Intel·ligència Artificial

Cinc Cèntims de IA

Mario Vilar

6 de febrer de 2024

Universitat de Barcelona

Índex

1

Cerca no informada

1.1 Agents

Definició 1.1 (Agents**).** Un agente es una entidad que percibe su entorno a través de sensores y actúa sobre el mismo mediante "actuadores". $f : \mathcal{P} \longrightarrow \mathcal{A}$ tal que $p \longmapsto a$, p una percepció i a una acció.

Definició 1.2 (Agents intel·ligents o amb objectius**).** Planifican antes de actuar. Buscan secuencias de acciones que conducen a estados deseados (objetivos). Etapas de resolución de problemas con objetivos:

1. *Formulación de objetivos:* definir los estados objetivo y los factores que pueden influir su el grado de satisfacción.
2. *Formulación del problema:* decidir qué acciones y estados considerar. Coste individual de las acciones mayor igual que 0.
3. *Búsqueda de la solución:* calcular la (mejor) secuencia de acciones que llevan a algún estado objetivo (a partir del inicial).
4. *Ejecución:* ejecutar las acciones calculadas. Tiene un coste total (aditivo).

Para cualquier clase de entornos y tareas dadas, buscamos el agente (o clase de agentes) con el mejor resultado. Las limitaciones computacionales hacen que la racionalidad perfecta sea inalcanzable.

Exemple 1.3 (Aspirador**).** Percepcions: localització, continguts. $[A, \text{brut}]$. Accions: *esquerra, dreta i aspirar*.

- *Descripció del problema*. Un mundo discreto de dos posiciones (celdas) adyacentes habitado por un robot.
- *Formulació dels objectius*. Se pretende llegar a una situación en la que el mundo esté limpio.
- *Formulació del problema*. El robot puede estar situado en cualquiera de las dos posiciones. Las celdas pueden contener suciedad. Las percepciones del robot le permiten observar su posición y si hay suciedad o no en ella. Las acciones que el robot aspirador puede hacer son:
  - Moverse a la derecha (si hay una celda a su derecha se mueve, si no se queda igual).
  - Moverse a la izquierda (equivalente a derecha).
  - Aspirar la suciedad de su celda (si no hay suciedad, se queda igual).

  Redacció de la funció successor, que donat un número d'estat ens dona el conjunt d'estats següents possibles. De la forma:

$$S(e) = \{\langle a_i, e_i \rangle\},\ e \in \{1, \ldots, 8\}.$$

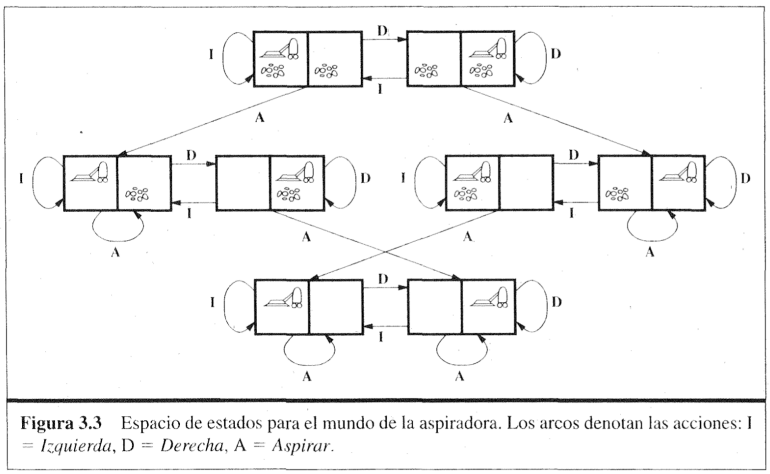

Figura 3.3 Espacio de estados para el mundo de la aspiradora. Los arcos denotan las acciones: I = *Izquierda*, D = *Derecha*, A = *Aspirar*.

Figura 1: Conjunt d'estats de l'aspiradora.

- *Cerca de la solució*. És la seqüència d'accions guanyadora. Suposant que partim de l'estat 1, podria ser, per exemple: {aspirar, dreta, aspirar}.
- *Execució*. Cost de les accions. En aquest cas, 1 unitat per cada i tenim un cost de 3.

Observació 1.4. Todo problema de búsqueda tiene asociado un grafo de estados (vértices). La función sucesor se representa en base a los arcos (edges). En contadas ocasiones se puede construir este grafo en memoria. Se puede llegar al mismo vértice por más de un camino.

```
función BEST-FIRST-SEARCH(problema, f) devuelve una solución o fallo
    nodo ← CREAR-NODO(problema.estado-inicial), alcanzados ← ∅,  frontera ← ∅
    INSERTAR((problema.estado-inicial, nodo), alcanzados)        Lookup table
    INSERTAR(nodo, frontera)                                     Priority queue, ordenada por f
    bucle while not VACIA? (frontera) hacer
        nodo ←SACAR-BORRANDO-PRIMERO(frontera)                   Pop
        si ES-OBJETIVO (nodo.estado)  entonces devolver SOLUCION(nodo)
        bucle for each sucesor en EXPANDIR(nodo, problema) hacer
            s ← sucesor.estado                                   expansión : obtener sucesores
            si s no está en alcanzados o (sucesor.coste-camino < alcanzados[s].coste-camino) entonces
                alcanzados[s]  ←  sucesor
                INSERTAR(sucesor, frontera)
    devolver fallo
```

Figura 2: Best First Search. *Alcanzados* ha de ser un *dict* o bé un *set*.

1.1.1 Breadth First Search (BFS)

1. La búsqueda primero en anchura expande primero el nodo raíz, a continuación se expanden todos los sucesores del nodo raíz, y después el resto. En general, *se expanden todos los nodos a una profundidad en el árbol de búsqueda antes de expandir cualquier nodo del próximo nivel*.

2. La cola FIFO *frontera* pone todos los nuevos sucesores generados al final de la cola, lo que significa que los nodos más superficiales se expanden antes que los nodos más profundos.

La búsqueda primero en anchura (BFS) es óptima cuando todos los costos son iguales, porque siempre expande el nodo no expandido más superficial.

1.1.2 Cerca de cost uniforme

Equivalente a la búsqueda primero en anchura si el coste de todos los pasos es igual. En vez de expandir el nodo más superficial, al búsqueda de costo uniforme expande el nodo n con el camino de costo más pequeño. La *frontera es una cola ordenada por coste del camino*. La búsqueda de costo uniforme no se preocupa por el número de pasos que tiene un camino, pero sí sobre su *coste total*.

Observació 1.5. *El coste de cada paso deberá ser mayor que o* para no quedar atrapado en un bucle infinito al expandir un nodo que tenga una acción de coste cero que vuelva al mismo estado.

1.1.3 DFS (Depth First Search)

La búsqueda primero en profundidad siempre expande el nodo más profundo en la frontera actual del árbol de búsqueda. *La frontera es una pila LIFO*: los nuevos sucesores se colocan al principio.

1.2 Estats i nodes

Observació 1.6.

1. Un estado es una (representación de) una configuración física.
2. Un nodo es una estructura de datos que forma parte de un árbol de búsqueda que incluye estado, padre, hijos, profundidad, coste del camino $g(x)$.
3. *¡Los estados no tienen padres, hijos, profundidad o coste del camino!*

Expandir un nodo consiste en utilizar la función sucesor (función `resultado(s, acción)` del problema) para obtener los estados correspondientes y con ellos crear nuevos nodos.

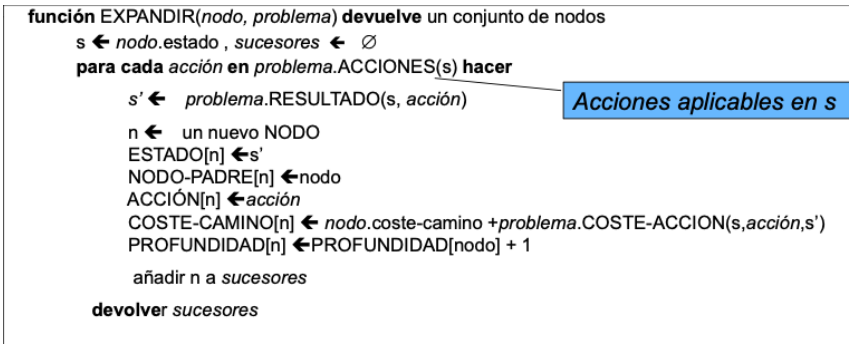

Figura 3: Pseudocodi d'*expand*.

Definició 1.7.

1. *Espacio de estados:* el espacio conceptual en el que se realiza la búsqueda.
2. *Árbol/grafo de exploración:* el que se va creando al realizar efectivamente la búsqueda (representa una parte del anterior)
3. *Frontera:* lista de los nodos visitados que aún no han sido expandidos.
4. *Nodos expandidos:* nodos sacados de la frontera para los que se ha obtenido sus sucesores
5. *Nodos alcanzados:* lista de nodos que han sido creados.

2

Cerca informada

La búsqueda informada mejora a la búsqueda no informada mediante la introducción en el proceso de búsqueda, de información específica del problema que permita acelerar la búsqueda.

Definició 2.1 (Heurística**).** Una heurística $h(n)$ es una *estimación* de lo cercano que se encuentra un nodo al objetivo. Las heurísticas únicamente son válidas para un problema (no son generales).

2.1 Greedy

Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$). La frontera se ordena siguiendo un orden *creciente* de la heurística: se sacan primero los nodos de *menor* heurística. La *función de evaluación*[1] va a ser $f(n) = h(n)$. **Sempre agafem el node de menor heurística, per molt que estiguem explorant una altra branca**. Cal consultar l'algorisme 2.

2.2 A^*

La idea es evitar expandir caminos que ya son caros. La función de evaluación es $f(n) = g(n) + h(n)$:

1. $g(n)$ es el coste coste de alcanzar n *desde el estado inicial*.
2. $h(n)$ es el coste estimado desde n hasta el objetivo, nuestra *heurística*.
3. $f(n)$ es el coste estimado de la mejor solución (del estado inicial al objetivo) que pase por n.

Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of $g(n) + h(n)$. In fact, provided that the heuristic function $h(n)$ satisfies certain conditions, A^* search is both complete and optimal. The algorithm is identical to *uniform-cost-search* except that A^* uses $g + h$ instead of g [RNC22].

A^* usa $h(n)$ para acotar la exploración en el espacio de estados.

[1] An evaluation function (cf. 3.1) is a function used by game-playing computer programs to estimate the value or goodness of a position in a game tree, [Wik23b].

Definició 2.2 (Heurística admisible**).** Una heurística $h(n)$ es admisible si para todo nodo n, $h(n) \leq h^*(n)$, donde $h^*(n)$ es el coste real de alcanzar el estado objetivo desde n.

Una heurística admisible nunca sobre-estima el coste de alcanzar el objetivo, es decir, es optimista. Això és perquè: $g(n)$ is the actual cost to reach n along the current path, and $f(n) = g(n) + h(n)$, we have as an immediate consequence that $f(n)$ never overestimates the true cost of a solution, [RNC22].

Definició 2.3 (Heurística consistent**).** A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$$h(n) \leq c(n, a, n') + h(n').$$

Every consistent heuristic is also admissible. Consistency is therefore a stricter requirement than admissibility, but one has to work quite hard to concoct heuristics that are admissible but not consistent. Si h és consistent,

1. A^* expande todos los nodos a los que se puede llegar desde el estado inicial por caminos que cumplen: $f(n) < C^*$ (cost del camí òptim).
2. A^* puede llegar a expandir nodos en el «contorno del objetivo» ($f(n) = C^*$).
3. A^* nunca expande nodos con $f(n) > C^*$.

Tenim:

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n).$$

$f(n)$ es no decreciente a lo largo de cualquier camino.

Teorema 2.4. *Si $h(n)$ es admisible, la búsqueda A^* usando búsqueda en árbol es óptima (ningún algoritmo con la misma h explorará menos estados).*

Teorema 2.5. *Si $h(n)$ es consistente, la búsqueda A^* utilizando búsqueda en grafos es óptima.*

Definició 2.6 (Problema relaxat**).** Un problema con menos restricciones en las acciones se denomina problema relajado. El coste de una solución óptima a un problema relajado es una heurística admisible del problema original.

Exemple 2.7. Para el 8-puzzle, $h_1(n)$ puede ser el número de cuadrados fuera de sitiom y $h_2(n)$ la distancia total de Manhattan (es decir, número de movimientos desde la posición actual hasta la posición objetivo para cada cuadrado). Si $h_2(n) \geq h_1(n)$ para todo n (siendo ambas admisibles) entonces h_2 domina h_1.

- Si las reglas del 8-puzzle se relajan de forma que un cuadrado se pueda mover a cualquier parte, entonces $h_1(n)$ proporciona la solución más corta.
- Si las reglas se relajan de forma que un cuadrado se puede mover a cualquier cuadrado adyacente, entonces $h_2(n)$ nos da la solución más corta.

3

Joc amb oponents

El objetivo de los algoritmos es calcular una política que recomiende el movimiento a realizar en cada estado. Diversos tipus:

1. Deterministes (un jugador, més d'un jugador).
2. Estocàstics.
3. Informació completa i incompleta.

3.1 Deterministes, un jugador

1. El agente es el que juega.
2. Información perfecta: conocemos las reglas, las acciones y el objetivo.
3. Reinterpretación, cada nodo almacena un valor: el mejor que puede conseguir por esa rama.
4. Tras la búsqueda se pueden escoger los movimientos que llevan al mejor nodo.

3.2 Deterministes, dos jugadors

Juego por turnos, i jocs de *suma zero*: el que un jugador guanya ho perd l'altre (where the total payoff to all players is the same for every instance of the game). Els escacs no són un joc de suma zero.

Definició 3.1 (Funció d'avaluació**).** Función que va de los resultados (estados del mundo) a números reales que representan las preferencias de un agente. **Anomenarem *utilitats* als valors**. En un joc, els valors resumeixen els objectius de l'agent: la utilitat suma un punt si guanya i en resta un si perd.

Teorema 3.2. *Cualquier conjunto de preferencias coherente se puede representar mediante una función de utilidad.*

3.3 Minimax

We first consider games with two players, whom we call MAX and MIN for reasons that will soon become obvious. MAX moves first, and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser. A game can be formally defined as a kind of search problem with the following elements:

1. S_0: The initial state, which specifies how the game is set up at the start.
2. PLAYER(s): Defines which player has the move in a state.
3. ACTIONS(s): Returns the set of legal moves in a state.
4. RESULT(s, a): The transition model, which defines the result of a move.

5. TERMINAL-TEST(s): A terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states.
6. UTILITY(s, p): A utility function (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p.

Observació 3.3. The minimax value of a node is the utility (for MAX) of being in the corresponding state, assuming that both players play optimally from there to the end of the game. Obviously, the minimax value of a terminal state is just its utility. Furthermore, given a choice, MAX prefers to move to a state of maximum value, whereas MIN prefers a state of minimum value.

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Figure 5.3 An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state. The notation $\text{argmax}_{a \in S} f(a)$ computes the element a of set S that has the maximum value of $f(a)$.

Figura 4: Algoritme del minimax, [RNC22]

.

If the maximum depth of the tree is m and there are b legal moves at each point, then the time complexity of the minimax algorithm is $O(b^m)$. The space complexity is $O(bm)$ for an algorithm that generates all actions at once, or $O(m)$ for an algorithm that generates actions one at a time. Completo si el árbol es finito.

3.4 PODA ALFA-BETA

Com reduïm la despesa computacional?

1. Buscar hasta una profundidad determinada. La profundidad es clave.
2. Reemplazar las utilidades de los nodos terminales por una evaluación heurística. Debe parecerse lo más posible a la utilidad de esa posición.

$$\text{Eval}(s) = \sum_{i=1}^{n} w_i f_i(s).$$

3. Si fem això, **la garantía de optimalidad desaparece.**

Unfortunately, we can't eliminate the exponent, *but it turns out we can effectively cut it in half*. The particular technique we examine is called alpha–beta pruning. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision (compute the correct minimax decision without looking at every node in the game tree).

Exemple 3.4. Fixem-nos en la següent iteració d'un exemple de poda alfa-beta:

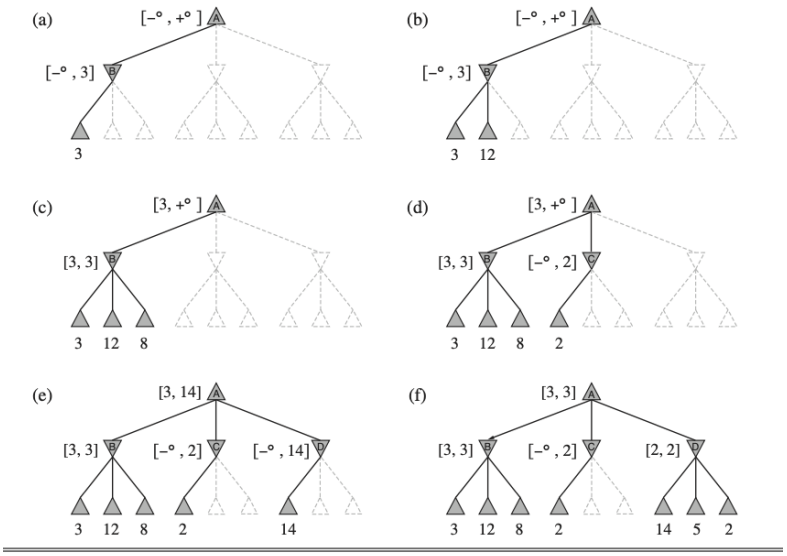

Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successor states, so the value of B is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

Figura 5: Exemple d'una poda alfa-beta.

The value of the root and hence the minimax decision are independent of the values of the pruned leaves x and y, de manera que descartar-les amb una poda alfa-beta és correcte. En efecte:

$$\textsc{Minimax}(root) = \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) = \max(3, \min(2, x, y), 2)$$
$$= \max(3, z, 2) = 3, \;\; z = \min(2, x, y) \leq 2, \;\; \forall x, y.$$

Observació 3.5. Alpha–beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtrees rather than just leaves.

Calculando el mínimo en n (recorriendo los hijos de n), la estimación del valor de n va decrementando. Sea α el valor máximo que MAX puede obtener en cualquier elección a lo largo del camino actual. Si estimación de $n < \alpha$ podemos dejar de expandir los hijos de n, pues tenemos una alternativa mejor. Podemos definir de la misma forma β para MIN.

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

Figura 6: Algorisme de la poda alfa-beta.

Propietat 3.6 (Poda alfa-beta**).**

1. *No se sacrifica la optimalidad.*
2. *Una correcta ordenación de los hijos puede mejorar la cantidad de ramas podadas. De hecho, con una ordenación perfecta se explora el doble de profundidad que sin poda.*

3.5 Expectimax

Many games mirror this unpredictability by including a random element, such as the throwing of dice. We call these *stochastic games*. ¿Qué pasa si no conocemos cuál será el resultado de una acción? (Problema no determinista.)

Exercici 3.7. *Si a cada estudiante le pregunto "qué forma de evaluar el ejercicio prefieres?":*

1. *Elegir por sorteo 12 ejercicios y a esos les pongo un 10 de nota y al resto 5.*
2. *Poner a esos 12 un 8 y al resto un 7.*
3. *Elegir los 12 mejores ejercicios, ponerles un 10 y al resto un 5*
4. *Poner a esos 12 un 8 y al resto un 7.*

$$\textsc{Expectiminimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

Figura 7: Algorisme de l'expectiminimax.

Demostració. El càlcul de l'esperança (o *expected value*, en anglès) és $E(x) = \sum_x x \cdot P(X = x)$, i en el cas que ens ocupa es fa de la següent manera: posem $\Omega = \{ne := \{\text{nota escollida}\}, nne := \{\text{nota no escollida}\}\}$. X es una variable aleatoria, que representa un hecho cuyo resultado desconocemos. $X = \{\text{nota}\}$. Així:

$$E(x) = P(X = ne) \cdot ne + P(X = nne) \cdot nne = P(X = ne) \cdot ne + (1 - P(X = ne)) \cdot nne.$$

Posem que la classe és de 50 alumnes. Si hem fet un mal exercici, clarament no serem escollit entre els 12 millors exercicis. La probabilitat de ser escollit aleatòriament entre 12 persones és de $\frac{12}{50} \approx 0,25$; en canvi, suposem que la probabilitat de ser escollit entre els 12 millors exercicis (si hem fet un bon exercici) és de 0,75. Tenim el següent:

$$\begin{aligned} 0,25 \cdot 10 + 0,75 \cdot 5 &= 6,25. \\ 0,25 \cdot 8 + 0,75 \cdot 7 &= \mathbf{7,25}. \\ 0,75 \cdot 10 + 0,25 \cdot 5 &= \mathbf{8,75}. \\ 0,75 \cdot 8 + 0,25 \cdot 7 &= 7,75. \end{aligned}$$

Per tant, si hem fet un bon exercici escollir la tercera opció; si hem fet un mal exercici, la segona. ■

Observació 3.8. *Principio de máxima utilidad esperada:* Un agente debe seleccionar la acción que maximice su utilidad esperada dado su conocimiento.

Obviously, we still want to pick the move that leads to the best position. However, positions do not have definite minimax values. Los nodos del «enemigo» en lugar de calcular el mínimo calcula la esperanza (el valor medio).

This leads us to generalize the minimax value for deterministic games to an expectiminimax value for games with chance nodes. Terminal nodes and MAX and MIN nodes (for which the dice roll is known) work *exactly the same way as before*. For chance nodes we *compute the expected value*.

Definició 3.9 (Distribució de probabilitat**).** Una *distribución de probabilidad* es una asignación de pesos a los diferentes eventos del hecho. A medida que tenemos más información, las probabilidades cambian (condicionadas).

Observació 3.10 (Freqüentista vs. Bayes**).**

1. Visión frecuentista: Medias sobre experimentos repetidos. Se estiman a partir de observaciones históricas. Nos hace pensar en hechos inherentemente aleatorios como lanzar dados.

2. Visión Bayesiana: Grados de creencia sobre variables no observadas (en base al conocimiento). Se pueden aprender a partir de la experiencia, ya que las nuevas experiencias modifican nuestras creencias.

Definició 3.11. Sigui $f(X)$ una funció d'una variable aleatòria X. El valor esperado de una función es su valor medio, ponderando cada valor de su entrada por la distribución de probabilidad.

$$E_{P(X)}[f(X)] = \sum_x f(x)P(x).$$

En minimax el valor de las funciones de evaluación no importa, tan sólo es importante el orden relativo (si un estado es mejor que otro o no). Para expectiminimax necesitamos además que las magnitudes de los valores sean correctas.

Observació 3.12 (Expectimax i poda alfa-beta**).** It may have occurred to you that something like alpha–beta pruning could be applied to game trees with chance nodes. It turns out that it can. The analysis for MIN and MAX nodes is unchanged, but we can also prune chance nodes. If we put bounds on the possible values of the utility function, then we can arrive at bounds for the average without looking at every number. For example, say that all utility values are between −2 and +2; then the value of leaf nodes is bounded, and in turn we can place an upper bound on the value of a chance node without looking at all its children.

3.6 Multijugador

Similar al minimax:

1. Las utilidades son ahora tuplas.
2. Cada jugador maximiza su propia entrada y propaga el resultado al siguiente nivel.
3. Diplomacy game.

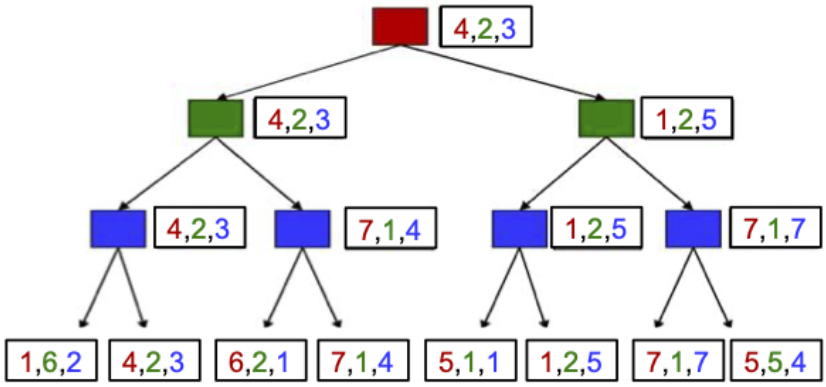

Figura 8: Exemple d'un multijugador.

4

Història de la IA

Definició 4.1 (Intel·ligència Artificial**).** Estudio de *agentes inteligentes* capaces de percibir su entorno y tomar medidas (acciones) que maximicen sus objetivos. Quatre corrents de pensament:

	Estàndard humà	Racionalitat
Raonament	Sistemas que piensan como humanos. *Enfoque cognoscitivo*	Sistemas que piensan racionalmente. *Enfoque lógico*.
Comportament	Sistemas que actúan como humanos. *Prueba de Turing*[2]	Sistemas que actúan racionalmente. *Enfoque de agentes racionales*.

Taula 1: Segons la literatura (taxonomia).

Alan Turing sugirió los principales componentes de IA: *representación del conocimiento, razonamiento, comprensión del lenguaje, aprendizaje*. El test de Turing no es reproducible, constructivo, o susceptible al análisis matemático.

Definició 4.2 (IA, segons [CDT19]**).** Sistemas que muestran un comportamiento inteligente analizando su entorno y realizando acciones —con cierto grado de *autonomía*— para conseguir objetivos específicos. Los sistemas basados en IA pueden estar:

1. basados en SW, actuando en el mundo virtual;
2. incrustados en dispositivos de HW.

Propietat 4.3 (Requirements for trustworthy AI, [CDT19]**).**

1. *Human agency and **oversight**. Including fundamental rights, human oversight.*
2. *Technical robustness and **safety**. Resilience to attack and security, general safety, accuracy, reliability and reproducibility.*
3. ***Privacy** and data governance. Including respect for privacy, quality and integrity of data, and access to data.*
4. ***Transparency**. Including traceability, explainability and communication*
5. *Diversity, **non-discrimination** and fairness. Including the avoidance of unfair bias, accessibility and universal design, and stakeholder participation.*
6. *Societal and environmental **wellbeing**. Including sustainability social impact, society and democracy.*
7. ***Accountability**. Including auditability, minimisation and reporting of negative impact.*

Se pueden interpretar en términos de los valores éticos que afectan.

[2] Test operativo para evaluar el *pensamiento* inteligente: el juego de imitación. Herramienta de evaluación de la capacidad de una máquina para exhibir un *comportamiento* inteligente similar al de un ser humano o indistinguible de este [Wik23a].

Utilizamos el término racional de una forma concreta:

1. Racional: aquel que *maximiza* unos objetivos establecidos.
2. Las decisiones son racionales o no independientemente de los procesos mentales que las generen.
3. Los objetivos se expresan mediante una *función de utilidad*.
4. Ser racional significa maximizar la *utilidad esperada*.

En ocasiones calcular la mejor decisión costaría meses de cómputo. Necesitamos racionalidad limitada.

Definició 4.4 (Agent autònom**).** Ente artificial capaz de actuar (tomando decisiones para conseguir sus objetivos) sin intervención humana.

Les aplicacions de la IA són en diversos camps: la sanitat, l'automoció, les finances, els videojocs, la publicitat i el món militar, art (openAI). Els fonaments de la IA han estat diversos àmbits, en aspectes concrets:

1. *Filosofia*. Lógica, razonamiento. Mente como sistema físico. Aprendizaje, racionalidad.
2. *Matemàtiques*. Formalidad y pruebas. Probabilidad. Algoritmos, computación, (in)decisión, (in)flexibilidad.
3. *Psicologia*. Adaptación. Fenómenos de percepción y control motor. Técnicas experimentales.
4. Economia. Teoría formal de las decisiones racionales.
5. Lingüística. Representación del conocimiento. Gramática.
6. Neurociència. Sustrato físico plástico para la actividad mental.
7. Teoria de control. Sistemas homeostáticos, estabilidad. Diseños de agente óptimo simples.

Count	Esdeveniment		
	Any	*Nom*	*Explicació* (si escau)
1	1956	Conferència de Dartmouth	On es parla primer de IA.
2	60s	Invierno de la IA	Significado en las traducciones.
3	70s		Micro-mundos: Eliza (psicoterapeuta).
4	80s		Sistemas expertos: diagnóstico.
5	90s-00s	Agentes Inteligentes	Robótica de consumo.
6	1997	Deep Blue (IBM)	Guanya al campió d'escacs.
7	2005	DARPA Grand Challenge	Stanford, cotxe autònom.
8	2011	Watson (IBM)	Guanya Jeopardy.
9	2016	Alpha Go (Google)	Guanya al campió de Go.
10	2020	AlphaFold	Guanya el Casp Challenge, proteïnes.
11	2023	ChatGPT	Boom de les IA generatives.

Taula 2: Esdeveniments importants en la gestació de la IA.

Definició 4.5 (Deep Learning**).** És una tècnica d'extracció i transformació de noves característiques del processament de la informació, les quals poden ser de forma supervisada o no. Són algoritmes que funcionen en un sistema per capes, simulant el funcionament bàsic del cervell que s'utilitza amb les neurones [Viq23]. La seva història de manera molt succinta:

1. Té els seus orígens el 1940 amb el sorgiment de les NN (Neural Networks).
2. Als anys 80 comença a reconèixer codis postals escrits a mà.
3. *Algorismes Speech-to-text*. És una part de la intel·ligència artificial que té com a objectiu permetre la comunicació parlada, capaç de processar el senyal de veu i reconèixer la informació que porta.
4. *Recurrent Long-Short Term Memories*. És una tècnica per a millorar el desenvolupament de la intel·ligència artificial formada per xarxes neuronals recurrents (RNN).

Referències

[CDT19] European Commission, Content Directorate-General for Communications Networks i Technology. *Ethics guidelines for trustworthy AI*. Publications Office, 2019. doi: doi/10.2759/346720.

[RNC22] Stuart J. (Stuart Jonathan) Russell, Peter Norvig i Ming-Wei Chang. *Artificial intelligence : a modern approach*. eng. Fourth Edition, Global Edition. Pearsons series in artificial intelligence. Harlow: Pearson Education Limited, 2022. isbn: 9781292401133.

[Viq23] Viquipèdia. *Aprenentatge profund — Viquipèdia, l'Enciclopèdia Lliure*. [Internet; descarregat 31-octubre-2023]. 2023. url: %5Curl%7B//ca.wikipedia.org/w/index.php?title=Aprenentatge_profund&oldid=32624467%7D.

[Wik23a] Wikipedia. *Prueba de Turing — Wikipedia, La enciclopedia libre*. [Internet; descargado 11-octubre-2023]. 2023. url: %5Curl%7Bhttps://es.wikipedia.org/w/index.php?title=Prueba_de_Turing&oldid=154540713%7D.

[Wik23b] Wikipedia contributors. *Evaluation function — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Evaluation_function&oldid=1160711683. [Online; accessed 4-November-2023]. 2023.