar el valor minimax, V(J) de un nodo J, hacer lo siguiente:
- Si J es un nodo terminal, devolver V(J)=f(J); en otro caso
- Para k=1,2,..,b, hacer:
  - Generar J_k, el k-ésimo sucesor de J
  - Calcular $V(J_k)$
  - Si k=1, hacer $AV(J) \leftarrow V(J_1)$; en otro caso, para k>=2,
  - hacer $AV(J) \leftarrow \max\{AV(J), V(J_k)\}$ si J es un nodo MAX o
  - hacer $AV(J) \leftarrow \min\{AV(J), V(J_k)\}$ si J es un nodo MIN
- Devolver V(J)=AV(J)



3.3 ALGORITMO ALFA-BETA

	Para nodos	Se calcula	es
Cota alfa	Nodos MIN	Máximo de los nodos MAX	Cota inferior
Cota beta	Nodos MAX	Mínimo de los nodos MIN	Cota superior



Para calcular el valor V(J, alfa, beta), hacer lo siguiente:

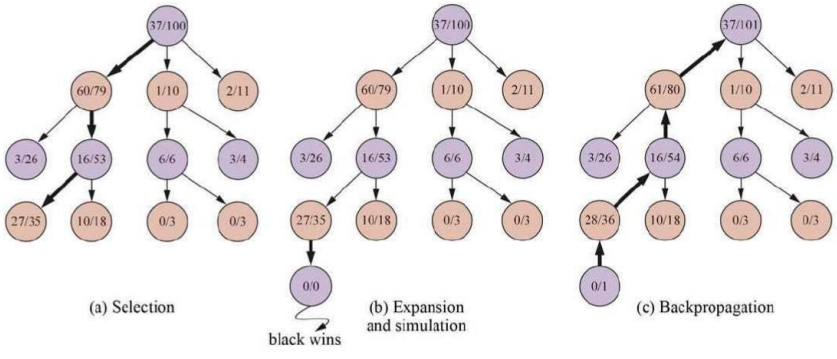
1. Si J es un nodo terminal, devolver V(J)=f(J). En otro caso, sean $J_1, J_k, \ldots, J_b$ los sucesores de J. Hacer k← 1 y, si J es un nodo MAX ir al paso 2; si J es un nodo MIN ir al paso 5.
2. Hacer alfa← max(alfa, V(J_k, alfa, beta)).
3. Si alfa >= beta devolver beta; si no, continuar
4. Si k=b, devolver alfa; si no, hacer k←k+1 y volver al paso 2.
5. Hacer beta← min(beta, V(J_k, alfa, beta)).
6. Si beta <= alfa devolver alfa; si no, continuar
7. Si k=b, devolver beta; si no, hacer k←k+1 y volver al paso 5.

3.4 IDEA DE MCTS

El modelo básico de MCTS no hace uso de funciones heurísticas. Estima el valor de un estado como la utilidad promedio sobre un número de simulaciones de juegos completos empezando en dicho estado. Hay que determinar que jugada hace cada jugador durante la simulación:

- Aleatoria.
- Mediante políticas.

Exploración/Explotación.



Comportamiento Inteligente: Representación del conocimiento e inferencia basados en lógica

1. REPRESENTACIÓN DEL CONOCIMIENTO EN IA

Hemos estudiado varias formas de modelar el mundo de un agente, entre ellas:

- Representaciones icónicas: Simulaciones del mundo que el agente podía percibir.
- Representaciones descriptivas: Valores binarios que describían aspectos ciertos o falsos sobre el mundo.

Las representaciones descriptivas tienen ciertas ventajas sobre las icónicas:

- Son más sencillas.
- Son fáciles de comunicar a otros agentes.
- Se pueden descomponer en piezas más simples.

Además, hay información del entorno del agente que no se puede representar mediante modelos icónicos, tales como:

- Leyes generales. "Todas las cajas azules pueden ser cogidas".
- Información negativa. "El bloque A no está en el suelo", sin decir dónde está el bloque A.
- Información incierta. "O bien el bloque A está sobre el bloque C, o bien el bloque A está sobre el bloque B".

Sin embargo, este tipo de información es fácil de formular como conjunto de restricciones sobre los valores de las características binarias del agente. Estas restricciones representan conocimiento sobre el mundo.

A menudo, este conocimiento sobre el mundo puede utilizarse para razonar sobre él y hallar nuevas características del mismo.

Ejemplo:

- El conocimiento que se tiene es "Todos los pájaros vuelan"; y "Piolín es un pájaro".
- Se puede razonar, por tanto, que "Piolín vuela".

Otro Ejemplo: Un robot sólo puede levantar un bloque si tiene suficiente batería y el bloque es elevable. Entonces, el conocimiento sobre el mundo es: "Si el bloque es elevable y hay suficiente batería, entonces es posible levantar el bloque".

El robot "sabrá" si es capaz de levantar el bloque a partir de este conocimiento sobre su entorno.

2. EL CÁLCULO PROPOSICIONAL

Elementos de representación: proposiciones y conectivas

∧ (끬毋 (끬殼› (끬彈끬彈끬彈끬彈끬彈끬彈끬彈끬彈끬彈끬彈

Inferencia: deducciones con reglas, hechos y Modus-Ponens

Ejemplos: llueve, (¬Nieva∨llueve)⇒Hay-hielo

Ventaja: representación de tipo general, y decidible (en tiempo finito es capaz de decidir si una proposición es deducible de la información disponible o no)

Problema: si se quiere razonar sobre conjuntos de cosas. Por ejemplo, grafos, o jerarquías de conceptos.

2.1 REGLAS DE INFERENCIA

Las reglas de inferencia nos permiten producir nuevas FBFs a partir de las que ya existen, Algunas de las más comunes son:

- Q puede inferirse a partir de P 뀔殆 ⊃(뀔殊dus ponens)
- 뀔殆 ∧s뀔殊uede inferir a través de la conjunción de P y Q
- 뀔殊 ∧s뀔殆uede inferir desde뀔殆 ∧(뀔殊mutatividad)
- P (también Q) se puede inferir desde뀔殊 ∧ 뀔殆
- 뀔殆 ∨s뀔殊uede inferir bien desde P, bien desde Q
- P se puede inferir desde(¬ 뀔殆)

2.2 DEFINICIÓN DE DEMOSTRACIÓN

Supongamos∆ un conjunto de FBFs, y una secuencia de n FBFs $w_1, w_2, w_3, \ldots, w_n$.

Esta secuencia de FBFs se llama demostración o deducción de a partir de∆ si, y sólo si, cada w_i de la secuencia pertenece a∆ o puede inferirse a partir de FBFs en∆.

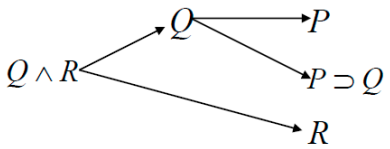
Si existe tal demostración, entonces decimos que w_n es un teorema de∆, y decimos que w_n puede demostrarse desde∆ con la siguiente notación: $\Delta \vdash w_n$, o como $\Delta \vdash_R w_n$ para indicar que w_n se demuestra desde∆ mediante las reglas de inferencia R.

EJEMPLO

Sea el conjunto de FBFs∆, $\Delta = \{$ 뀔殆 , 뀔殆 ⊃ 뀔殊 , 뀔殊 , 뀔殆 ∧ 뀔殊 $\}$ Entonces la siguiente secuencia es una demostración de la Fórmula Bien Formada $R \wedge Q$:

$$\{ 뀔殆 , 뀔殆 \supset 뀔殊 , 뀔殊 , 뀔殆 \wedge 뀔殊 \}$$

La demostración se puede llevar a cabo fácilmente a través del siguiente árbol de demostración, utilizando∆, y las reglas de inferencia:



2.2.1 INTERPRETACIÓ N

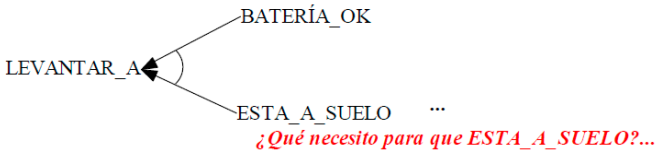
A la hora de resolver problemas con IA, el papel de la semántica es esencial: Hay que hacer una correcta interpretación del sistema lógico subyacente. Conlleva asociar conceptos del lenguaje lógico con su significado (semántica) en el mundo real o en el mundo del entorno del agente.

Ejemplo: Se desea implantar el conocimiento "Si la batería funciona y el bloque A está en el suelo, entonces se puede levantar" dentro de un agente.

- Definimos los átomos *BATERIA_OK*, *ESTA_A_SUELO*, *LEVANTAR_A*.
- Definimos la FBF뀔歪뀔歪뀔歪뀔歪뀔殊뀔歪뀔歪 _ 뀔殄뀔殄 ∧ 뀔歪뀔蹙뀔歪뀔歪_ 뀔歪 _ 뀔蹙뀔殛뀔歪뀔殛뀔殄

En un agente cuyo objetivo sea "levantar el bloque A", con este conocimiento puede especificar las acciones que debe llevar a cabo para realizar su acción. Esta planificación se puede hacer mediante árboles de demostración. La representación de grafos Y/O es muy útil en este tipo de problemas.

Ejemplo: "Debo levantar el bloque A, ¿qué necesito para poder levantarlo?



2.2.2 TABLAS DE LA VERDAD

Las tablas de verdad establecen la semántica de las conectivas proposicionales. Para una representación interna de un agente con n características, el número de combinaciones (formas de ver el mundo) es 2^n

Para dos características A y B:

A	B	A y B	A o B	No A	A implica B
V	V	V	V	F	V
V	F	F	V	F	F
F	V	F	V	V	V
F	F	F	F	V	V

2.3 SATISFACIBILIDAD Y MODELOS

Una interpretación satisface una FBF cuando a la FBF se le asocia el valor V bajo esa interpretación. A la interpretación que satisface una FBF se le denomina modelo.

Bajo una interpretación una FBF debe indicar una restricción que nos informe sobre algún aspecto del mundo. Ejemplo: $\alpha \wedge \beta \supset \gamma$

- Para la interpretación A=BATERIA_OK, B=ESTA_A_SUELO, C=LEVANTAR_A, la semántica se corresponde con el entorno y el mundo a modelar.
- Para la interpretación A=TOCA_LOTERIA, B=TENGO_SALUD, C=TIRAR_POR_VENTANA, la semántica es inconsistente con lo que se modela. Esta interpretación no es válida porque no satisface la FBF.

2.4 VALIDEZ Y EQUIVALENCIA

Se dice que una FBF es válida si se cumple independientemente de la interpretación que se le asocie. Ejemplo: $\alpha \supset \alpha$, $\neg(\alpha \wedge \neg\alpha)$. Las interpretaciones válidas no modelan aspectos del mundo y deben ser evitadas en el diseño del comportamiento del agente.

Dos FBFs son equivalentes si sus tablas de verdad son idénticas. Ejemplo: $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$. En el diseño de agentes, debemos evitar FBFs con interpretaciones equivalentes para hacer más eficiente el proceso de razonamiento.

2.5 CONSECUENCIA LÓGICA

Si una FBF w tiene el valor verdadero bajo todas aquellas interpretaciones para las cuales cada una de las FBFs del conjunto Δ tiene el valor verdadero, entonces decimos que Δ lleva lógicamente a w, que w se sigue lógicamente de Δ, o que w es una consecuencia lógica de Δ

- $\{P\} \models P$
- $\{\alpha, \alpha \supset \beta\} \models \beta$
- $\alpha \wedge \beta \models \alpha$
- La noción de consecuencia lógica es importante ya que nos proporciona un mecanismo para demostrar que si ciertas proposiciones son ciertas entonces otras deben serlo también.

2.6 SOLIDEZ Y COMPLETITUD

Si, para el conjunto de FBFs Δ y para la FBF w, $\Delta \vdash w$ implica que $\Delta \models w$, decimos que el conjunto de reglas de inferencia R es sólido.

Si, para para el conjunto de FBFs Δ y para la FBF w, tenemos que siempre que $\Delta \models w$ existe una demostración de w a partir de Δ utilizando el conjunto de reglas de inferencia R, decimos que R es completo.

3. CÁLCULO DE PREDICADOS

El cálculo proposicional es limitado. Supongamos nuestro mundo de bloques. Para decir que el bloque A está sobre el bloque B, deberíamos establecer una interpretación SOBRE_A_B. Para representar esta situación con todos los bloques usando cálculo proposicional, necesitaríamos tantos literales como posibilidades. Además, supongamos dos literales P y Q, con la semántica asociada P=SOBRE_A_B, Q=SOBRE_B_C.

En lenguaje natural y mediante el conocimiento que tenemos del problema, nosotros (diseñadores, personas, etc.) sabemos que A está sobre B, y que B está sobre C. Por tanto, C está por debajo de A. Sin embargo, necesitaríamos más proposiciones y más complejas para implementar este conocimiento utilizando únicamente cálculo proposicional.

Sería de gran utilidad un lenguaje que permitiese definir objetos y relaciones entre ellos. El cálculo de predicados nos permite esta opción y, además solventa los problemas planteados en la diapositiva anterior

Ejemplo: Para decir que 꾉쬹꾉殄꾉歪꾉殊꾉歪 _ 꾉歪para cualquier 꾉殊꾉歪꾉歪_꾉將 predicados nos evita tener que reescribir todas las proposiciones del cálculo proposicional de las situaciones que pueden darse. Podemos abstraer los objetos a variables y escribir:

꾉쬹꾉殄꾉歪꾉殊꾉歪 (꾉鬱 , 꾉毋) ⊃ ¬ 꾉殛꾉歪꾉歪꾉殊꾉歪 (꾉毋)

El significado sería "cuando un objeto x esté sobre otro y, entonces y no estará libre

Elementos de representación:

- Términos: Constantes (UGR), Variables (X), Funciones (siguiente(X))
- Fórmulas atómicas: Predicados definidos sobre términos
  - trabaja-como(empleado1, director)
  - tiene-hijos(empleado1,1)
- Fórmulas bien formadas (fbf): Fórmulas atómicas unidas por conectivas (∧,∨,¬,→) y cuantificadas (∀,∃)
  - ∀X,Y trabaja-como(X,director), tiene-hijos(X,Y), Y<=2gana(X,60000)
  - ∀X,Y trabaja-como(X,director), tiene-hijos(,;Y), Y>2gana(X,70000)

3.1 INFERENCIA

Inferencia: Todas las de lógica proposicional + instanciación universal

Instanciación universal: si tenemos ∀X p(X) entonces se puede deducir p(a), p(Y) . . .

Ejemplo: Todos los hombres son mortales, Sócrates es un hombre, por tanto Sócrates es mortal:

a. R1:∀X hombre(X)→ mortal(X)
b. hombre(4ócrates)
c. R1 y X=4ócrates: hombre(4ócrates)→ mortal(4ócrates)
d. (b y c) mortal(4ócrates)

DEDUCCIÓ N CO N MO DUS-PO NENS

En primero del grado en Informática de la UGR se imparte las asignaturas de Fundamentos de Programación, . . .

1. curso-asignatura(FundamentosdePrograma ción,1)
2. curso-asignatura(FundamentosdelSoftware,1)

Ana Morales Pérez acaba de matricularse en primero de la titulación.

a. primera-matricula(anaMorales,UGR)

Si X=anaMorales, U=UGR, e Y=FundamentosdePrograma ción, (unificación) por el Modus-Ponens, a partir de la regla R1, de 1 y de a), se puede deducir que

- matriculado-en(anaMorales, FundamentosdePrograma ción)

Si X=anaMorales, U=UGR, y Y=FundamentosdelSoftware, por el Modus-Ponens, a partir de la regla R1, de 2 y de a), se puede deducir que

- matriculado-en(anaMorales, FundamentosdelSoftware)

3.2 CARACTERÍSTICAS DEL CÁLCULO DE PREDICADOS

Ventaja: representación de tipo general más rica que la proposicional

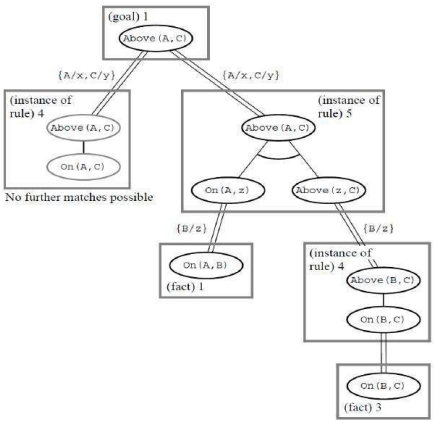
Características de un sistema de razonamiento lógico:

- solidez: para estar seguro que una conclusión inferida es cierta.
- completitud: para estar seguros de que una inferencia tarde o temprano producirá una conclusión verdadera.
- decidibilidad: para estar seguros de que la inferencia es factible.

La refutación mediante resolución es sólida y completa.

Problema: el cálculo de predicados es semidecidible y adem en los casos en que la refutación mediante resolución termi el procedimiento es NP-duro.

Solución: subconjuntos decidibles de lógica de predicadc (cláusulas de Horn)



Existe un lenguaje de programación que permite crear y ejecutar programas en lógica de predicados: PROLOG

conectados(X,Y) :-conectados(Y,X).

alcanzable(X,Y) :-conectados(X,Y).

alcanzable(X,Y) :-conectados(X,Z), alcanzable(Z,Y).

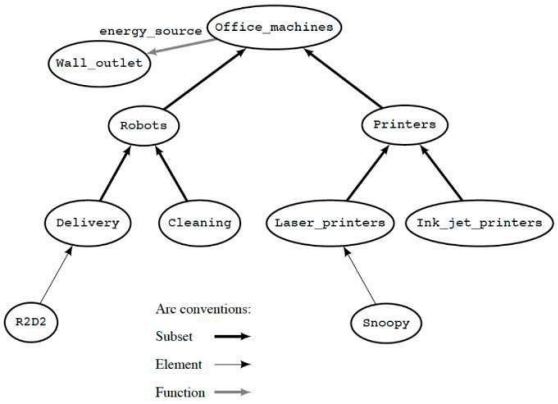
3.3 REDES SEMÁNTICAS

Las redes semánticas son estructuras gráficas qu codifican el conocimiento taxonómico sobre objetos propiedades de estos

Propiedades: nodos etiquetados con constantes relación

Objetos: nodos etiquetados con constantes de objet

- Arcos de jerárquica
- Arcos de pertenencia
- Arcos de función



3 .4 OTRAS LÓGICAS

Lógicas de segundo orden (o de orden superior). Tienen dos (o tres) tipos definidos: los objetos y los conjuntos o funciones sobre los mismos (o ambos). Es equivalente a decir que los predicados pueden tomar otros predicados como argumentos

Lógicas modales y temporales. Necesario y posible

Lógica difusa. Grados de pertenencia

3.4.1 LÓ GICA DIFUSA

Extensión de la lógica clásica diseñada para permitir el razonamiento sobre conceptos imprecisos

- ”la velocidad del motor es muy alta”
- ”el paciente tiene fiebre moderada”
- ”si el paciente tiene fiebre muy alta y es muy joven, entonces la dosis debe de ser moderada”

Ejemplo: control del movimiento de un robot

4. INTRODUCCIÓN A LOS SISTEMAS BASADOS EN EL CONOCIMIENTO

Una gran cantidad de aplicaciones reales de la IA se basan en la existencia de una gran masa de conocimiento: Diagnóstico médico, Diseño de equipos, Sistemas de Recomendación, etc.

Este tipo de sistemas se denominan Sistemas Basados en el Conocimiento, ya que este ocupa la parte central de la solución al problema a resolver.

Un Sistema Basado en el Conocimiento (SBC) necesita 3 componentes básicas:

- Una Base de Conocimiento (BC), que contenga el conocimiento experto necesario sobre el problema a resolver. Puede ser:
  - Estática, si la BC no varía a lo largo del tiempo.
  - Dinámica, cuando se añaden nuevos hechos o reglas, o se modifican las existentes a lo largo del tiempo.
- Un Motor de Inferencia, que permite razonar sobre el conocimiento de la BC y los datos proporcionados por un usuario.
- Una interfaz de usuario para entrada/salida de datos.

4.1 SISTEMAS EXPERTOS BASADOS EN REGLAS (SEBR)

Un SEBR es un SBC donde el conocimiento se incluye en forma de reglas y hechos. Estas reglas y hechos pueden implementarse, por ejemplo, mediante el cálculo de predicados.

El proceso de construcción de un SEBR es el siguiente:

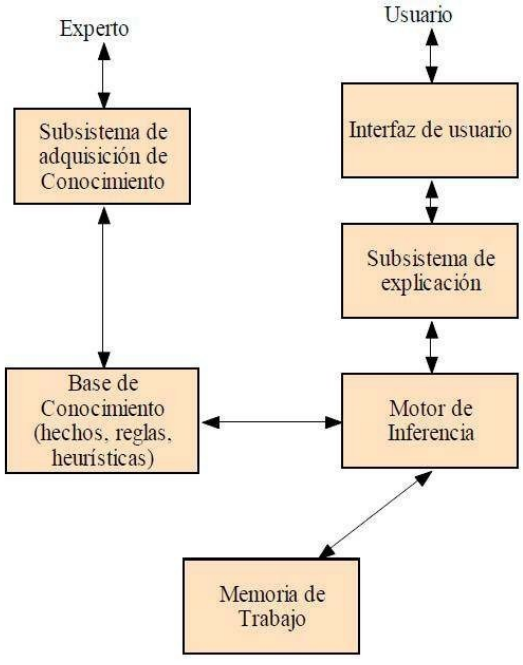
- Se extrae el conocimiento experto (bibliografía, entrevistas a expertos reales, etc.).
- Se modela y se adquiere el conocimiento, utilizando un lenguaje adecuado (cálculo de predicados, otras lógicas más avanzadas, etc.)
- Se crea la Base de Conocimiento con el conocimiento adquirido.

Por otra parte, también se necesita:

- Una interfaz de usuario, para poder utilizar el sistema y adquirir/enviar datos.
- Un subsistema de explicación, para los casos en los que sea necesario indicar al usuario porqué se llega a las conclusiones que se llegan.
- Un Motor de Inferencia, para razonar sobre la Base de Conocimiento y los datos proporcionados por el usuario.

El esquema general de diseño de un SEBR es el siguiente:

La memoria de trabajo contiene la información relevante que el Motor de Inferencia está usando para razonar las respuestas para el usuario.



Introducción al Aprendizaje Automático

1. DISTINTOS TIPOS DE APRENDIZAJE

El aprendizaje es una capacidad fundamental de la inteligencia humana, que nos permite:

- Adaptarnos a cambios de nuestro entorno.
- Desarrollar una gran variedad de habilidades.
- Adquirir experiencia en nuevos dominios.

El aprendizaje automático cubre una amplia gama de fenómenos como:

- El perfeccionamiento de la habilidad.
- La adquisición del conocimiento.

El aprendizaje es esencial en entornos desconocidos.

Programa de IA (Búsqueda, SBC, Planificación, ...):

- Su límite está en el conocimiento que se les ha proporcionado.
- No resuelven problemas más allá de esos límites.

El aprendizaje modifica el mecanismo de decisión del agente para mejorar su comportamiento.

Aprendizaje automático: programas que mejoran su comportamiento con la experiencia.

Un programa de ordenador se dice que aprende de la experiencia E con respecto a alguna clase de tareas T y a alguna medida de comportamiento P, si su comportamiento en tareas de T, medido a través de P, mejora con la experiencia E.

1.1 APRENDIZAJE INDUCTIVO

Uno de los puntos clave para el aprendizaje es el tipo de realimentación disponible en el proceso:

- **Aprendizaje supervisado:** Aprender una función a partir de ejemplos de sus entradas y salidas.
- **Aprendizaje no supervisado**: Aprender a partir de patrones de entradas para los que no se especifican los valores de sus salidas.

APRENDIZAJE SUPERVISADO

Métodos basados en modelos: representan el conocimiento aprendido en algún lenguaje de representación (modelo o hipótesis).

Métodos basados en instancias: representan el conocimiento aprendido como un conjunto de prototipos descritos en el mismo lenguaje usado para representar la evidencia.

Una hipótesis estará bien generalizada si se pueden predecir ejemplos que no se conocen. La hipótesis se dice **consistente** ya que satisface a todos los datos

Navaja de Ockham: elegir la hipótesis más simple consistente con los datos

Las hipótesis se pueden expresar de diversas formas: Árboles de decisión, Reglas, Redes neuronales, Modelos bayesianos o probabilísticos, etc.

Los árboles de decisión y las reglas son algunos de los modelos más usados en aprendizaje automático.

Se dice que un problema de aprendizaje es **realizable** si el espacio de hipótesis contiene a la función verdadera

Algoritmos más ampliamente utilizados:

- **Algoritmos basados en el "divide y vencerás" (splitting)**: consisten en ir partiendo sucesivamen los datos en función del valor de un atributo seleccionado cada vez (aprendizaje de árboles de decisión).
- **Algoritmos basados en el "separa y vencerás" (covering)**: consisten en encontrar condiciones d las reglas que cubran la mayor cantidad de ejemplos de una clase y la menor en el resto de la clase (aprendizaje de reglas).

2. MODELOS INDUCTIVOS SOBRE ÁRBOLES DE DECISIÓN

Un **árbol de decisión** toma como entrada un objeto o una situación descrita a través de un conjunto de atributos y devuelve una "decisión", el valor previsto de la salida dada la entrada.

Atributos: discreto o continuos.

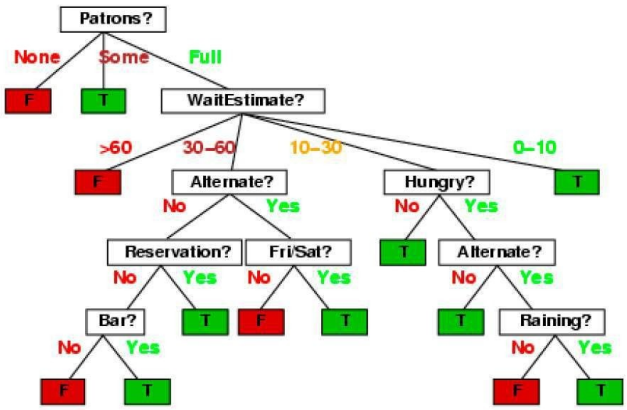
Salida:

- Discreta: clasificación.
- Continua: regresión.

Los ejemplos positivos son aquellos en los que la meta esperar es verdadera ($X_1, X_3, \ldots$)

Los ejemplos negativos son aquellos en los que es falsa ($X_2, X_6, \ldots$)

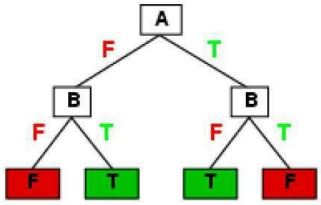
El conjunto de ejemplos completo se denomina **conjunto de entrenamiento**



2.1 EXPRESIVIDAD DE LOS ÁRBOLES DE DECISIÓN

Los árboles de decisión pueden expresar cualquier función a partir de los atributos de entrada. Por ejemplo, para funciones Booleanas, cada fila de la tabla de verdad se traslada a un camino del árbol:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



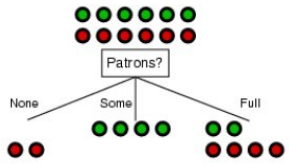
De forma trivial, hay un árbol de decisión consistente para cualquier conjunto de entrenamiento con un camino asociado a cada ejemplo, pero seguramente no será bueno para generalizar nuevos ejemplos. Es preferible encontrar árboles de decisión **más compactos**.

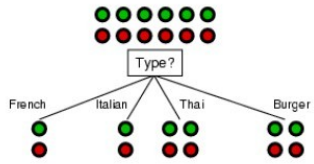
2.2 INDUCCIÓN DE ÁRBOLES DE DECISIÓN

Múltiples formas de inferir el árbol:

- **Trivial**: se crea una ruta del árbol por cada instancia de entrenamiento.
  - Árboles excesivamente grandes.
  - No funcionan bien con instancias nuevas.
- **Óptimo**: el árbol más pequeño posible compatible con todas las instancias (navaja de Ockham).
  - Inviable computacionalmente.
- **Pseudo-optimo (heurístico)**: selección del atributo en cada nivel del árbol en función de la calidad de la división que produce.
  - Los principales programas de generación de árboles utilizan procedimientos similares.

Idea: un buen atributo debería dividir el conjunto de ejemplos en subconjuntos que sean o "todos positivos" o "todos negativos". Patrons = Clientes es una buena elección

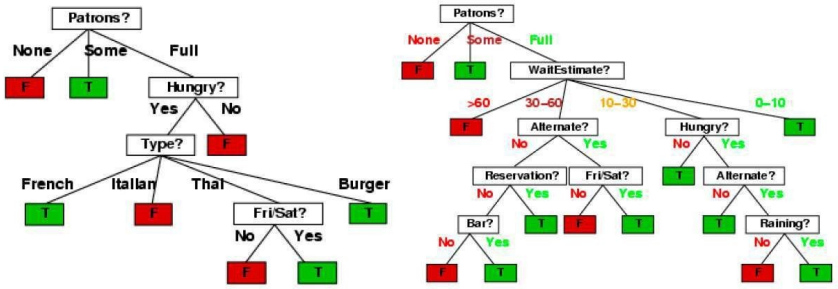




1. No quedan ejemplos: valor por defecto calculado a partir de la mayoría en nodo padre.
2. Todos los ejemplos son positivos o negativos.
3. No quedan atributos: voto de la mayorí de los ejemplos que quedan.
4. Quedan ejemplos positivos y negativos.

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
      best ← CHOOSE-ATTRIBUTE(attributes, examples)
      tree ← a new decision tree with root test best
      for each value v_i of best do
          examples_i ← {elements of examples with best = v_i}
          subtree ← DTL(examples_i, attributes − best, MODE(examples))
          add a branch to tree with label v_i and subtree subtree
      return tree
```

Árbol de decisión obtenido utilizando los 12 ejemplos



Es más simple que el árbol "verdadero".

2.2.1 ELECCIÓN DE LOS ATRIBUTOS DE TEST

Un atributo perfecto divide los ejemplos en conjuntos que contienen solo ejemplos positivos o negativos.

Definir una medida de atributo "bastante adecuado" o "inadecuado".

$I(P(v_1), \dots, P(v_n)) = \Sigma_{i=1} -P(v_i) \log_2 P(v_i)$

Para un conjunto de entrenamiento que contenga p ejemplos positivos y n ejemplos negativos

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Intuición: Mide la ausencia de "homegeneidad" de la clasificación.

Teoría de la Información: cantidad media de información (en bits) necesaria para codificar la clasificación de un ejemplo.

Ejemplos:

- $I([9+, 5-]) = -9/14 \times \log_2 9/14 - 5/14 \times \log_2 5/14 = 0,94$
- $I([k+, k-]) = 1$ (ausencia total de homogeneidad)
- $I([p+, 0]) = I([0, n-]) = 0$ (homogeneidad total)

2.2.2 GANANCIA DE INFORMACIÓN

Entropía esperada después de usar un atributo A en el árbol:

$$resto(A)=\sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i})$$

Ganancia de información esperada después de usar un atributo:

$$Ganancia(A)=I(\frac{p}{p+n},\frac{n}{p+n})-resto(A)$$

Se elige el atributo con mayor valor de G (ID3).

El criterio de ganancia tiene un fuerte sesgo a favor de test con muchas salidas:

Ratio de Ganancia:

$$RGanancia(A)=\frac{Ganancia(A)}{dINFO(A)}$$

con

$$dINFO(A)=-\sum_{i=1}^{v}\frac{p_i+n_i}{p+n}\log_2(\frac{p_i+n_i}{p+n})$$

2.3 VALORACIÓN DE LA CALIDAD DEL ALGORITMO DE APRENDIZAJE

Un algoritmo de aprendizaje es bueno si produce hipótesis que hacen un buen trabajo al predecir clasificaciones de ejemplos que no ha sido observados.

METODOLOGÍA

1. Recolectar un conjunto de ejemplos grande.
2. Dividir el conjunto de ejemplos en dos conjuntos: el conjunto de entrenamiento y el conjunto de test.
3. Aplicar el algoritmo de aprendizaje al conjunto de entrenamiento, generando la hipótesis h.
4. Medir el porcentaje de ejemplos del conjunto de test que h clasifica correctamente.
5. Repetir los pasos del 1 al 4 para conjuntos de entrenamiento seleccionados aleatoriamente para cada tamaño.

RUIDO Y SOBREAJUSTE

Ruido: dos o más ejemplos con la misma descripción (en términos de atributos) pero diferentes clasificaciones.

Sobreajuste: encontrar "regularidades" poco significativas en los datos.

Se dice que una hipótesis h se sobreajusta al conjunto de entrenamiento si existe alguna otra hipótesis h' tal que el error de h es menor que el de h' sobre el conjunto de entrenamiento, pero es mayor sobre la distribución completa de ejemplos del problema (entrenamiento + test).

VALORES PERDIDOS

Asignar el valor **más común entre todos los ejemplos** de entrenamiento pertenecientes al nodo.

Asignar **una probabilidad a cada uno de los posibles valores** del atributo basada en la **frecuencia observada** en los ejemplos pertenecientes al nodo. Finalmente, distribuir de acuerdo a dicha probabilidad.