

DESARROLLO DE JUEGOS CON INTELIGENCIA ARTIFICIAL

Índice

BLOQUE I

- 0. Introducción IA**
- 1. IA, agentes inteligentes y juegos**
- 2. Búsqueda no-informada**
- 3. Búsqueda con heurísticas débiles**
- 4. Búsqueda con heurísticas fuertes**
- 5. Búsqueda multiagente**

Javier Castro Magro

Tema 0. – Introducción

¿QUÉ ES LA INTELIGENCIA ARTIFICIAL?

Es el conjunto de áreas de la ciencia (matemáticas, informática, ...) que estudian y desarrollan técnicas que tienen como objetivo producir sistemas capaces de comportarse tal y como lo haría un humano en un ámbito concreto.

¿ES EL FIN DE LA IA SUSTITUIR AL HUMANO?

Hay muchos casos, menos fantasiosos, donde un humano no puede vivir en el entorno en el que hay que trabajar (espacio, agua, reactor...) procesar toda la información suficientemente rápido para decidir.

O simplemente no está disponible para echar una partida.

DISCIPLINAS DE LA IA



La IA tiene diferentes disciplinas o áreas de estudio que cubren la mayor parte de las actividades propias del ser humano.

En este curso nos centraremos en dos técnicas muy empleadas en videojuegos:

- **Búsqueda en el espacio de estados (Planificación)**
- **Aprendizaje automático**

DEDUCCIÓN, RAZONAMIENTO Y RESOLUCIÓN DE PROBLEMAS

Esta disciplina estudia la forma en que los humanos realizamos **razonamiento y deducción, y los empleamos para resolver problemas**, y como replicar estos mecanismos en los computadores. Algunos casos de uso:

- Sistemas de **diagnóstico médico**.
- Sistemas de **resolución de problemas matemáticos**.
- **Asesoramiento legal** automático.

REPRESENTACIÓN DEL CONOCIMIENTO

Estudia la **forma en que se representan la información y el conocimiento** y cómo hacerlo de forma que una inteligencia artificial pueda usarlos con facilidad. Representaciones habituales incluyen:

- **Grafos**
- **Ontologías**
- **Redes semánticas**

PLANIFICACIÓN

Tiene como objetivo la producción de planes para ser ejecutados a través de un agente, teniendo en cuenta un estado inicial, un estado final y un conjunto de acciones posibles. Por ejemplo:

- **Resolver el juego de las Torres de Hanoi**
- **Encontrar un camino a través de un laberinto**
- **Jugar a las damas**

PERCEPCIÓN Y VISIÓN POR COMPUTADOR

Estudia como dotar a los sistemas inteligentes de la **capacidad de entender el mundo que los rodea a través de sensores y datos**. Incluye tareas como el reconocimiento de imágenes, la segmentación semántica de estas, pero también el procesamiento de otras señales como señales de audio y otros sensores.

APRENDIZAJE MÁQUINA

Desarrolla algoritmos que tienen la capacidad de aprender a resolver tareas a partir de los datos, sin necesidad de programarlos explícitamente. Algunas de las áreas de estudio:

- Aprendizaje supervisado
- Aprendizaje no supervisado
- Aprendizaje por refuerzo

ROBÓTICA

Combina otras disciplinas de la inteligencia artificial con las máquinas físicas (robots) para darles la capacidad de interactuar y manipular su entorno. Incluye tanto el aspecto físico (motores, sensores) como el software necesario para crear robots inteligentes. Algunos tipos de robots:

- Asistentes personales
- Robots guías
- Robots

PROCESAMIENTO DEL LENGUAJE NATURAL

Estudia las interacciones entre las computadoras y el lenguaje humano. Permite a las máquinas entender, interpretar y generar lenguaje humano.

- Traducción
- Análisis de sentimientos
- Asistentes de voz
- ChatGPT, Midjourney, Dall-E, Microsoft Bing Search, ...

INTELIGENCIA SOCIAL

Estudia la capacidad de los sistemas de inteligencia artificial de entender e interactuar con grupos de personas y causar impacto real en la sociedad. Algunos casos de uso:

- Gestión de catástrofes ambientales (Huracán Ian – 2018)
- Gestión de pandemias (COVID 19)
- IA para el bien social (Standford Data Science for Social Good)

LA IA EN LA INDUSTRIA DEL VIDEOJUEGO

Controlador de agentes enemigos → Sustituir al contrincante

– Los primeros enemigos tenían un patrón fijo de movimiento

La dificultad estaba en la desventaja en número

– Al aumentar la complejidad de los juegos (1^a y 3^a persona, estrategia, ...) aumenta la necesidad de un comportamiento más inteligente

Movimiento propio según el terreno, acciones lógicas

Controlador de agentes cooperantes → Sustituir al compañero

– Aparece con la evolución de la narrativa de los juegos

El jugador puede interactuar con otros personajes que no participan en la acción (NPC, non-player characters) que le pueden ayudar, dar pistas, etc.

– Aún más elaborado, si el cooperante sigue a nuestro personaje. En caso contrario el jugador tendrá la sensación de ser su “niñera”

Por ej. en Medal of Honor, Uncharted, ...

Adaptación de la dificultad → Sustituir al game master

– Un vj. debe mantener un nivel de entretenimiento.

Muy fácil = aburrimiento = Muy difícil

– Pero la dificultad es diferente para cada jugador

Aumentar el realismo

– En juegos de carreras, para simular el tráfico y los peatones

– En juegos de deportes, para recrear el ambiente del estadio/campo

Interfaces de jugador más intuitivos o realistas

– Detección de movimiento

– Órdenes mediante voz

– Ondas cerebrales

REQUISITOS DE LA IA EN LA INDUSTRIA DEL VIDEOJUEGO

FUN

Son más **laxos** que en otras aplicaciones ya que no son ni sistemas críticos ni se toman decisiones relevantes para un colectivo de personas.

Además, una IA “muy buena” puede hacer imbatible al ordenador (p.ej. en ajedrez)

SERIOUS

Pueden llegar a ser aún más **exigentes** que otras aplicaciones de IA como reconocimiento facial/voz, ...

Investigación muy actual.



CONCLUSIÓN

La IA es un elemento clave de cualquier videojuego.

Las herramientas de desarrollo integradas (Unreal Engine, Unity, ...) incorporan diferentes algoritmos.

En esta asignatura se presentan una serie de técnicas fundamentales.

- Un **diseñador** debe conocer qué limitaciones tienen los algoritmos
- Un **desarrollador** debe conocer los algoritmos incorporados en las herramientas
- Un **ingeniero** debe conocer las limitaciones, las posibilidades de las herramientas y cómo construir nuevas técnicas para integrarlas en los motores del vj.

PARA SABER MÁS

Búsqueda y Agentes

- A* para salir de un laberinto --> <https://www.youtube.com/watch?v=qXZt-B7iUyw>

Aprendizaje Automático

- AI Learns to Walk --> https://www.youtube.com/watch?v=L_4BPjLBF4E
- OpenAI – Escondite --> <https://www.youtube.com/watch?v=Lu56xVlZ40M&t=118s>
- NVIDIA - Learning to fight --> <https://www.youtube.com/watch?v=1kV-rZZw50Q>

Machine Learning – Deep Learning

- IA character controller --> <https://www.youtube.com/watch?v=Ul0Gilv5wvY>

IAs generativas

- IAs generativas --> <https://www.youtube.com/watch?v=3eMmmj3roOs>

Tema 1. – IA, agentes inteligentes y juegos

INTELIGENCIA ARTIFICIAL: HISTORIA

HISTORIA ANTIGUA

Efeso, dios de la metalurgia y sus robots de oro



III A.d.C, Ctebisio y su reloj de agua automático



Siglos VIII – XI primeros artefactos mecánicos (musicales)

Siglos XV – XVI Golems judíos para defender el guetto de Praga (Judah Loew)

PRIMERA MITAD DEL SIGLO XX



1900/20

- Bertrand Russel y Alfred North Whitehead desarrollan sus Principia Mathematica que revolucionan la lógica formal.

1937

- Alan Turing publica “Números computables”, base de la teoría de la computación moderna.

1940

- Konrad Zuse fabrica la primera computadora programable que se comercializó, la Z4.
- Warren McCulloch y Walter Pitts describen matemáticamente las primeras neuronas artificiales.
- John von Newman y Oskar Morgenstern publican “Teoría de los Juegos y el comportamiento económico”, donde sientan las bases de la teoría de juegos.

SEGUNDA MITAD DEL SIGLO XX

1950

Primeros programas de IA funcionales (ajedrez, damas, resolución de problemas geométricos)

Arthus Samuel (IBM) desarrolla el primer programa que “aprende” a jugar a las damas. Semilla del machine learning.

Aparece el término inteligencia artificial. Se desarrolla el Perceptrón.

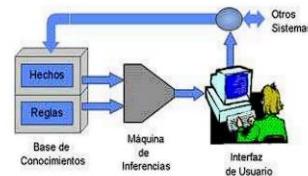
1960/70

Primeras etapas de las redes de neuronas.

Primeros sistemas expertos (Dendral, Prospector, Mycin)

Declive de la computación neuronal (análisis de los Perceptrones de Minsky)

Un Sistema Experto es un sistema que emplea conocimiento humano capturado en un ordenador para resolver problemas que normalmente resolverían humanos expertos.



Esquema de un Sistema Experto

1980

Aplicaciones comerciales de los sistemas expertos

Proyecto de software de “quinta generación” en Japón (fracaso).

1990

Regreso de las redes de neuronas.

Primeros pasos en reinforcement learning.

Deep Blue bate a Kasparov (1997).

SIGLO XXI**2000**

Resurgimiento de la IA.

Modelos de incertidumbre (cadenas de Markov, redes Bayesianas)

Agentes inteligentes (robots autónomos, sistemas multiagente)

Primer coche autónomo (Google).

2010 -> IA como la conocemos hoy en día

Avance de las Deep Neural Networks

2012 – Alex Net gana el ImageNet Large Scale Visual Recognition bajando el porcentaje de error del 30% al 15%. Marca un punto de inflexión, usando GPUs para realizar Deep Learning.

2015 – AlphaGo bate a Fan Hui profesional del Go 2º Dan.

2019 – AlphaStar se convierte en “Grandmaster” en Starcraft 2, batiendo al 99.9% de los jugadores humanos.

2020 – Presentación de GPT-3

2022 – Primera reléase de ChatGPT

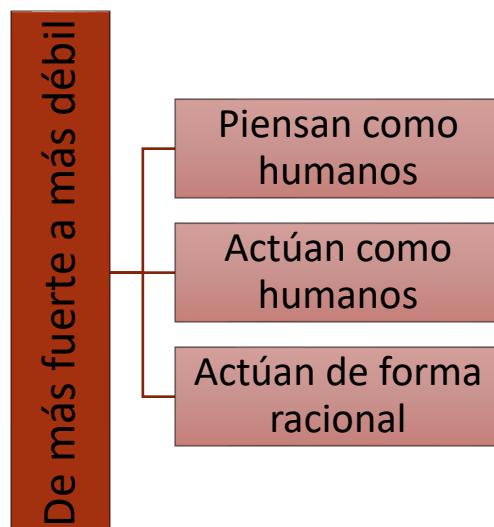
INTELIGENCIA ARTIFICIAL DÉBIL VS FUERTE



¿CHATGPT QUÉ TIPO DE INTELIGENCIA ERES, FUERTE O DÉBIL?

Soy una inteligencia artificial "débil" o "estrecha" (Weak AI). Esto significa que estoy diseñado para realizar **tareas específicas**, como responder preguntas y generar texto en lenguaje natural, pero **no poseo** conciencia, **autoconciencia** ni la capacidad de comprender o razonar independientemente. Mi funcionamiento se basa en patrones de lenguaje y datos estadísticos, y mi conocimiento se limita a la información disponible hasta mi fecha de conocimiento en septiembre de 2021. No soy una inteligencia artificial "fuerte" o "general" que posea una inteligencia generalizada y capacidad de aprendizaje autónomo en diversos dominios.

ALGUNAS DEFINICIONES



SISTEMAS QUE PIENSAN COMO HUMANOS

Capacidad de pensamiento general y **autoconsciencia**. Comprender y razonar, entender el mundo, **adquirir nuevos conocimientos** de forma autónoma.

SISTEMAS QUE ACTÚAN COMO HUMANOS

Capacidad de realizar tareas de forma **similar a como los humanos** las realizamos (a veces incluso mejor). No implica comprensión ni autoconsciencia. Pero dentro de un contexto, puede parecer inteligencia humana.

SISTEMAS QUE ACTÚAN DE FORMA RACIONAL

Capacidad de realizar tareas de forma **coherente y lógica**, normalmente basándose en reglas y algoritmos. Tampoco implica comprensión ni autoconsciencia.

PENSAR COMO HUMANOS

Si hablamos de pensar igual que los humanos

- Estamos en el campo de la IA fuerte.
- Hablamos de AGI (Artificial General Intelligence)
- Es ciencia ficción a corto/medio plazo.
- Pero... ¿cómo piensan los humanos?



Si hablamos de simular que piensan como los humanos

- Entramos en el campo de la IA débil.
- ChatGPT, Midjourney, generadores de contenido, asistentes de salud mental, ...
- El test de Turing busca determinar si una IA está en esta área...



ACTUAR COMO HUMANOS**IAs que actúan como humanos**

- Seguimos en el campo de la IA débil.
- Diseñadas para realizar acciones y/o tareas específicas.
- Pueden imitar el comportamiento humano en situaciones concretas.
- En ocasiones pueden realizar sus tareas mejor que los humanos.
- Sistemas de reconocimiento de imágenes, traductores automáticos, robots de servicio, ...

En videojuegos

- NPCs en juegos como los SIMs, GTA V, Red Dead Redemption 2, ...

Ej del RDR II --> <https://www.youtube.com/watch?v=ai8g4hFDHkA>

ACTUAR RACIONALMENTE**IAs que actúan racionalmente**

- Seguimos en el campo de la IA débil.
- Actúan de forma racional para resolver una tarea concreta.
- Aplican reglas y algoritmos lógicos para resolver la tarea para la que han sido programados.
- Sistemas de conducción autónoma, planificadores de rutas (GPS), sistemas de recomendación (Netflix), robots en fábricas o Amazon...

En videojuegos

- Sistemas de juego como el ajedrez, las damas o el Go.
- Videojuegos deportivos como FIFA, Formula 1 2023 o simuladores como Euro Truck Simulator 2
- Videojuegos de estrategia como Starcraft

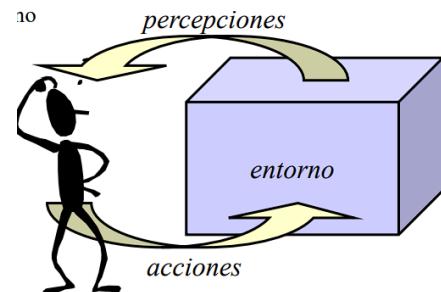
Ej Euro Truck Simulator -->

https://www.youtube.com/watch?si=2_MHtlygl7n7nJ8d&v=bCdNLrmNAvQ&feature=youtu.be

AGENTES

Agente:

- ente activo embebido en un entorno
- “cuerpo”:
 - Percibe el entorno por medio de sensores
 - Actúa sobre el entorno por medio de efectores
- “mente”:
 - Determina las acciones a partir de las percepciones
 - Medida de rendimiento que guía dicho proceso



TIPOS DE AGENTES

AGENTES NATURALES

- Cuerpo biológico y entorno natural
- Sensores: ojos, oídos, lengua, etc.
- Efectores: piernas, brazos, manos, etc.
- Medida de rendimiento: sobrevivir, reproducirse, ...

AGENTES ARTIFICIALES

Agentes hardware (robots):

- interactúan directamente con un entorno físico
- disponen de un “cuerpo” físico
- sensores: cámaras, telémetros infrarrojos, etc.
- efectores: ruedas/piernas, manipuladores, etc.

Agentes software (softbots):

- actúan en entornos virtuales (p.e. Internet)
- todo software: no necesitan manipular físicamente el entorno
- sensores y efectores: dependientes del entorno

AGENTE INTELIGENTE

Actúan de forma racional en su entorno

Determinantes de un comportamiento racional:

- Medida de rendimiento: define el grado de éxito del agente
- Secuencia de percepciones: la experiencia del agente
- Conocimientos a priori sobre su entorno
- Capacidades: las acciones que el agente pueda emprender

Comportamiento racional:

- A partir de la secuencia de percepciones hasta el momento, y el conocimiento a priori sobre el entorno
- Elegir entre las capacidades la acción que maximice la medida de rendimiento

Racionalidad ≠ Omisciencia

- La selección racional de acciones sólo se basa en la información disponible

AUTONOMÍA**Problema:**

- Los conocimientos a priori compilan la “inteligencia” del diseñador
- Un agente que no presta atención a sus percepciones
 - no sería inteligente
 - sólo podría actuar en entornos extremadamente simples
 - no puede actuar con éxito en situaciones no anticipadas

Autonomía:

- “no bajo el control inmediato de una persona”
- un agente es más autónomo...
 - ... cuanto más se rige su comportamiento por su propia experiencia
 - ... cuanto menos depende de sus conocimientos a priori

Agente inteligente = comportamiento racional + autonomía

PROGRAMA Y ARQUITECTURA DE AGENTE**Programas de Agente:**

- Software que determina el comportamiento del agente
- Implementa la función percepción-acción

{simple agent program}

memory \leftarrow perceive(memory, percept)

action \leftarrow action-selection(memory, performance-measure)

memory \leftarrow act(memory, action)

Arquitectura de agente:

- Los módulos que componen el agente
- Estructura el programa de agente
- Partes imprescindibles:
 - componente de percepción
 - componente de selección de acciones
 - componente de acción

ENTORNO DE JUEGOS

- Construir agentes inteligentes es más difícil cuanto más complejo sea el entorno en el que actúan
- Juegos:
 - Entornos bien definidos:
 - El progreso del juego se puede concebir como una secuencia de estados
 - Las reglas del juego definen cómo puede evolucionar el estado del juego
 - En la mayoría de los estados, el grado de complejidad del entorno impide al agente (jugador) determinar acciones (jugadas) óptimas
 - Juegos unipersonales, bipersonales, y de múltiples jugadores
 - Juegos de estrategia, juegos de acción, juegos de realidad aumentada, ...
 - ¿Qué características determinan si un entorno/juego es más o menos complejo?

PROPIEDADES

- **Accesible frente a inaccesible:**

- ¿El agente puede determinar inequívocamente el estado de su entorno?
 - Accesible: Ajedrez, damas, “tres en raya”
 - Inaccesible: póker, laberinto desconocido, Pacman (comecocos)

- **Determinista frente a no determinista:**

- ¿Las acciones del agente en un estado actual determinan completamente el estado resultante?
 - Determinista: ajedrez, damas, laberinto
 - No determinista: Piedra-papel-tijera, Pacman (comecocos)

- **Estático frente a dinámico:**

- ¿El estado del entorno pueda cambiar mientras que el agente delibera? ¿Puede cambiar sin que el agente actúe?
 - Estático: laberinto (entorno no cambia)
 - “Semidinámico”: ajedrez (cambios previsibles)
 - Dinámico: videojuegos (cambios imprevisibles)

- **Discreto frente a continuo:**

- ¿Los conjuntos de posibles percepciones y/o acciones son discretos?
- Discreto: ajedrez, labirinto, Pacman (comecocos)
- Continuo: juegos de realidad aumentada

Tema 2. – Búsqueda no-informada

AGENTES BASADOS EN BÚSQUEDA

2.1 BÚSQUEDA EN ESPACIOS DE ESTADOS

ENTORNOS SIMPLES

Características de entornos simples (aka: problemas bien definidos)

- **discreto:**

se puede concebir el mundo en estados

en cada estado hay un conjunto finito de percepciones y acciones

- **accesible:** el agente puede acceder a las características relevantes del entorno

puede determinar el estado actual del mundo

puede determinar el estado del mundo que le gustaría alcanzar

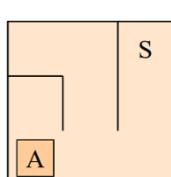
- **estático y determinista:** no hay presión temporal ni incertidumbre

el mundo cambia sólo cuando el agente actúa

el resultado de cada acción está totalmente definido y previsible

EJEMPLOS DE ENTORNOS SIMPLES

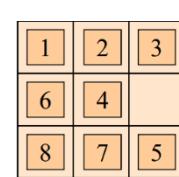
Juegos sin contrario (pasatiempos) --> **El agente es el único jugador**



Laberinto



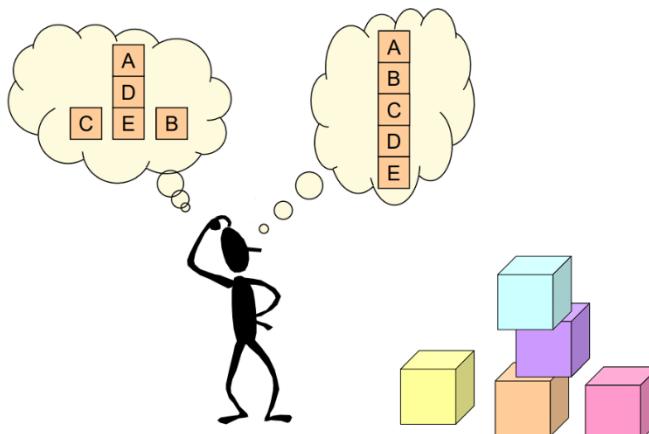
n-reinas



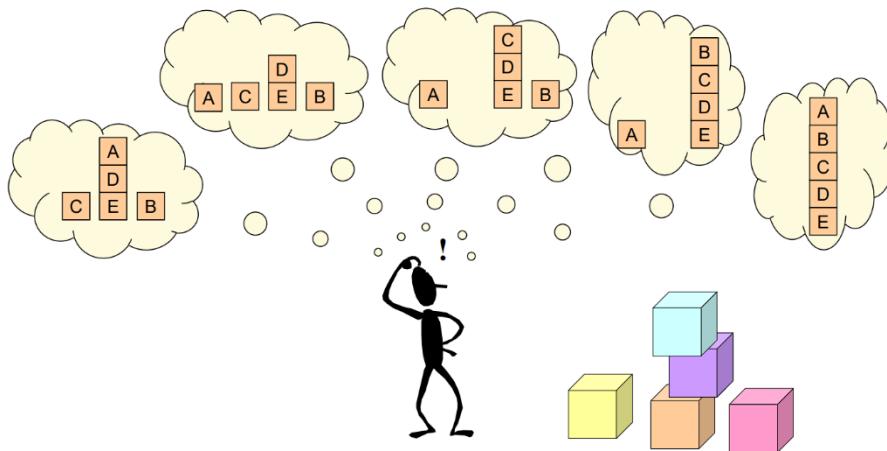
n-puzzle

AGENTES INTELIGENTES PARA ENTORNOS SIMPLES

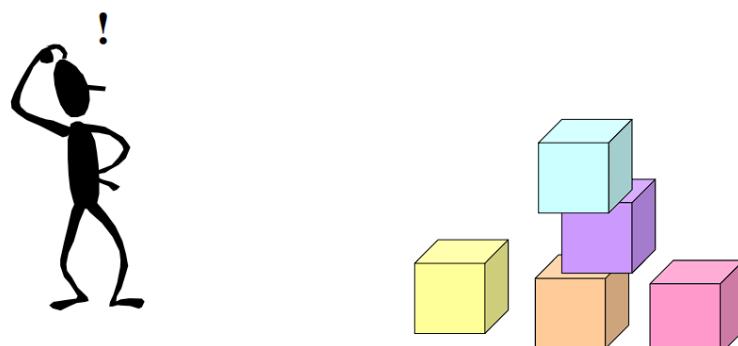
Desean modificar el estado del entorno de acuerdo con sus objetivos.



Con tal fin, anticipan los efectos esperados de sus acciones sobre el modelo, – generando planes de actuación en el modelo...



... antes de ejecutar las acciones del plan en el entorno real



RESOLVIENDO ENTORNOS SIMPLES

Ejemplo: Torres de Hanoi

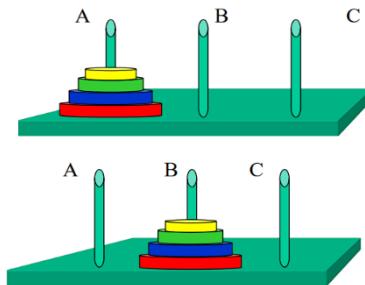
Objetivo:

- Trasladar los discos de la aguja A a B en el mismo orden

Restricción:

- un disco mayor nunca debe reposar sobre uno de menor tamaño

¿Cómo escribir el programa de agente correspondiente?



SOLUCIÓN 1: TABLAS DE ACTUACIÓN

Tablas de actuación específicos del problema:

- para **cada situación** hay una entrada en una tabla de actuación; dicha entrada compila la secuencia de acciones a emprender:

cuatro discos en A ⇒

disco 1 de A a C / disco 2 de A a B / disco 1 de C a B / disco 3 de A a C / disco 1 de B a A / disco 2 de B a C / disco 1 de A a C / disco 4 de A a B / disco 1 de C a B / disco 2 de C a A / disco 1 de B a A / disco 3 de C a B / disco 1 de A a C / disco 2 de A a B / disco 1 de C a B

- Problemas:

- debemos conocer la solución a priori.
- es una solución específica para este problema y esta dimensión.
- escala mal. n^2-1

SOLUCIÓN 2: ALGORITMOS ESPECÍFICO

Algoritmos específicos del problema:

- El diseñador del agente conoce un método para resolver el problema
- Codifica este método en un algoritmo particular para el problema
- Mejorar la flexibilidad: parametrizar el algoritmo
- Problema: el diseñador ha de anticipar todos los escenarios posibles
- Los entornos reales suelen ser demasiado complejos como para anticipar todas las posibilidades

```
PROCEDURE MoverDiscos(n:integer; origen,destino,auxiliar:char);  
  
{ Pre: n > 0  
  
Post: output = [movimientos para pasar n discos de la aguja origen a la aguja destino] }  
  
BEGIN IF n = 0 THEN {Caso base} writeln ELSE BEGIN {Caso recurrente} MoverDiscos(n-1,origen,  
auxiliar,destino);  
  
write('Pasar disco',n,'de',origen,'a',destino);  
  
MoverDiscos(n-1, auxiliar,destino,origen);  
  
END; {fin ELSE}  
  
END; {fin MoverDiscos}
```

SOLUCIÓN 3: ALGORITMO GENÉRICO

Métodos independientes del problema:

- generamos una abstracción (modelo simbólico) del problema

- eliminamos los detalles de la representación del problema dejando los elementos mínimos que permitan resolverlo.

- algoritmo genérico:

- genera una solución a cualquier problema representado mediante el modelo simbólico.

- mayor flexibilidad:

- el diseñador no necesita conocer la solución de antemano
 - es más fácil adaptar el método a nuevas características del problema

Aquí surgen los algoritmos de búsqueda

BÚSQUEDA EN ESPACIOS DE ESTADOS

Problema de búsqueda: proceso para decidir la secuencia de acciones más eficiente que se debe seguir para alcanzar la situación final partiendo de la situación inicial.

Espacio de estados: modelo del mundo representado por un grafo

mundos	modelo	representación
<ul style="list-style-type: none"> • situación • situación inicial • situación final • acción • esfuerzo • secuencia de acciones 	<ul style="list-style-type: none"> • estado • estado inicial • estado meta • operador • coste • plan 	<ul style="list-style-type: none"> • nodo • nodo inicial • nodo final/meta • arista/arco • valor de la arista • camino

secuencia más eficiente == plan más eficiente == camino de menor coste

EJEMPLO: N-PUZZLE

Problema:

Estados:

– Tablero compuesto por n casillas numeradas + 1 vacía. El tablero se divide en raíz cuadrada de $(n + 1)$ filas y columnas.

Acciones:

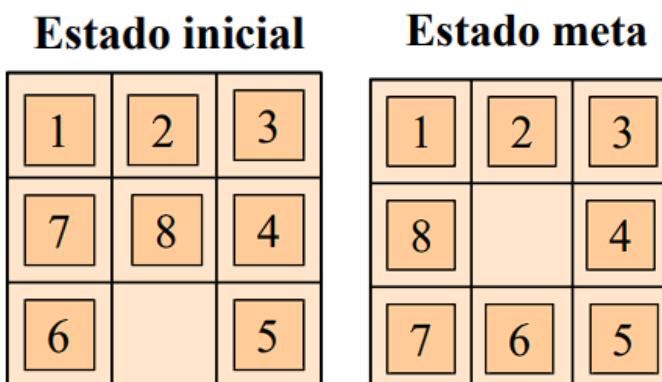
- mover pieza adyacente a la posición de la vacía
- de 2 a 4 operadores aplicables, según el estado

Coste:

- Cada acción tiene el mismo coste

Solución:

- secuencia de acciones que lleva de un estado inicial al estado meta



EJEMPLO DE MODELIZACIÓN: 3-PUZZLE

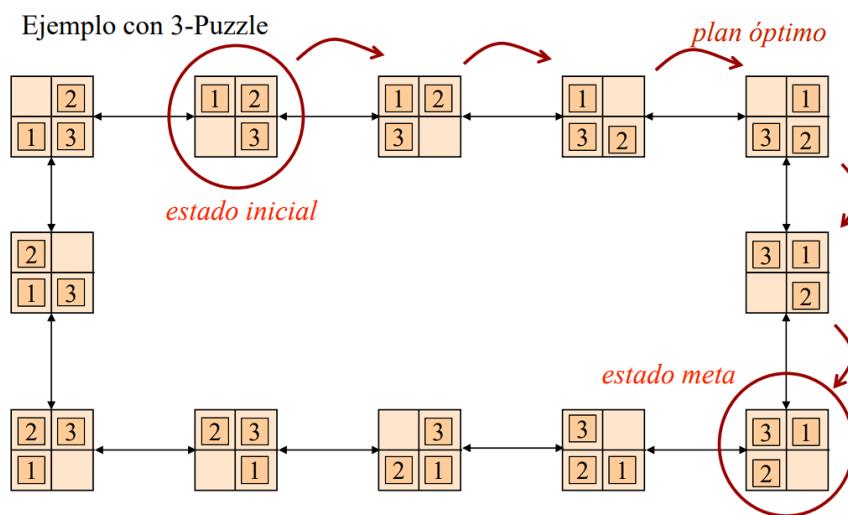
Representación de estados $\begin{bmatrix} x_{11}, x_{12} \\ x_{21}, x_{22} \end{bmatrix}$ $x_{i,j} \in \{0,1,2,3\}$, $x_{1,1} \neq x_{1,2} \neq x_{2,1} \neq x_{2,2}$

Estado inicial $\begin{bmatrix} 1,2 \\ 0,3 \end{bmatrix}$ Estado meta: $\begin{bmatrix} 3,1 \\ 2,0 \end{bmatrix}$

Operadores (o acciones)

$$\begin{array}{ll} \text{OP1: } \begin{bmatrix} 0, x \\ y, z \end{bmatrix} \rightarrow \begin{bmatrix} x, 0 \\ y, z \end{bmatrix} & \text{OP4: } \begin{bmatrix} y, x \\ 0, z \end{bmatrix} \rightarrow \begin{bmatrix} y, x \\ z, 0 \end{bmatrix} \\ \text{OP2: } \begin{bmatrix} 0, x \\ y, z \end{bmatrix} \rightarrow \begin{bmatrix} y, x \\ 0, z \end{bmatrix} & \text{OP5: } \begin{bmatrix} y, x \\ 0, z \end{bmatrix} \rightarrow \begin{bmatrix} 0, x \\ y, z \end{bmatrix} \\ \text{OP3: } \begin{bmatrix} x, 0 \\ y, z \end{bmatrix} \rightarrow \begin{bmatrix} 0, x \\ y, z \end{bmatrix} & \text{OP6: } \begin{bmatrix} x, z \\ y, 0 \end{bmatrix} \rightarrow \begin{bmatrix} x, z \\ 0, y \end{bmatrix} \\ \text{OP4: } \begin{bmatrix} x, 0 \\ y, z \end{bmatrix} \rightarrow \begin{bmatrix} x, z \\ y, 0 \end{bmatrix} & \text{OP7: } \begin{bmatrix} x, z \\ y, 0 \end{bmatrix} \rightarrow \begin{bmatrix} x, 0 \\ y, z \end{bmatrix} \end{array}$$

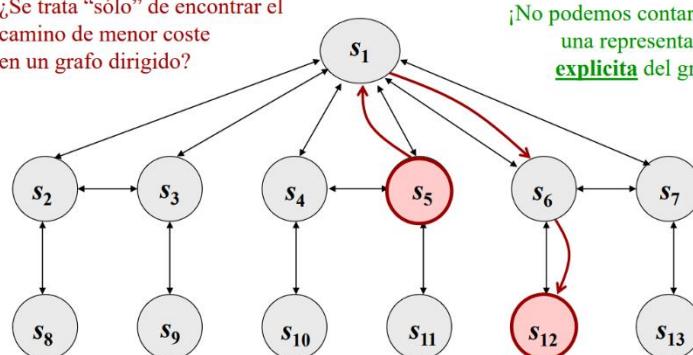
3-PUZZLE: ESPACIO DE ESTADOS



EL PROBLEMA DE BÚSQUEDA

¿Se trata "sólo" de encontrar el camino de menor coste en un grafo dirigido?

¡No podemos contar con una representación explicita del grafo!



REMODELANDO EL PROBLEMA DE BÚSQUEDA

Representación implícita del problema de búsqueda

Conocimientos mínimos a priori de un agente:

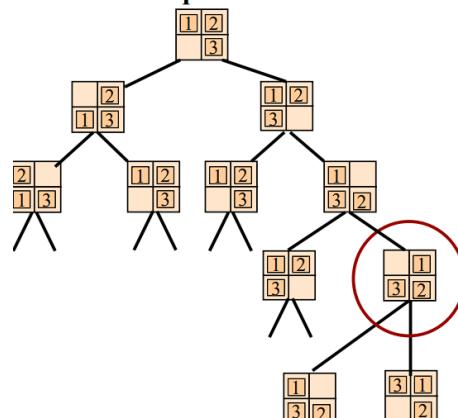
- s_0 Estado *inicial*
 - $\text{expandir}: s \mapsto \{s_{i_1}, \dots, s_{i_n}\}$ Conjunto finito de *sucesores* de un estado
 - $\text{meta?}: s \mapsto \text{verdad} \mid \text{falso}$ Prueba de *éxito* en un estado
 - $c: (s_i, s_j) \mapsto v, v \in \mathbb{N}$ *Coste* de un operador
- $$c(s_{i_1}, s_{i_2}, \dots, s_{i_n}) = \sum_{k=1}^{n-1} c(s_{i_k}, s_{i_{k+1}})$$
- Coste* de un plan

MÉTODO GENERAL DE BÚSQUEDA**Método de búsqueda:**

- estrategia para explorar el espacio de estados
- en cada paso se expande un estado
- se desarrolla sucesivamente un árbol de búsqueda

Método general de búsqueda:

1. seleccionar nodo hoja
2. comprobar si es nodo meta
3. expandir este nodo hoja

Árbol de búsqueda:

ALGORITMO GENERAL DE BÚSQUEDA

Elementos del algoritmo

- el árbol se representa en base a un registro del tipo nodo
- abierta es una lista de nodos, con las hojas actuales del árbol
- vacía? determina si una lista es vacía
- primero quita el primer elemento de una lista
- ordInsertar añade un nodo a una lista, **clasificado** según una función de orden

```

{búsqueda general}
abierta ←  $s_0$ 
Repetir
  Si vacia?(abierta) entonces
    devolver(negativo)
  nodo ← primero(abierta)
  Si meta?(nodo) entonces
    devolver(nodo)
  sucesores ← expandir(nodo)
  Para cada  $n \in$  sucesores hacer
    n.padre ← nodo
    ordInsertar(n, abierta, <orden>)
  Fin {repetir}
  
```

Muy importante

CLAVE DE LA IA

NUEVA DIAPO ACTUALIZADA POR EL PROFE

Búsqueda en el espacio de estados

```

1: procedure SPACESTATESEARCH(initialState)
2:   openList ← {initialState}
3:   while openList is not empty do
4:     currentState ← first(openList)
5:     if currentState is_goal? then
6:       return currentState
7:     end if
8:     successorStates ← expand currentState
9:     for all successor in successorStates do
10:      sucesor.parent ← {currentState}
11:      openList ← {sucesor}, <orden>
12:    end for
13:  end while
14:  return no solution found
15: end procedure

```

- **initialState**, **is_goal?**, **expand** representan el **conocimiento mínimo a priori** del agente
- **initialState**, **currentState**, **sucesor** son nodos del arbol
- **openList** es la **lista de nodos** con las hojas que conocemos del árbol de búsqueda
- **first(openList)** extrae el **primer nodo** de la lista
- **openList** ← {**sucesor**}, <**orden**> **inserta un nuevo nodo** en la lista abierta
- <**orden**> es el orden de inserción del **nuevo estado en la lista**

¡¡Aquí está la clave de la IA!!

ESTADOS REPETIDOS

Problema:

- el mismo estado puede repetirse varias veces en el árbol de búsqueda
- puede generarse el mismo subárbol varias veces

Soluciones:

- ignorarlo
- evitar ciclos simples (solucionado al expandir):
 - no añadir el padre de un nodo al conjunto de sucesores
- evitar ciclos generales (solucionado al expandir):
 - no añadir un antecesor de un nodo al conjunto de sucesores
- evitar todos los estados repetidos (lista cerrada):
 - no añadir ningún nodo existente en el árbol al conjunto de sucesores

CLASIFICACIÓN DE MÉTODOS

Características:

- Completitud: se encuentra una solución si existe
- Optimalidad: se encuentra la mejor solución si hay varias
- Complejidad en tiempo: ¿cuánto se tarda en encontrar la solución?
- Complejidad en espacio: ¿cuánta memoria se utiliza en la búsqueda?

Tipos de métodos de búsqueda:

- No informados (Sesión 2)
 - utilizan sólo los conocimientos mínimos (búsqueda en amplitud, búsqueda de coste uniforme, ...)
- Heurísticos (Sesiones 3-6)
 - además utilizan información aproximada, y específica del problema, para guiar la búsqueda (Algoritmo A* y extensiones, búsqueda multiagente)

2.2 BÚSQUEDA NO-INFORMADA

BÚSQUEDA EN AMPLITUD

Búsqueda en amplitud:

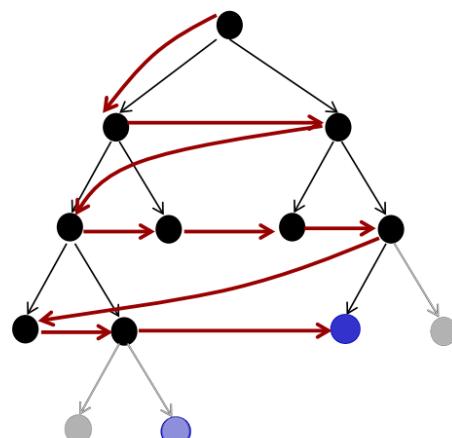
- Inglés: breadth first search

Estrategia:

- generar el árbol por niveles de profundidad
- expandir todos los nodos de nivel i , antes de expandir nodos de nivel $i+1$

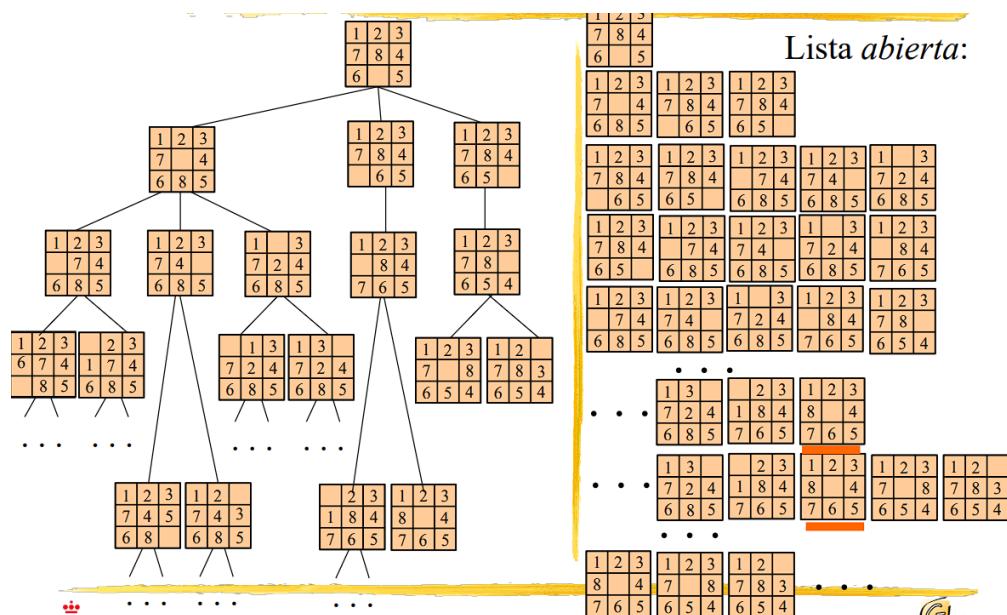
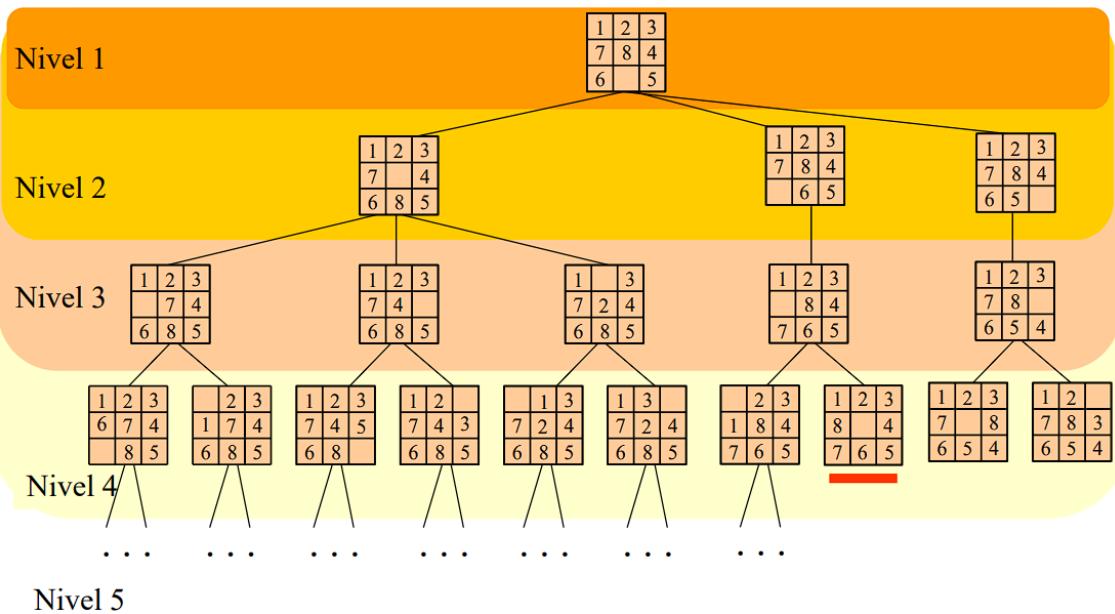
Resultado:

- considera primero todos los caminos de longitud 1, después los caminos de longitud 2, etc.
- Se encuentra el estado meta de menor profundidad



ÁRBOL DE BÚSQUEDA EN AMPLITUD

Búsqueda en amplitud (evitando ciclos simples):



ALGORITMO PARA BÚSQUEDA EN AMPLITUD

Algoritmo:

- usar el algoritmo general de búsqueda
- añadir nuevos sucesores al final de la lista abierta
- abierta funciona como cola
 - inserción al final
 - recuperación desde la cabeza
- **Estructura FIFO:**
 - siempre expandir primero el nodo más antiguo
(es decir: menos profundo)

{búsqueda en amplitud}

abierta $\leftarrow s_0$

Repetir

Si vacía?(abierta) entonces

devolver(negativo)

nodo \leftarrow primero(abierta)

Si meta?(nodo) entonces

devolver(nodo)

sucesores \leftarrow expandir(nodo)

Para cada $n \in$ sucesores hacer

n.padre \leftarrow nodo

ordInsertar(n,abierta,final)

Fin {repetir}

Algoritmo:

- **Breath First Search (BFS)**
- usar el **algoritmo general de búsqueda**
- añadir nuevos sucesores al **final de la lista abierta**
- la lista abierta funciona como una **cola/estructura FIFO**
 - ✓ inserción al final
 - ✓ recuperación desde la cabeza
 - ✓ siempre se expande primero el nodo más antiguo en la lista
(es decir: el menos profundo)

Búsqueda en amplitud

```

1: procedure SPACESTATESEARCH(initialState)
2:   openList  $\leftarrow \{initialState\}$ 
3:   while openList is not empty do
4:     currentState  $\leftarrow$  first(openList)
5:     if currentState is goal? then
6:       return currentState
7:     end if
8:     successorStates  $\leftarrow$  expand currentState
9:     for all successor in successorStates do
10:      succesor.parent  $\leftarrow \{currentState\}$ 
11:      openList  $\leftarrow \{succesor\}, FINAL_LISTA
12:    end for
13:  end while
14:  return no solution found
15: end procedure$ 
```

COMPLEJIDAD

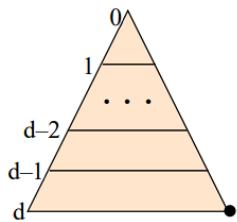
Complejidad en tiempo y espacio:

- proporcional al número de nodos expandidos

Suponemos que en el árbol de búsqueda

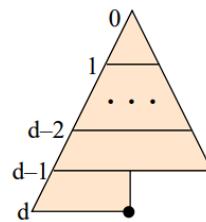
- el factor de ramificación es b
- el mejor nodo meta tiene profundidad d

Peor caso



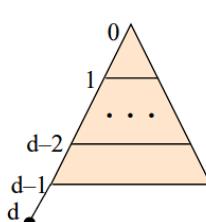
$$1+b+\dots+b^{d-1}+b^d \in O(b^d)$$

Caso medio



$$1+b+\dots+b^{d-1}+b^d/2 \in O(b^d)$$

Mejor caso



$$1+b+\dots+b^{d-1}+1 \in O(b^d)$$

REQUERIMIENTO DE TIEMPOS Y MEMORIA

Ejemplo: recursos requeridos por la búsqueda en amplitud en el peor caso

- factor de ramificación efectivo: 10
- tiempo: 1.000.000 nodos/segundo
- memoria: 1.000 bytes/nodo

<i>d</i>	nodos	tiempo	memoria
2	110	0,11 ms	107 KB
4	11.110	11 ms	10,6 MB
6	10^6	1,1 s	1 GB
8	10^8	2 min	103 GB
10	10^{10}	3 horas	10 TB
12	10^{12}	13 días	1.000 TB
14	10^{14}	3,5 años	99 PB
16	10^{16}	350 años	10.000 PB

VENTAJAS Y DESVENTAJAS DE LA BÚSQUEDA EN AMPLITUD

Ventajas

- completo:
- Siempre se encuentra un nodo meta si existe
- Óptimo (para operadores de coste uno):
- Siempre se encuentra en el nodo meta menos profundo.

Problemas

- Complejidad
- Exponencial incluso en el mejor caso.
- Los problemas de espacio son aún más graves que los problemas de tiempo.

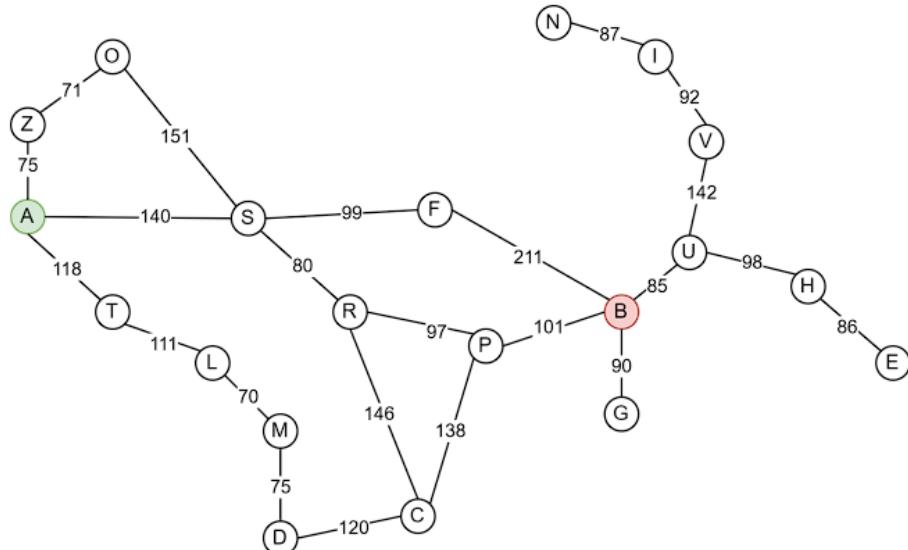
PROBLEMAS DE ENCONTRAR RUTAS

Estado: estancia en una ciudad

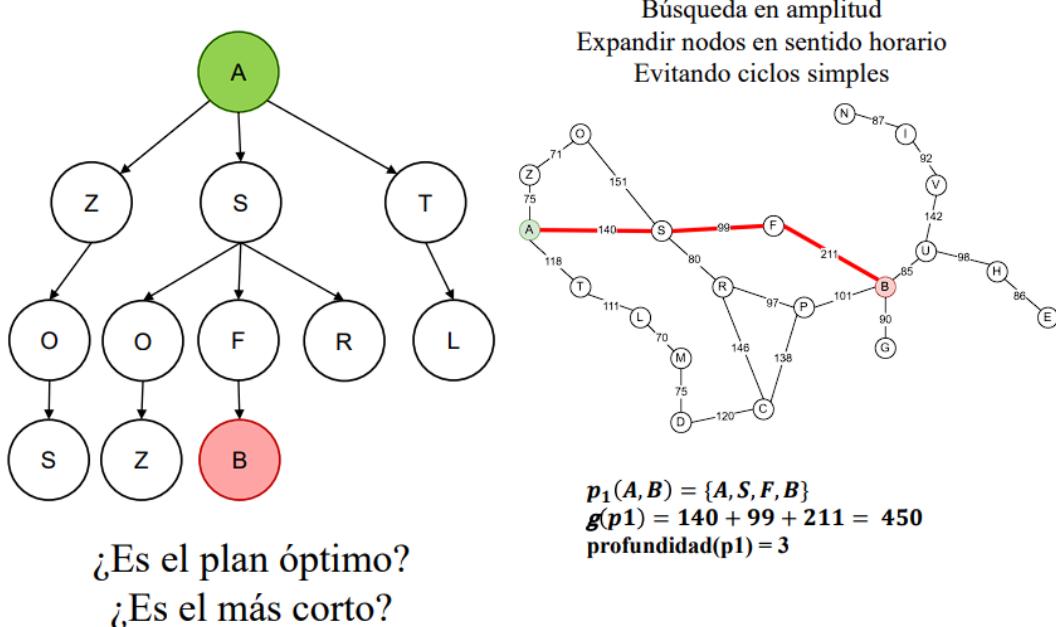
Coste de un operador: distancia por carretera a la ciudad vecina

Operadores: ir a una ciudad vecina

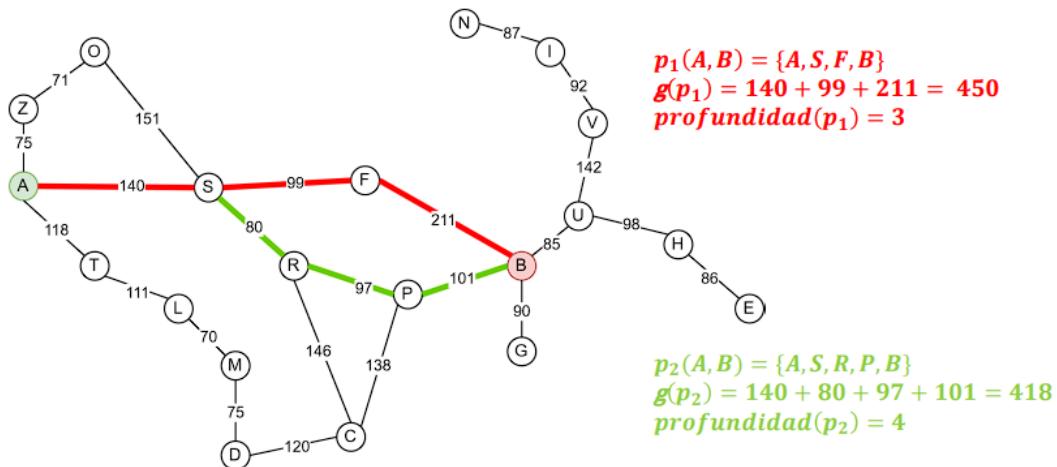
Coste de un plan: suma de distancias entre las ciudades visitadas



EJEMPLO



PROBLEMA DE ENCONTRAR RUTAS



Problema:

- La búsqueda en amplitud encuentra el nodo meta de menor profundidad; éste puede *no* ser el nodo meta de coste mínimo.

$$\begin{aligned} \text{profundidad}(p_1) &= 3 < 4 = \text{profundidad}(p_2) \\ g(p_1) &= 450 > 418 = g(p_2) \end{aligned}$$

BÚSQUEDA DE COSTE UNIFORME

Búsqueda de coste uniforme:

- inglés: uniform cost search
- Idea: – guiar la búsqueda por el coste de los operadores
- Método:
 - $g(n)$: coste mínimo para llegar del nodo inicial al nodo n
 - expandir siempre el nodo de menor coste g primero
- Algoritmo:
 - almacenar cada nodo con su valor g
 - insertar los nuevos nodos en abierta en orden ascendente según su valor g

{búsqueda de coste uniforme}

abierta $\leftarrow s_0$

Repetir

Si vacío?(abierta) **entonces**

devolver(negativo)

 nodo \leftarrow primero(abierta)

Si meta?(nodo) **entonces**

devolver(nodo)

 sucesores \leftarrow expandir(nodo)

Para cada $n \in$ sucesores **hacer**

 n.padre \leftarrow nodo

 ordInsertar(n ,abierta, g)

Fin {repetir}

Algoritmo:

- **Uniform Cost Search (UCS)**
- usar el **algoritmo general de búsqueda**
- los nodos ahora contienen el coste de los operadores para llegar al nodo desde el nodo inicial

$$g(n_i) = g(\{n_0 - n_i\})$$
- añadir nuevos sucesores al **según el coste g del nodo**

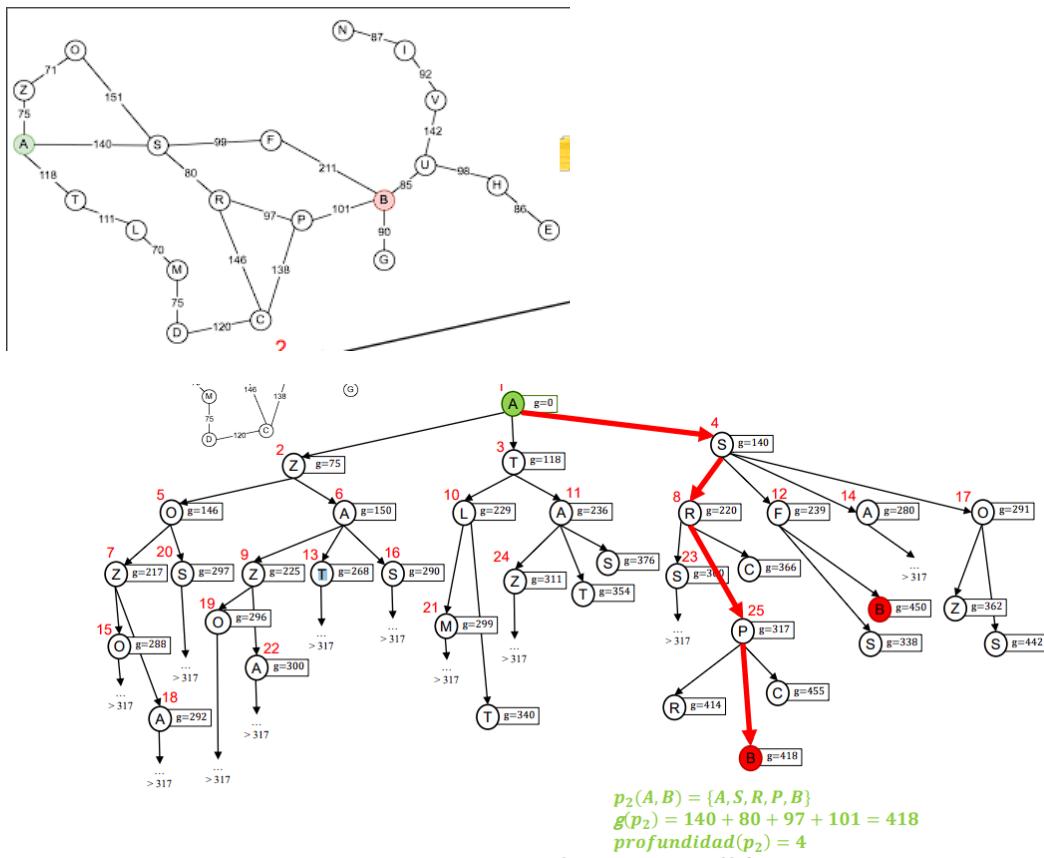
Búsqueda de coste uniforme

```

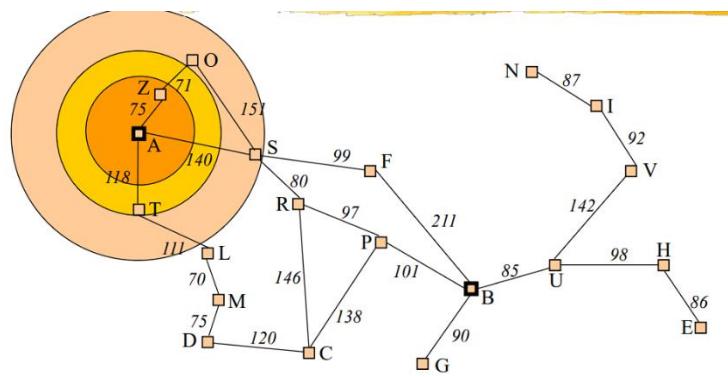
1: procedure SPACESTATESEARCH(initialState)
2:   openList  $\leftarrow \{\text{initialState}\}$ 
3:   while openList is not empty do
4:     currentState  $\leftarrow$  first(openList)
5:     if currentState is_goal? then
6:       return currentState
7:     end if
8:     successorStates  $\leftarrow$  expand currentState
9:     for all successor in successorStates do
10:      successor.parent  $\leftarrow \{\text{currentState}\}$ 
11:      openList  $\leftarrow \{\text{successor}, \text{successor.g}\}$ 
12:    end for
13:  end while
14:  return no solution found
15: end procedure

```

PROBLEMA DE ENCONTRAR RUTAS



LÓGICA DE LA BÚSQUEDA DE COSTE UNIFORME



CARACTERÍSTICAS DE LA BÚSQUEDA DE COSTE UNIFORME

Análisis:

- La búsqueda de coste uniforme enumera sucesivamente todos los nodos del espacio de estados por costes (valores de g) crecientes.
 - la sucesión de valores de g crece de forma monótona en todos los caminos del árbol de búsqueda, ya que el coste de cualquiera de los operadores es positivo.
- La búsqueda de coste uniforme es completa:
 - al ser los costes números enteros positivos, la sucesión de valores de g no es acotada.
 - por tanto, si un nodo meta existe en el espacio de estados, será expandido alguna vez.
- La búsqueda de coste uniforme es óptima:
 - al expandir un nodo cualquiera, todos los nodos de menor valor de g han sido expandidos antes.
 - el primer nodo meta expandido es el nodo meta de menor valor de g .
 - el nodo meta de menor valor de g es el nodo meta de menor coste.

HOJA DE PROBLEMAS I

1. El enfoque de los Agentes Inteligentes concibe el objetivo de la Inteligencia Artificial como el intento de construir sistemas ...

- a) ... que actúen como los seres humanos.
- b) ... que actúen de forma racional.**
- c) ... que piensen como los seres humanos.
- d) ninguna de las anteriores.

Porque resuelve los problemas de forma racional

2. ¿Para cual(es) de las siguientes tareas pueden construirse agentes basados en algoritmos de búsqueda en el espacio de estados?

- a) Encontrar una solución al problema del n-puzzle.**
- b) Gestionar el tráfico rodado en una red de autopistas urbanas.
- c) Conducir un taxi en las calles de Barcelona. d) Jugar a las 4 en ralla contra un jugador humano.

**Resolvemos problemas de un jugador. Para las torres de Hanoi
también, aunque no es la forma más eficiente**

3. ¿Cuáles de las siguientes afirmaciones acerca de los algoritmos de búsqueda no informados es/son cierta(s)?

- a) Los algoritmos de búsqueda no informados requieren de información heurística para que sean óptimos.
- b) La búsqueda en amplitud es óptima y completa siempre y cuando el coste de los operadores sea constante. (es cierta ya que la búsqueda en amplitud no tiene en cuenta los costes)
- c) La búsqueda en profundidad es óptima y completa siempre que el coste de los operadores sea constante. (Falsa porque la búsqueda en profundidad no es completa ni óptima)
- d) Tanto la complejidad en tiempo como la complejidad en espacio de la búsqueda en amplitud se pueden expresar en función del número de nodos expandidos.

4. Contemple el problema de búsqueda de la figura 1. ¿Cuál es el factor de ramificación del árbol de búsqueda generado por los métodos de búsqueda en el espacio de estados, si no se filtran estados repetidos?

- a) 0.5
- b) 2
- c) 4
- d) 6

Cada nodo expande a otros cuatro

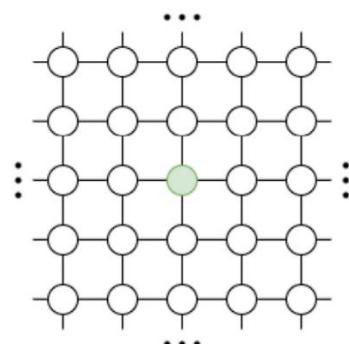


Figura 1

5. Contemple el problema de búsqueda de la figura 1. En el árbol generado por la búsqueda en amplitud, si no se filtran estados repetidos, ¿cuántos nodos hay a nivel de profundidad k (suponiendo que la raíz tiene nivel 0, es decir $k \geq 0$)

- a) $k/4$
- b) 4^k Factor de ramificación elevado a
- c) $4 * k^2$
- d) k^4

Ejercicio 2: modelado del problema

En una mesa se encuentran dos jarras, una con una capacidad de 3 litros (llamada Tres), y la otra con una capacidad de 4 litros (llamada Cuatro). Inicialmente, Tres y Cuatro están vacías. Cualquiera de ellas puede llenarse con el agua de un grifo G. Asimismo, el contenido tanto de Tres como de Cuatro puede vaciarse en una pila P. Es posible echar toda el agua de una jarra a la otra. No se dispone de dispositivos de medición adicionales. Se trata de encontrar una secuencia de operadores que deje exactamente dos litros de agua en Cuatro.

1. Modela este problema como un problema de búsqueda en el espacio de estados. Con tal fin, define el estado inicial, el conjunto de estados meta, los operadores (especificando sus precondiciones y/o postcondiciones), así como el coste de cada operador.
2. Caracteriza el conocimiento a priori del agente de resolución del problema correspondiente. Facilita ejemplos de los resultados de la función expandir.
3. Encuentra mediante búsqueda no informada una solución al problema.

Ejercicio 3: búsqueda en amplitud

Figura 2 El grafo que se muestra en la figura 2 determina un problema de búsqueda. Cada nodo representa un estado. Los arcos modelan la aplicación de operadores. El estado A es el estado inicial, los estados E y K son los estados meta.

1. Desarrolla el árbol de búsqueda que genera la búsqueda en amplitud, indicando el orden en que se comprueban los nodos. ¿Cuál de los nodos meta se encuentra en primer lugar?
2. Indica el orden en que se expanden los nodos.
3. Indica el estado de la lista abierta en cada paso del algoritmo.

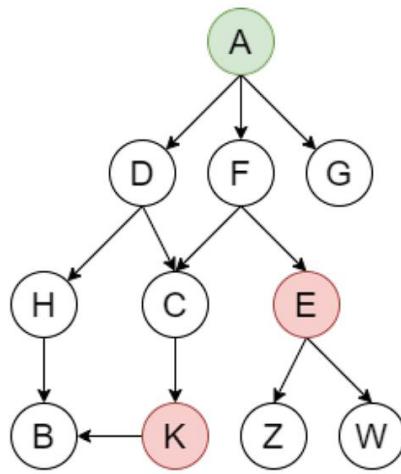


Figura 2

Ejercicio 4: búsqueda en profundidad.

Dado el algoritmo de búsqueda en el espacio de estados genérico que se presenta a la derecha:

1. ¿Qué habría que hacer para implementar una búsqueda en profundidad?
2. Para el grafo de la figura 2, desarrolla el árbol de búsqueda que genera la búsqueda en profundidad, indicando el orden en que se comprueban los nodos. ¿Cuál de los nodos meta se encuentra en primer lugar?
3. Haz un análisis de complejidad de dicho algoritmo, similar al de la diapositiva 40, asumiendo un límite de profundidad d fijado a priori.

Algorithm 1 Space State Search

```

1: procedure SPACESTATESEARCH(initialState)
2:   openList  $\leftarrow \{\text{initialState}\}
3:   while openList is not empty do
4:     currentState  $\leftarrow \text{first}(\text{openList})
5:     if currentState is _goal? then
6:       return currentState
7:     end if
8:     successorStates  $\leftarrow \text{expand } \text{currentState}
9:     for all successor in successorStates do
10:      successor.parent  $\leftarrow \{\text{currentState}\}
11:      openList  $\leftarrow \{\text{successor}\}, < \text{order} >
12:    end for
13:  end while
14:  return no solution found
15: end procedure$$$$$ 
```

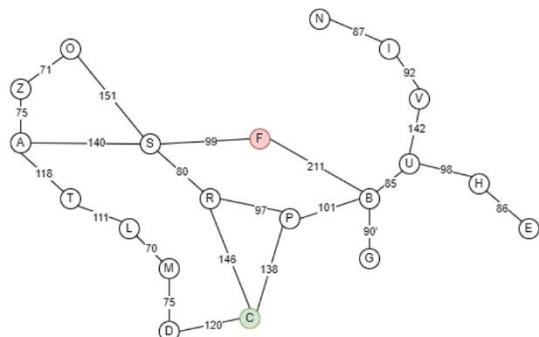
Ejercicio 5: búsqueda en el espacio de estados

Considera el siguiente problema:

Un hombre se encuentra en la orilla izquierda de un río junto con un lobo, una oveja y una col. Quiere cruzar el río llevando consigo el lobo, la oveja y la col. En la barca sólo hay dos plazas, una de las cuales debe ir ocupada por el hombre. Cada uno de los restantes pasajeros (lobo, oveja, col) ocupa una plaza, de tal modo que sólo uno puede acompañar al hombre en cada viaje. Además, no puede dejar solos en una orilla al lobo con la oveja, ni a la oveja con la col, ni los tres, porque el lobo se comería a la oveja y/o la oveja se comería la col. Supón que se modela el problema como un espacio de estados, donde cada estado se describe como un par de conjuntos, indicando quien(es) se encuentran en cada orilla del río ($c = \text{col}$; $o = \text{oveja}$; $l = \text{lobo}$; $h = \text{hombre}$). Por tanto, el estado inicial del problema sería $(\{c, o, l, h\}, \{\})$, y el estado meta $(\{\}, \{c, o, l, h\})$.

1. Simula la estrategia de búsqueda en amplitud para este problema, asumiendo que se filtran todos los estados repetidos. Expande los sucesores de los nodos siguiendo las preferencias del hombre, las cuales se ordenan de menor a mayor como sigue: viajar con la oveja, viajar con la col, viajar solo, viajar con el lobo. Dibuja el árbol de búsqueda correspondiente e indica el orden en que se exploran los nodos.
2. Simula ahora la estrategia de búsqueda en profundidad para este problema. Dibuja de nuevo el árbol de búsqueda correspondiente e indica el orden en que se exploran los nodos, asumiendo que se filtran todos los estados repetidos y siguiendo las preferencias del apartado anterior.
3. Supón ahora que no se filtra ningún estado repetido. ¿La búsqueda en amplitud y la búsqueda en profundidad seguirían siendo completas? Razona brevemente tu respuesta.

Ejercicio 6: búsqueda de coste uniforme



Supón que en la red de carreteras de Rumanía presentada en clase nuestra agente se encuentra en Caiva (C) y desea trasladarse a Fagaras (F).

1. Desarrolla el árbol de búsqueda que genera la búsqueda de coste uniforme, indicando los valores de g para cada nodo.
2. Indica el orden en que se expanden los nodos.
3. Indica el estado de la lista abierta en cada paso del algoritmo.

Tema 3. – Búsqueda con heurísticas débiles

HEURÍSTICAS

Heurística (griego: *heuriskein*): “encontrar”, “descubrir”

Inteligencia Artificial:

- compila conocimiento “empírico” sobre un problema / un entorno para resolver problemas o tomar decisiones.

Heurísticas “débiles”: para guiar la búsqueda

- información heurística puede mejorar el rendimiento medio de un método de resolución de problemas, pero no garantiza una mejora en el peor caso
- método riguroso + información heurística
- búsqueda: mejora de complejidad no garantizado

Heurísticas “fuertes”: para limitar la búsqueda (el espacio de soluciones)

- una heurística suele facilitar la resolución de un problema, pero no garantiza que se resuelva
- búsqueda: optimalidad o incluso completitud no garantizados

FUNCIONES HEURÍSTICAS DÉBILES

Funciones heurísticas para búsqueda en el espacio de estados:

- estiman la adecuación de un nodo para ser expandido
- $h(nn)$: mide el coste real desde el nodo nn hasta el nodo meta más cercano
- $h^*(nn)$: es una función heurística que estima el valor de $h(nn)$
- una función heurística h^* es admisible, si no sobreestima el valor de

$$h(n) : h^*(n) \leq h(n) \quad \forall n \in S$$

Métodos de búsqueda “el mejor primero” eligen el nodo más prometedor para expandir:

- Búsqueda voraz: $f^*(n) = h^*(n)$
- Búsqueda A*: $f^*(n) = g(n) + h^*(n)$

Ejemplos de funciones heurísticas optimistas:

- Encontrar rutas: distancia en línea recta hasta un nodo meta
- Torres de Hanoi: número de discos descolocados
- Salir de un laberinto: distancia Manhattan al node meta

BÚSQUEDA A*

Idea: minimizar el coste estimado total de un camino en el árbol de búsqueda, combinando:

- el coste para llegar al nodo n , que se conoce exactamente: $g(n)$
- el coste aproximado para llegar a un nodo meta desde el nodo n , estimado por la función heurística $h(n)$

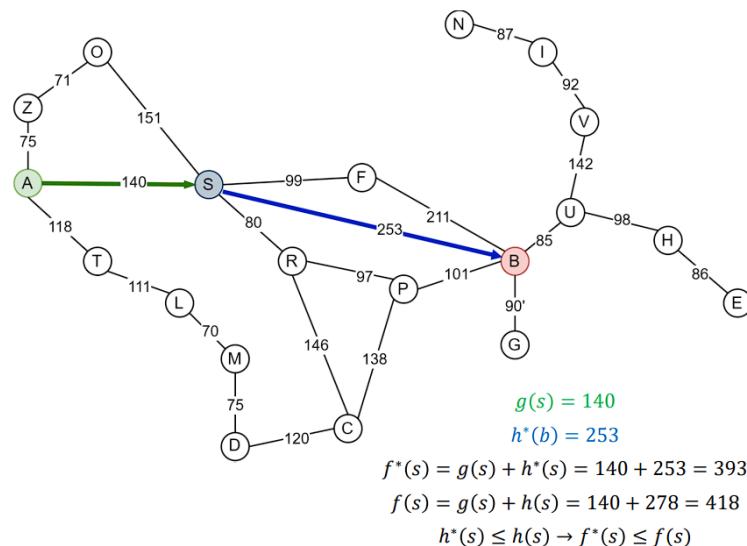
Función heurística de A*:

- $f(n) = g(n) + h(n)$: coste real del plan de mínimo coste que pasa por n
- $f^*(n) = g(n) + h^*(n)$: estimación de f
- Siempre se cumple que:
 - $h^*(n) \leq h(n) \rightarrow f^*(n) \leq f(n)$
 - $f^*(n_m) = g(n_m) + h^*(n_m) = g(n_m) + 0 = f(n_m)$

Estrategia A*:

- entre las hojas del árbol de búsqueda, elegir el nodo de valor f^* mínimo

FUNCIÓN HEURÍSTICA PARA ENCONTRAR RUTAS



ALGORITMO A*

Algoritmo:

- **A***
- Usar el **algoritmo general de búsqueda**
- los nodos ahora contienen el valor de la función

$$f^*(n) = g(n) + h^*(n)$$
- añadir nuevos sucesores a la *openList* **según el coste f del nodo** de menor a mayor

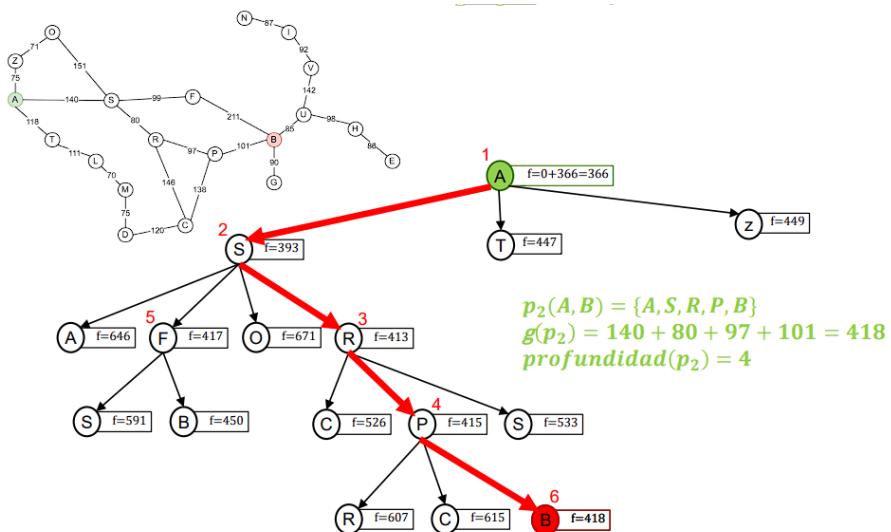
Búsqueda A*

```

1: procedure SPACESTATESEARCH(initialState)
2:   openList  $\leftarrow \{\text{initialState}\}$ 
3:   while openList is not empty do
4:     currentState  $\leftarrow \text{first}(\text{openList})$ 
5:     if currentState is _goal? then
6:       return currentState
7:     end if
8:     successorStates  $\leftarrow \text{expand } \text{currentState}$ 
9:     for all successor in successorStates do
10:      successor.parent  $\leftarrow \{\text{currentState}\}$ 
11:      openList  $\leftarrow \{\text{successor}, \text{successor.f}\}$ 
12:    end for
13:  end while
14:  return no solution found
15: end procedure

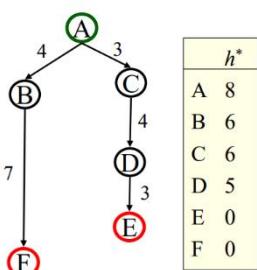
```

PROBLEMAS DE ENCONTRAR RUTAS



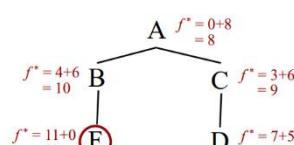
VALORES DE F*

Espacio de estados:

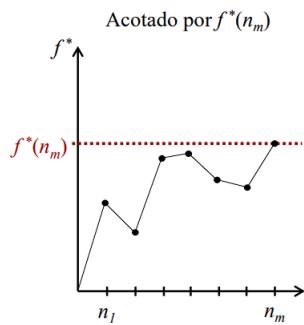


- Estado inicial: A
- Estados meta: E, F
- $g(E)=10 < 11=g(F)$

Árbol de búsqueda:



- A^* no encuentra el mejor nodo meta E, sino F
- h^* no es optimista: $h^*(D) = 5 > 3 = h(D)$
- En el camino de A al mejor nodo meta E ($f^*(E) = 10$) hay un “pico” D ($f^*(D) = 12$) que la búsqueda A^* no ha podido superar

VALORES DE f^* EN ÁRBOLES DE BÚSQUEDA A*


**Si h^* es admisible
(nodos meta)**

EVALUACIÓN DE A*

**Para ampliar, ver
Definiciones**

A* es Completo

- Siempre que no existan infinitos nodos n tal que $f^*(n) < f(n)$

A* es Óptimo

- Siempre si h^* es admisible

Complejidad en el tiempo

- Altamente dependiente de la calidad de h^* :

Si $h^*(n) = 0$, degenera en la búsqueda por coste uniforme, $O(b^C)$.

Si $h^*(n) = h(n)$, se ejecuta en tiempo lineal $O(C)$.

Normalmente conseguir $h^*(n) = h(n)$ equivale a resolver el problema completo.

Normalmente consigue una mejor notable con respecto a métodos no informados.

Complejidad en memoria

- Exponencial $O(b^C)$, este es su mayor problema. La mayor parte de las optimizaciones van dirigidas a resolver este.

DISEÑO DE FUNCIONES HEURÍSTICAS

¿Cómo diseñar funciones heurísticas admisibles para un problema?

Relajación de las restricciones del problema

Mantenemos el mismo conjunto de estados, estado inicial, estado meta, pero definimos **menos precondiciones/restricciones** para cada operador.

De esta forma **todas las soluciones al problema original también lo son al problema relajado**, y posiblemente algunos más, ya que la complejidad del problema ahora es menor.

Idea:

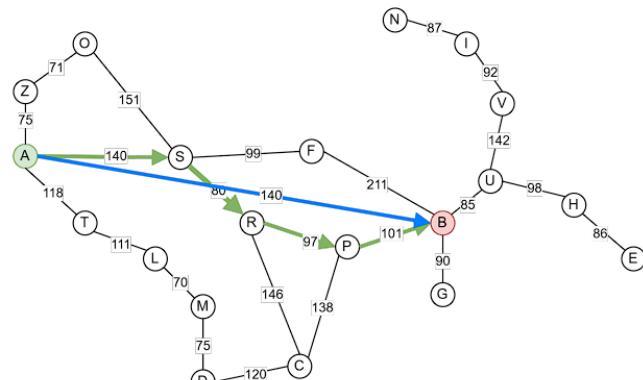
- Usar el coste exacto $h(n)$ de llegar desde el estado n a un nodo meta en el problema relajado como valor heurístico $h^*(n)$ en el problema original.
- Por construcción, una función heurística h^* así construida es admisible.

RELAJACIÓN DE LAS RESTRICCIONES DEL PROBLEMA

El problema de la navegación

Problema original p_1 :

- Para navegar de una ciudad a otra, es necesario utilizar las carreteras definidas.



Problema relajado p_2 :

- Se puede viajar en línea recta de una ciudad a otra.

$$h_{p1}^*(n) = h_{p2}(n)$$

EL PROBLEMA DEL 8-PUZZLE

Problema original p_1 :

- Una pieza puede moverse de A a B si son adyacentes y B está vacía.

2	7	3
1	8	4
6		5

estado inicial

Problemas relajados:

- p_2 : Una pieza puede moverse de A a B siempre

$$h_{p1}^*(n) = h_{p2}(n) = \text{número de piezas descolocadas}$$

$$h_{p1}^*(s_0) = 5$$
- p_3 : Una pieza puede moverse de A a B si B está vacía

$$h_{p1}^*(n) = h_{p3}(n) = \text{número de saltos necesarios}$$

$$h_{p1}^*(s_0) = 5$$
- p_4 : Una pieza puede moverse de A a B si A es adyacente a B

$$h_{p1}^*(n) = h_{p4}(n) = \text{suma de las distancias Manhattan}$$

$$h_{p1}^*(s_0) = 1 + 3 + 1 + 1 + 1 = 7$$

1	2	3
8		4
7	6	5

estado meta

CALIDAD DE LAS HEURÍSTICAS

**Para ampliar, ver
Definiciones**

Importancia del valor de h^* :

- en el 8-puzzle anterior h_{p4}^* es más informada que h_{p2}^* y
 - las piezas bien colocadas no cuentan en h_{p4}^* ni en h_{p2}^*
 - la distancia Manhattan de cada pieza descolocada es al menos
 - en consecuencia, en toda posible configuración nn del 8-puzzle la suma de las distancias es igual o mayor que la suma de piezas descolocadas
 - Por tanto, para todas las distintas disposiciones del puzzle,

$$h_{p4}^*(n) \geq h_{p2}^*(n) \quad \forall n \in S$$

Definición:

- Sean h_1^* y h_2^* dos funciones heurísticas admisibles para un mismo problema.

Se dice que h_1^* es más informada que h_2^* , si

$$h_1^*(n) \geq h_2^*(n) \quad \forall n \in S$$

- Si h_1^* es más informada que h_2^* entonces h_2^* expandirá siempre al menos tantos nodos como h_1^* .

- Si disponemos de varias funciones heurísticas, debemos escoger siempre la de mayor valor.

$$h^*(n) = \max_m [h_1^*(n), h_2^*(n), \dots, h_m^*(n)]$$

DEFINICIONES

OPTIMALIDAD DE A*

Lema 1: Sea $p(n_m)$ el conjunto de nodos en el camino desde la raíz a un nodo meta n_m cualquiera. Si h^* es admisible ($h^* \leq h$), entonces se cumple que

$$f^*(n_k) \leq f(n_m) \quad \forall n_k \in p(n_m)$$

Prueba:

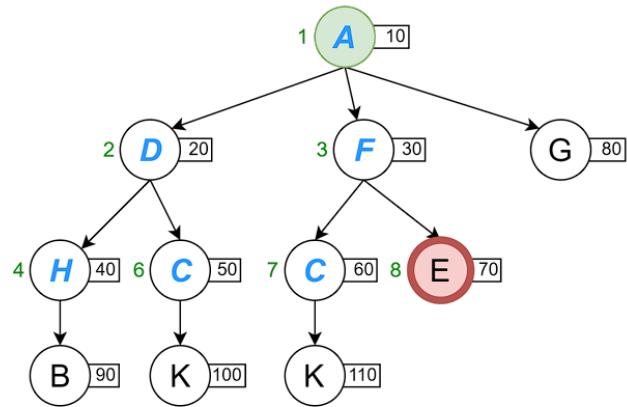
$$\begin{aligned} h^*(n_k) \leq h(n_k) \rightarrow g(n_k) + h^*(n_k) \leq g(n_k) + h(n_k) & \xleftarrow{\text{Porque } h^* \leq h} \\ g(n_k) + h(n_k) = g(n_m) \rightarrow g(n_k) + h^*(n_k) \leq g(n_m) & \xleftarrow{\text{Porque el coste real}} \\ f^*(n_k) = g(n_k) + h^*(n_k) \leq g(n_m) + 0 = f(n_m) & \xleftarrow{\text{el coste real hasta el nodo } n_k \text{ sumado al coste real desde el nodo } n_k \text{ al nodo meta } n_m \text{ es igual al coste total hasta el nodo } n_m} \\ \text{Porque el valor } f^* \text{ del nodo } n_m \text{ es el coste total hasta el propio nodo} & \xrightarrow{\text{f}^*(n_k) \leq f(n_m)} \end{aligned}$$

Corolario 1: Sea n_m el mejor nodo meta. Sea $p(n_i)$ el conjunto de nodos desde el nodo inicial n_0 a un nodo n_i cualquiera en el árbol de búsqueda, incluido n_i . Si h^* es accesible, entonces se cumple que el algoritmo $A^*(h^*)$ expande todos los nodos

$$n_i: \forall n_j \in p(n_i) \Rightarrow f(n_j) \leq f(n_m)$$

Se expanden:

$$\begin{aligned} p(A): A \rightarrow f^*(A) = 10 \leq 70 = f^*(E) \\ p(D): \{A \rightarrow f^*(A) = 10 \leq 70 = f^*(E), \\ D \rightarrow f^*(D) = 20 \leq 70 = f^*(E)\} \\ p(F): \{A \rightarrow f^*(A) = 10 \leq 70 = f^*(E), \\ D \rightarrow f^*(F) = 30 \leq 70 = f^*(E)\} \\ p(H): \{A \rightarrow f^*(A) = 10 \leq 70 = f^*(E), \\ D \rightarrow f^*(D) = 30 \leq 70 = f^*(E), \\ H \rightarrow f^*(H) = 40 \leq 70 = f^*(E)\} \\ p(C): \{A \rightarrow f^*(A) = 10 \leq 70 = f^*(E), \\ D \rightarrow f^*(D) = 20 \leq 70 = f^*(E), \\ C \rightarrow f^*(C) = 50 \leq 70 = f^*(E)\} \\ p(C): \{A \rightarrow f^*(A) = 10 \leq 70 = f^*(E), \\ D \rightarrow f^*(F) = 30 \leq 70 = f^*(E), \\ C \rightarrow f^*(C) = 50 \leq 70 = f^*(E)\} \end{aligned}$$



Teorema 1: Si h^* es admisible, entonces el método A^* es óptimo

$$h^*(n_k) \leq h(n_k)$$

Lema 1: $f^*(n_k) \leq f(n_{m1}) \quad \forall n_k \in p(n_{m1}) \rightarrow f^*(n_{m1}) \leq g(n_{m1})$

Prueba:

Sean dos nodos meta n_{m1}, n_{m2} tal que el coste de n_{m2} es mayor que el de n_{m1} : $g(n_{m2}) > g(n_{m1})$ ⁽¹⁾

Supongamos que el algoritmo A^* devuelve el nodo n_{m2} . Como A^* devuelve los nodos ordenados por el valor de f^* de menor a mayor, entonces debería existir un nodo $n_k \in p(n_{m1}) \quad \forall f(n_{m2}) \leq f^*(n_k)$ ⁽²⁾

Pero sabemos por el Lema 1 que $f^*(n_k) \leq f(n_{m1})$ y por (1) que $f(n_{m1}) < f(n_{m2})$ así que podemos afirmar que $f^*(n_k) \leq f(n_{m1}) < f(n_{m2})$ ⁽³⁾ y que siempre se expandirá n_k antes de $f(n_{m2})$

Vemos que (3) contradice (2), A^* siempre explorará n_k antes de $f(n_{m2})$ y por tanto siempre se alcanzará n_{m1} en primer lugar. Por tanto,

A^* es óptimo

Teorema 2: El método A^* es completo si existe solución

$$h^*(n_k) \leq h(n_k)$$

$$\text{Lema 1: } f^*(n_k) \leq f(n_{m1}) \forall n_k \in p(n_{m1}) \rightarrow f^*(n_{m1}) \leq g(n_{m1})$$

Prueba:

Sea un nodo meta n_{m1} y $p(n_{m1})$ el conjunto de nodos en el camino de la raíz al nodo.

Supongamos que el algoritmo A^* no encuentra el nodo meta n_{m1} (el algoritmo no es completo). Para que el nodo meta n_{m1} no sea devuelto, debería existir un camino con nodos infinitos p_{inf} del que se expanden todos los nodos $n_i \in p_{inf} \forall f(n_i) < f(n_{m1})$ ⁽¹⁾

Como el coste de los operadores crece siempre, y el camino p_{inf} es infinito, sabemos que existe algún nodo $n_i \in p_{inf} \forall f(n_i) = g(n_i) + h^*(n_i) > \max[p(n_{m1})]$ ⁽²⁾ así que se expandirán los nodos de $p(n_{m1})$ antes que n_i .

Así que (2) contradice (1) y, por tanto:

A^* es completo

CALIDAD DE LAS FUNCIONES HEURÍSTICAS

Teorema 3: Sean h_1^* y h_2^* dos funciones heurísticas admisibles para un mismo problema. Si h_1^* es más informada que h_2^* , entonces $A^*(h_2^*)$ expande **al menos tantos nodos** como $A^*(h_1^*)$.

Prueba:

Para un nodo meta n_m se cumple que $f_1^*(n_m) = f_2^*(n_m) = \dots = f_n^*(n_m)$ ⁽¹⁾

Como h_1^* es más informada que h_2^* se cumple que $h_1^*(n) \geq h_2^*(n)$ y por tanto $f_1^*(n) \geq f_2^*(n)$ ⁽²⁾

Por (1) y por el **Corolario 1** se continua que⁽³⁾:

- $A^*(h_1^*)$ expande todos los nodos n_j : $\forall n_j \in p(n_i) \Rightarrow f_1^*(n_j) \leq f(n_m)$
- $A^*(h_2^*)$ expande todos los nodos n_j : $\forall n_j \in p(n_i) \Rightarrow f_2^*(n_j) \leq f(n_m)$

Por (2) se verifica que $f_2^*(n) \leq f_1^*(n) \leq f(n_m)$ ⁽⁴⁾

Por (3) y (4) se concluye que cualquier nodo expandido por $A^*(h_1^*)$ también será expandido por $A^*(h_2^*)$

Conclusión:

**Si disponemos de varias funciones heurísticas admisibles,
debemos elegir siempre la de mayor valor**

$$h^*(n) = \max_m [h_1^*(n), h_2^*(n), \dots, h_m^*(n)]$$

Tema 4. – Búsqueda con heurísticas fuertes

BÚSQUEDA HEURÍSTICA

Búsqueda con heurísticas débiles:

- Se usan heurísticas para guiar la búsqueda
- No para reducir el espacio de posibles soluciones

Búsqueda con heurísticas fuertes:

- Reduce el espacio de búsqueda de forma efectiva (en anchura o profundidad)
- Se dejan de analizar soluciones posibles pero improbables
- Es posible que no se encuentre la mejor solución
- Objetivo:
 - encontrar soluciones “buenas” (no necesariamente óptimas)
 - consiguen mejorar el rendimiento de forma substancial, pero optimalidad y completitud ya no están garantizadas

MÉTODOS CON HEURÍSTICAS FUERTES

Búsqueda por subobjetivos.

Búsqueda en línea.

- Búsqueda por ascenso de colinas (hill climbing)
- Búsqueda por horizonte.
- Optimización mediante búsqueda on-line.

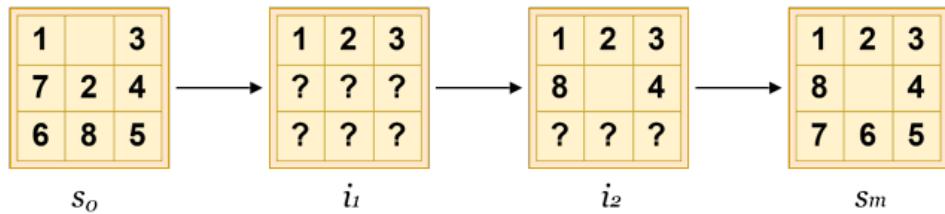
BÚSQUEDA POR SUBOBJETIVOS

Búsqueda guiada por subobjetivos (island-driven search)

- Idea: reducir la profundidad al subdividir el problema en varios problemas más pequeños mediante la aplicación de una heurística fuerte
- Determinar una secuencia de estados intermedios $i_1, i_2, i_3, \dots, i_n$ que “muy posiblemente” están en el camino óptimo
- Realizar búsquedas con un método base (amplitud, profundidad, A^* ...) desde el estado inicial al estado meta, a través de los estados intermedios: $s_0 \rightarrow i_1 \rightarrow i_2 \rightarrow i_3 \rightarrow \dots \rightarrow i_n \rightarrow s_m$

- Se aplica la heurística fuerte para elegir los estados intermedios

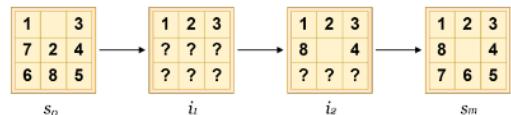
Ejemplo 8-puzzle, heurística fuerte: obtener filas correctas



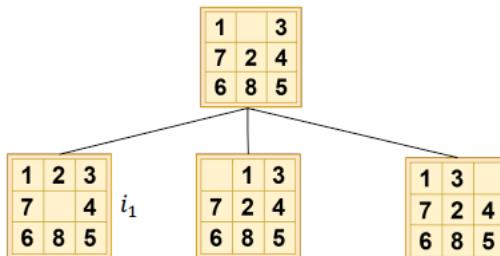
BÚSQUEDA POR SUBOBJETIVOS (EVITANDO CICLOS SIMPLES)

Método base: búsqueda en amplitud

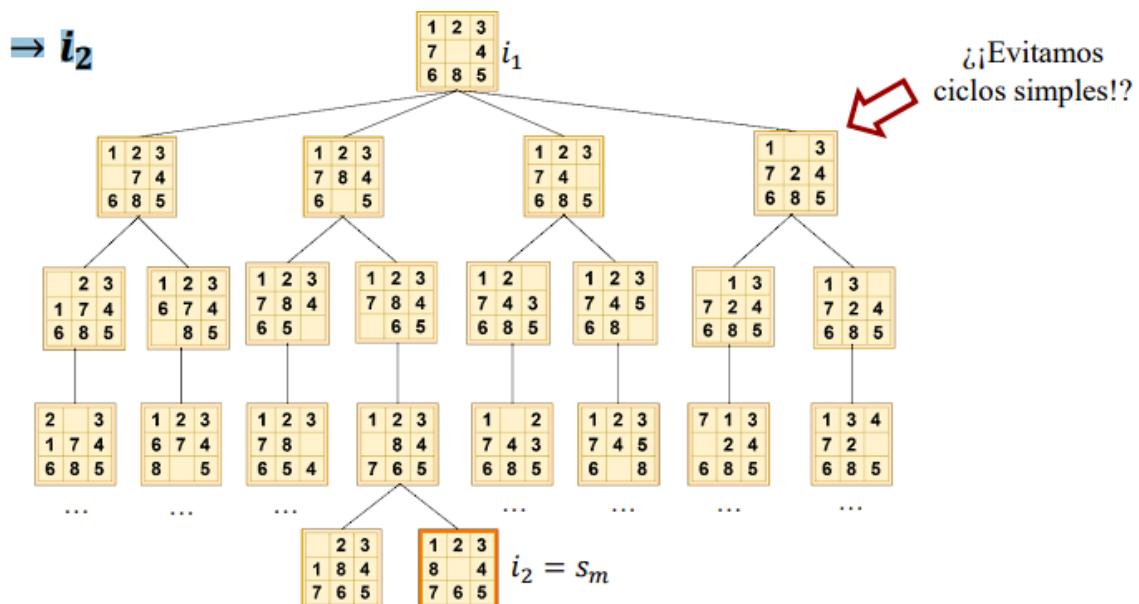
Heurística fuerte: obtener filas correctas



Camino $s_0 \rightarrow i_1$



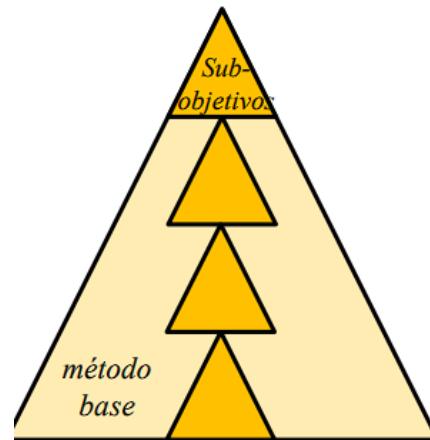
Camino $i_1 \rightarrow i_2$



BÚSQUEDA POR SUBOBJETIVOS: COMPLEJIDAD

Complejidad en espacio/tiempo:

- proporcional al número de nodos expandidos
- En general se obtiene una mejora de la complejidad respecto a los métodos de búsqueda que se usan como base
- la suma de varias búsquedas pequeñas es más eficiente que una búsqueda global
- depende de la selección inicial de los subobjetivos
- condición necesaria para una mejora: los caminos entre los pares de subobjetivos son “más cortos” que la solución global



BÚSQUEDA POR SUBOBJETIVOS: EJEMPLO COMPLEJIDAD

Parámetros:

- método base: búsqueda en amplitud
- factor de ramificación efectivo: b
- profundidad de la mejor solución: d
- x subobjetivos intermedios
- longitud de los caminos mínimos entre los subobjetivos: d'

Análisis del peor caso: nº de nodos expandidos

- Búsqueda en amplitud: $O(b^d)$
- Búsqueda por subobjetivos: $O(xb^{d'})$

Ejemplo:

- $b = 3; d = 20; x = 4; d' = 5$ (el camino encontrado tiene una longitud de 20)
- Búsqueda en amplitud: 3.486.784.401 nodos expandidos
- Búsqueda por subobjetivos: 972 nodos expandidos

BÚSQUEDA POR SUBOBJETIVOS: ANÁLISIS

Permite reducir de forma substancial la complejidad, especialmente en problemas de planes muy largos

- La búsqueda por subobjetivos no es necesariamente completa ni óptima.
 - Es completa si:
 - el método base es completo, y
 - los subobjetivos están en (por lo menos) un camino que lleva del estado inicial a la meta
 - Es óptima si:
 - el método base es óptimo, y
 - los subobjetivos están en este orden en el camino mínimo desde el estado inicial a la meta
- Es posible emplear búsquedas heurísticas como método base (p.e. A*)
 - En este caso se obtiene una reducción de complejidad mayor
 - Pero: ¡es necesario especificar funciones heurísticas para cada subobjetivo!
 - Adecuado en problemas con soluciones muy profundas (p.e. Cubo de Rubik, Ajedrez, 24-Puzzle, ...)

BÚSQUEDA EN LÍNEA

Búsqueda en línea (online search):

- denomina una clase de métodos que encadenan búsqueda (elección de acciones) y acción/percepción

Búsqueda Offline

- Percibir entorno
- Calcular plan
- Realizar plan

Búsqueda Online

- Repetir:
 - Percibir el entorno
 - Seleccionar la mejor acción
 - Realizar la acción

indicados cuando:

- El espacio de búsqueda es demasiado grande para buscar hasta la solución (y no se pueden aplicar las técnicas anteriores)
- No es realista usar un modelo determinista de los efectos de acciones
 - por frecuentes contingencias (p.e.: a veces el brazo deja caer un bloque)
 - porque no se dispone de un modelo del entorno (p.e.: un “mapa” completo)

medida de eficiencia:

- En general, no se puede asegurar optimalidad ni completitud
- Valorar el índice $\text{competitivo} = \frac{\text{coste del camino del agente en linea}}{\text{Coste del camino óptimo}}$
- El índice competitivo puede ser infinito, particularmente cuando hay acciones que no son reversibles

vamos a ver:

- Búsqueda por ascenso de colinas
- Búsqueda por horizonte
- Optimización mediante búsqueda on-line

BÚSQUEDA POR ASCENSO DE COLINAS (HILL CLIMBING)

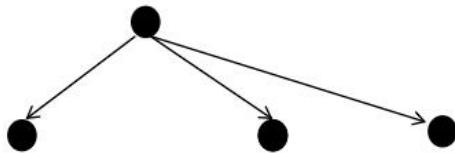
Idea subyacente: limitar la profundidad del árbol expandido en la búsqueda

- Generar un árbol de búsqueda de sólo **un** nivel
- Entre las hojas, elegir la más prometedora, y realizar la acción correspondiente
- Repetir el ciclo percepción/acción de forma continua

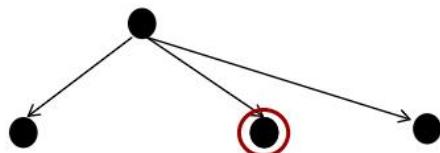


Repetir:

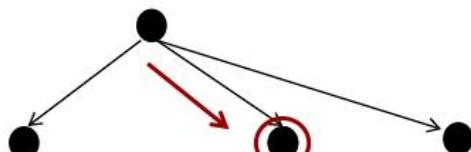
1. percibir estado
 2. expandir nodo
 3. elegir nodo más prometedor
 4. realizar acción
- Fin repetir



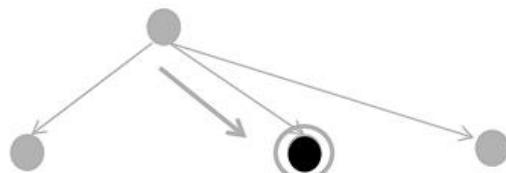
Repetir:
 1. percibir estado
 2. expandir nodo
 3. elegir nodo más prometedor
 4. realizar acción
 Fin repetir



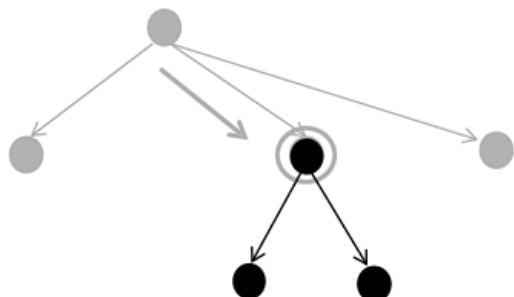
Repetir:
 1. percibir estado
 2. expandir nodo
 3. elegir nodo más prometedor
 4. realizar acción
 Fin repetir



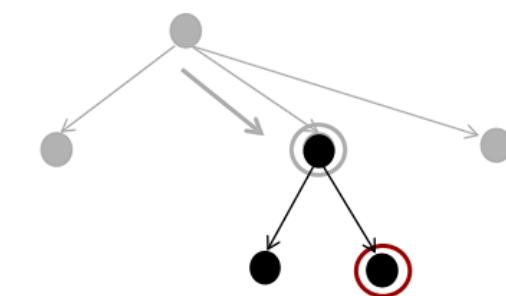
Repetir:
 1. percibir estado
 2. expandir nodo
 3. elegir nodo más prometedor
 4. realizar acción
 Fin repetir



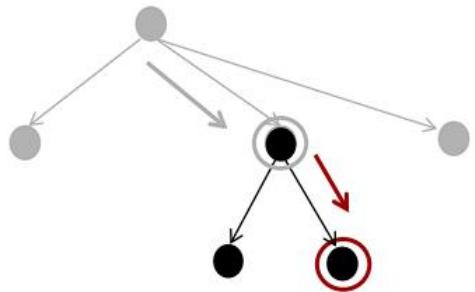
Repetir:
 1. percibir estado
 2. expandir nodo
 3. elegir nodo más prometedor
 4. realizar acción
 Fin repetir



Repetir:
 1. percibir estado
 2. expandir nodo
 3. elegir nodo más prometedor
 4. realizar acción
 Fin repetir



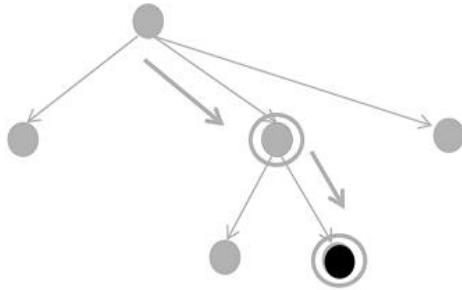
Repetir:
 1. percibir estado
 2. expandir nodo
 3. elegir nodo más prometedor
 4. realizar acción
 Fin repetir



Repetir:

1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
- 4. realizar acción**

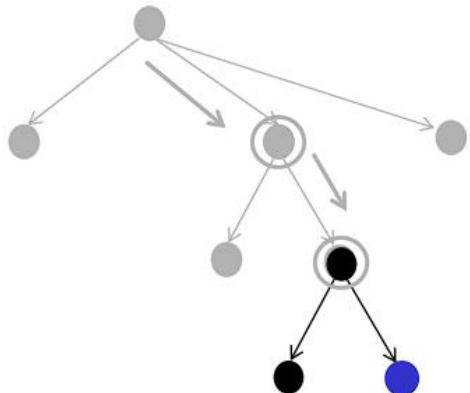
Fin repetir



Repetir:

- 1. percibir estado**
2. expandir nodo
3. elegir nodo más prometedor
4. realizar acción

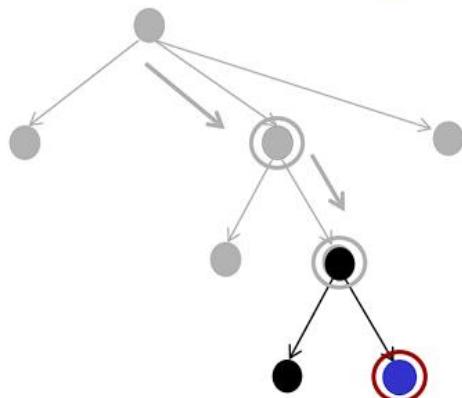
Fin repetir



Repetir:

1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
- 4. realizar acción**

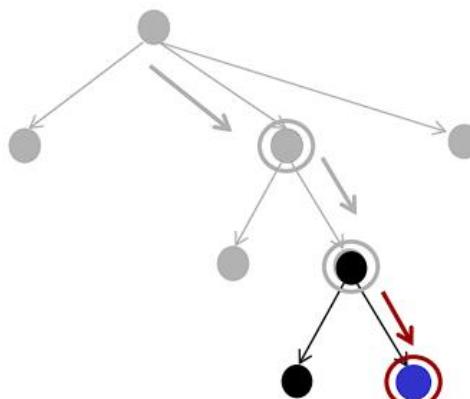
Fin repetir



Repetir:

1. percibir estado
2. expandir nodo
- 3. elegir nodo más prometedor**
4. realizar acción

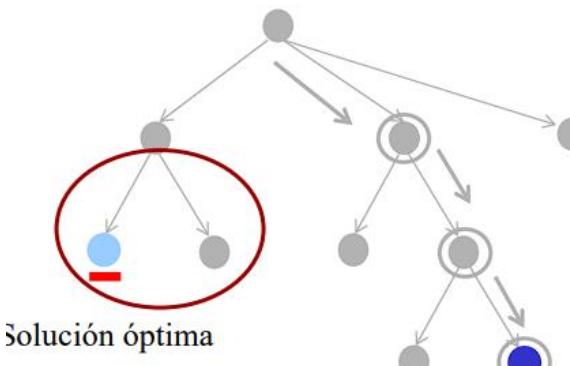
Fin repetir



Repetir:

1. percibir estado
2. expandir nodo
3. elegir nodo más prometedor
- 4. realizar acción**

Fin repetir

**Repetir:**

1. percibir estado
 2. expandir nodo
 3. elegir nodo más prometedor
 4. realizar acción
- Fin repetir**

Algoritmo:

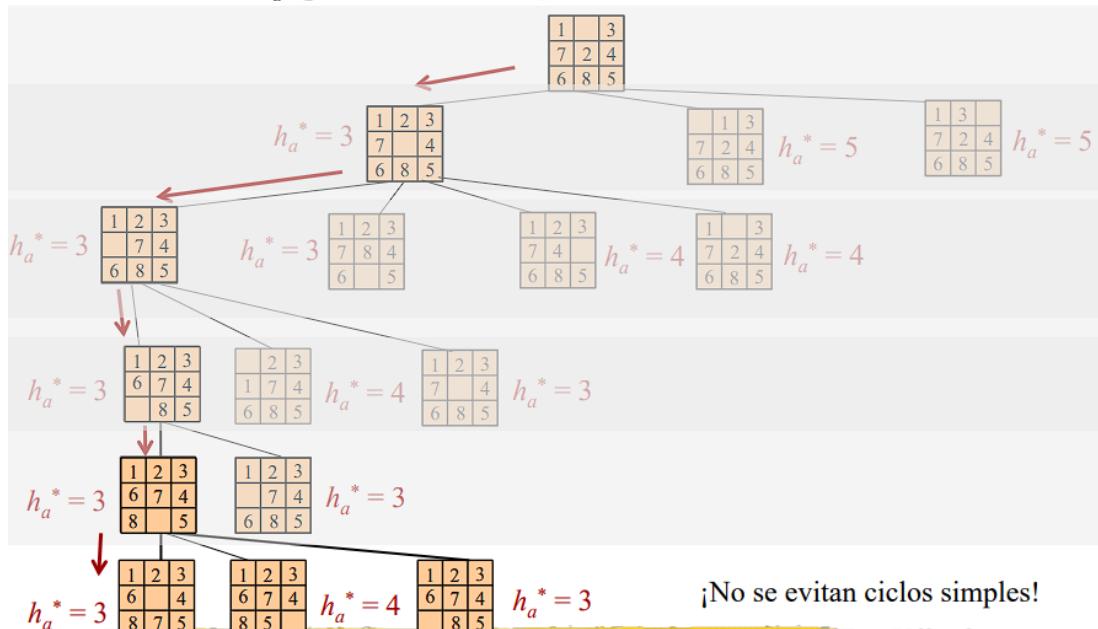
- h^* : función heurística que estima la distancia al nodo meta más cercano
- **percibir(entorno)**: devuelve el estado actual del problema
- **evaluar(n, h^*)**: calcula el valor de la función heurística h^* para el nodo n
 - debería comprobar si n es nodo meta
 - en este caso debería devolver el mínimo valor posible (normalmente $h^*(n)=0$)
- **acción(n, m)**: devuelve la acción que lleva de n a m
- **ejecutar($a, entorno$)**: efectúa la acción a en el entorno

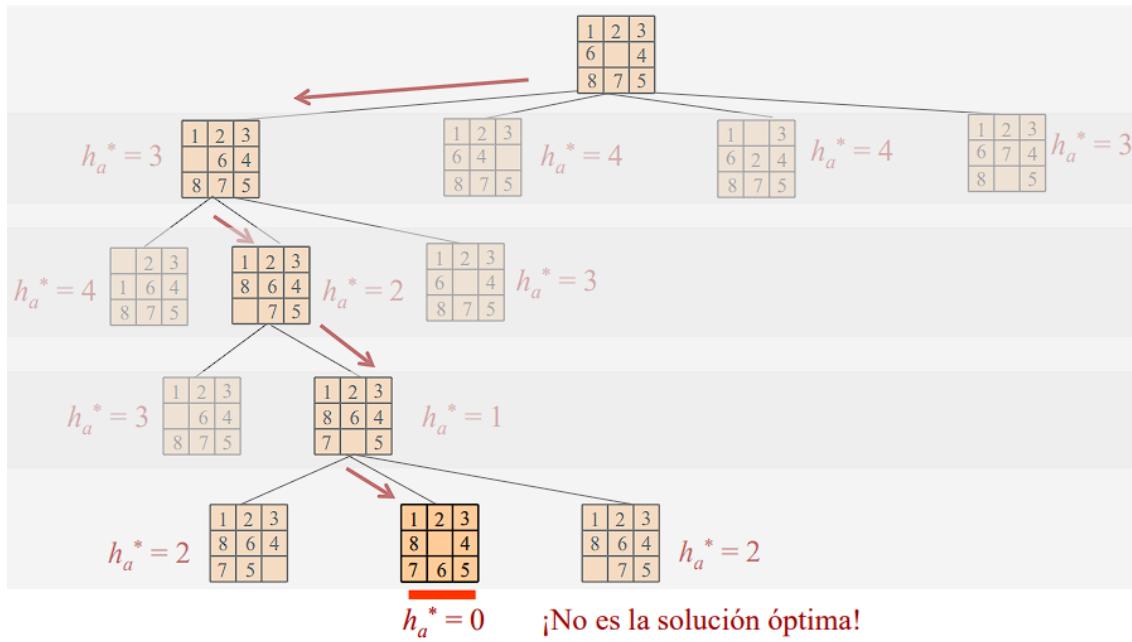
{búsqueda ascenso de colinas}

Repetir

```

nodo ← percibir(entorno)
Si meta?(nodo) entonces
    devolver(positivo)
    sucesores ← expandir(nodo)
    Si vacía?(sucesores) entonces
        devolver(negativo)
    mejor ← arg minn ∈ sucesores [evaluar( $n, h^*$ )]
    a ← acción( $n, mejor$ )
    ejecutar(a, entorno)
Fin {repetir}
  
```

EJEMPLOFunción heurística: h_a^* (piezas descolocadas)



BÚSQUEDA POR ASCENSO DE COLINAS: COMPLEJIDAD

Proporcional al número de nodos expandidos

- Parámetros:
 - d profundidad de la mejor solución,
 - b factor de ramificación efectivo
- Complejidad en espacio:
 - O(b) solo se mantiene en memoria los hijos de un nodo
- Complejidad en tiempo:
 - depende de la función heurística h^*
 - hay que contar la complejidad de calcular h^*
 - si $h^*(n) = h(n)$: complejidad en tiempo $O(d)$
 - la búsqueda “va directamente a la solución”
 - caso irreal, ya que significa que se conoce la solución de antemano
 - puede ser mayor que en la búsqueda por amplitud (si la función heurística es mala)

BÚSQUEDA POR ASCENSO DE COLINAS: ANÁLISIS

Análisis:

- Suele obtener buenos resultados en algunos casos, especialmente:
 - si no hay ciclos
 - la función heurística es muy buena
- En general, no se puede asegurar optimalidad ni completitud
- Si h^* es ideal ($h^*=h$, e.d. sin error heurístico), entonces es óptimo y completo
- A priori, no evitan ni siquiera ciclos simples
 - se podría evitar ciclos simples guardando el último estado conocido
- Se puede conseguir completitud:
 - si el conjunto de posibles estados es finito y
 - si se guardan todos los estados a los que se ha ido en el pasado:
 - cada vez que se llega a un estado repetido, se elige una acción que no se ha elegido anteriormente
 - pero, para eso hay que guardar todo el árbol hasta la meta y la ganancia en complejidad se pierde

BÚSQUEDA POR HORIZONTE

Idea subyacente:

- a mayor profundidad en el árbol de búsqueda, menor el error heurístico de h^*
- generar un árbol de búsqueda hasta un nivel k (horizonte), y determinar el mejor nodo a este nivel
- realizar la primera acción del plan que lleva a este nodo
- repetir el ciclo percepción/acción de forma continua

Algoritmo general:

- percibir el estado actual s
- aplicar un método de búsqueda base hasta el nivel k (o un nodo meta).
- sea H el conjunto de nodos hoja en el nivel k :

$$\text{Utilizar } h^* \text{ para determinar el "mejor" nodo hoja } n^* \in H: \quad n^* \leftarrow \arg \min_{n \in H} [h^*(n)]$$

- ejecutar la primera acción a^* en el camino que lleva a n^*
- repetir hasta que el agente se encuentra en un estado meta

EJEMPLO

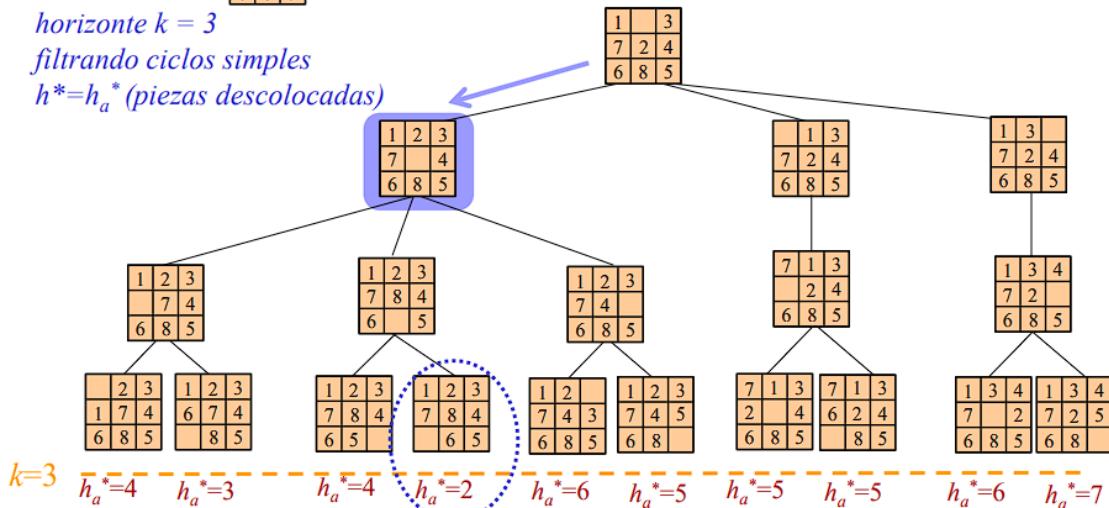
método base: búsqueda en profundidad limitada

estado actual = 

horizonte $k = 3$

filtrando ciclos simples

$h^* = h_a^*$ (piezas descolocadas)



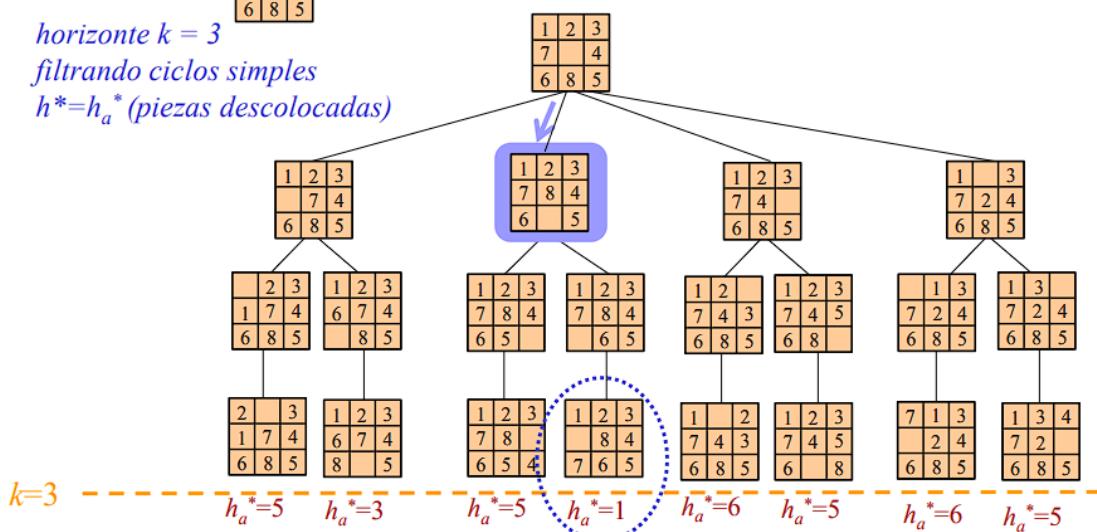
método base: búsqueda en profundidad limitada

estado actual = 

horizonte $k = 3$

filtrando ciclos simples

$h^* = h_a^*$ (piezas descolocadas)



método base: **búsqueda en profundidad limitada**

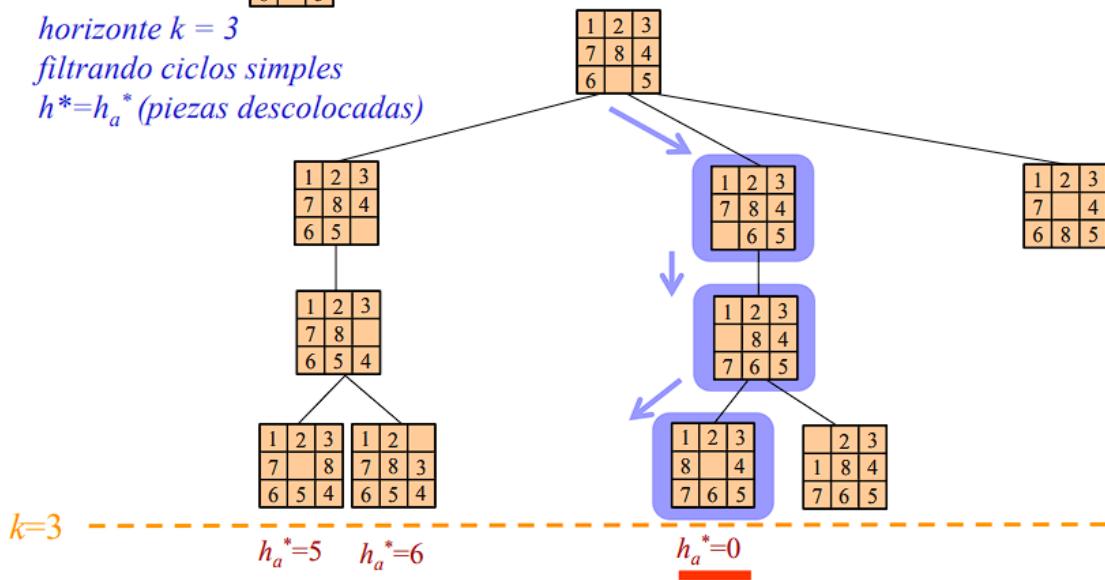
estado actual =

1	2	3
7	8	4
6		5

horizonte $k = 3$

filtrando ciclos simples

$h^* = h_a^*$ (piezas descolocadas)



BÚSQUEDA CON HORIZONTE: COMPLEJIDAD

Proporcional al número de nodos expandidos

Parámetros

- Método base: **búsqueda en profundidad limitada**
- b : factor de ramificación efectivo
- k : límite de profundidad

Complejidad en espacio:

- $O(b^k)$ (solo se mantiene el camino explorado y sus vecinos en la memoria)

Complejidad en tiempo:

- el encontrar un nodo meta y su profundidad depende de la función heurística
- hay que contar la complejidad de calcular h^*

Complejidad en tiempo:

- si se encuentra una solución con profundidad $d \leq k$:

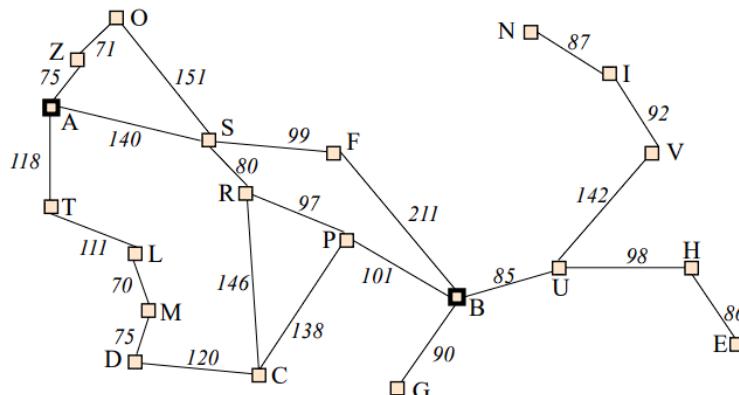
- igual que en la búsqueda en profundidad limitada:
- mejor caso: $O(d)$
- peor caso: $O(b^k)$

- si se encuentra una solución con profundidad $d > k$:
 - primero hay que realizar $d - k$ búsquedas en profundidad limitada completas
 - en el siguiente paso se encuentra la solución en la profundidad k
 - Mejor y peor caso: $O(d * b^k)$

Dominios con operadores de coste diferente:

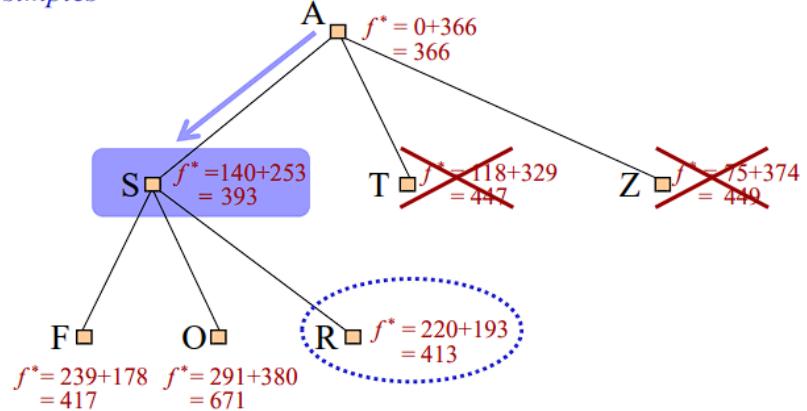
- utilizar f^* para evaluar la calidad de los nodos a nivel k ,
- sea H el conjunto de nodos hoja en el nivel k
- mantener una variable α que contiene el mínimo valor de f^* encontrado en el nivel k hasta el momento inicializar $\alpha = \infty$
- para cada nodo n (de nivel k) realizar: $\alpha = \min(\alpha, f^*(n))$
- Optimización mediante podas α
 - se puede aplicar si se sabe con seguridad que los valores de f^* crezcan siempre al ir de un nodo a otro (**f^* consistente**)
 - durante el proceso de búsqueda en profundidad limitada, abandonar cualquier rama a partir de un nodo n con $f^*(n) \geq \alpha$

FUNCIÓN HEURÍSTICA PARA ENCONTRAR RUTAS



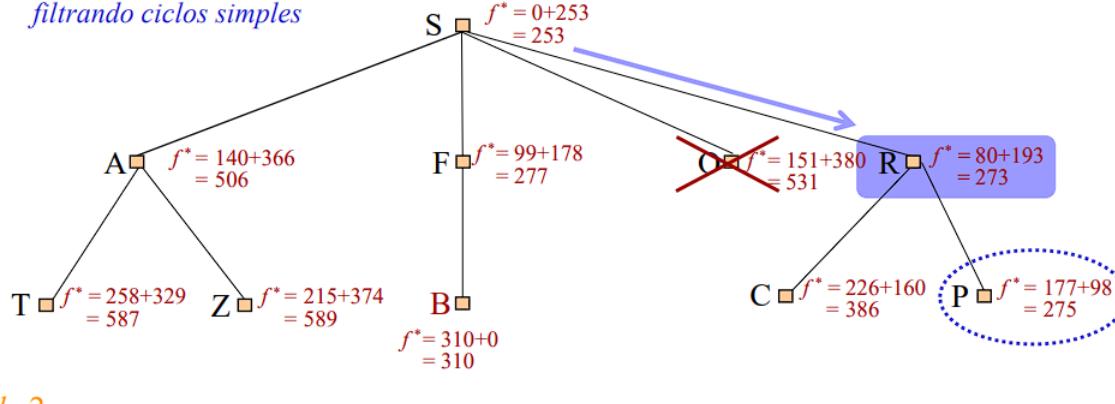
	h^*
A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374

estado actual $s = A$
horizonte $k = 2$
filtrando ciclos simples



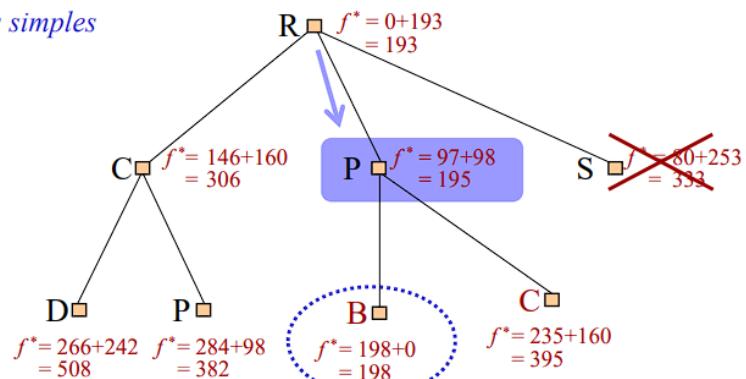
$k=2$ -----
 $\alpha = 417 \quad \alpha = 417 \quad \alpha = 413$

estado actual $s = S$
horizonte $k = 2$
filtrando ciclos simples



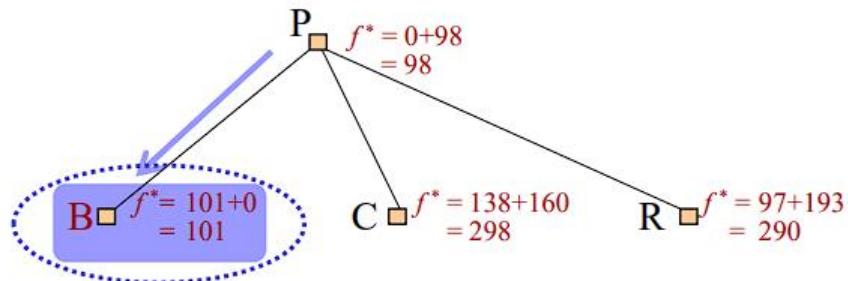
$k=2$ -----
 $\alpha = 587 \quad \alpha = 587 \quad \alpha = 310 \quad \alpha = 310 \quad \alpha = 275$

estado actual $s = R$
horizonte $k = 2$
filtrando ciclos simples



$k=2$ -----
 $\alpha = 508 \quad \alpha = 382 \quad \alpha = 198 \quad \alpha = 198$

*estado actual $s = P$
horizonte $k = 2$
filtrando ciclos simples*



$k=2$ -----

BÚSQUEDA CON HORIZONTE: ANÁLISIS

En general, no se puede asegurar optimalidad ni completitud

- depende de la función heurística
- Si h^* es ideal ($h^* = h$, e.d. error heurístico 0), entonces es óptimo y completo
- La búsqueda por ascenso de colinas es un caso especial de la búsqueda con horizonte (horizonte $k = 1$)
- Aumentar el horizonte k :
 - permite evitar ciclos de longitud k o menor
 - suele mejorar la calidad de la solución, puesto que la evaluación de los nodos a nivel k lleva menor error heurístico

OPTIMIZACIÓN MEDIANTE BÚSQUEDA ON-LINE

La búsqueda online es aplicable a problemas de optimización

Problema de optimización:

- el objetivo es encontrar el mejor estado según una función objetivo
- **no necesariamente se busca un estado exacto**, sino uno que se acerca al máximo al objetivo
- el camino para llegar al estado buscado puede ser irrelevante (coste 0)

Ejemplos: juegos lógicos y de configuración, n-reinas

Funciones heurísticas para problemas de optimización:

- Funciones que estimen el coste del camino hasta el nodo meta más próximo (todos los que hemos visto hasta ahora)
- Funciones que miden la calidad de un nodo respecto a una función objetivo (“funciones de evaluación”)
- A veces es más fácil definir una heurística que estime la “calidad” de los nodos

EL JUEGO DE LAS CIFRAS

Combinar una serie de CIFRAS mediante las operaciones de suma y multiplicación de tal forma que el resultado se acerca lo máximo a un número dado (EXACTO).

- Ejemplo:

CIFRAS: 6 2 5 25

EXACTO: 420

El objetivo consiste en encontrar una secuencia de operaciones, que aplicadas sobre (algunas de) las cifras de entrada, y sobre los resultados intermedios, permiten obtener un número lo más próximo a EXACTO.

Cada cifra se puede utilizar sólo una vez.

FORMALIZACIÓN DEL PROBLEMA:

Representación (eficiente) de estados:

(<último resultado>, <lista de los números disponibles todavía>)

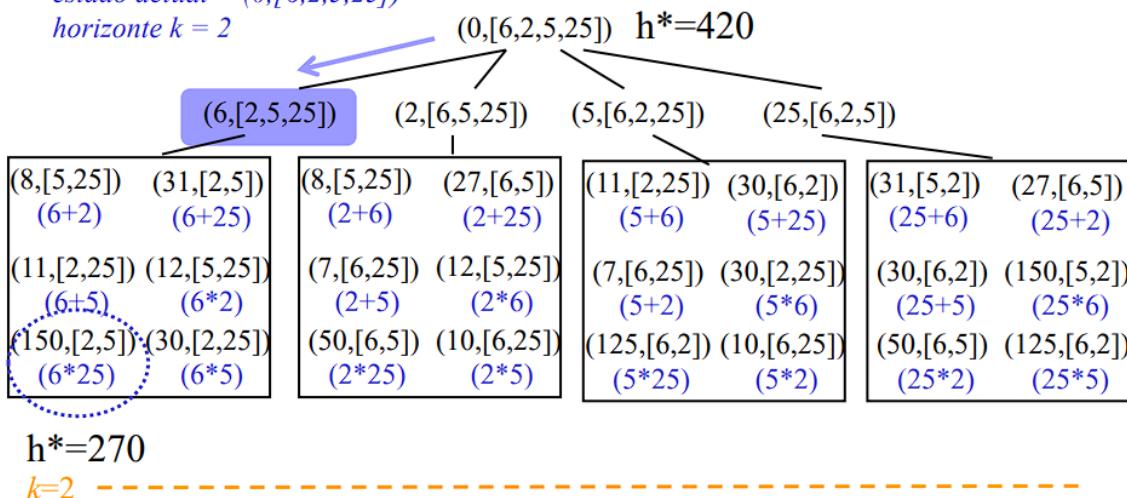
- Estado inicial: (0, [6,2,5,25])
- Estado meta: (x,y) tal que x es lo más cerca posible de 420
- Operadores: aplicar la suma/multiplicación
- Coste de un operador: 0
- Tipo de solución: estado
- Se puede resolver el problema con un método de búsqueda no informado usando como estado meta (420,y), pero ¿Qué pasa si no hay ninguna solución exacta?
- Mejor utilizar la búsqueda online:
 - Función heurística: $h^*((x,y)) = |420-x|$
 - mide la similitud del estado al objetivo, no mide el coste del camino restante
- Modificar los algoritmos:

- se termina cuando el estado elegido como más prometedor tiene un valor de h^* mayor que el estado actual

método base: búsqueda en profundidad limitada

estado actual = $(0,[6,2,5,25])$

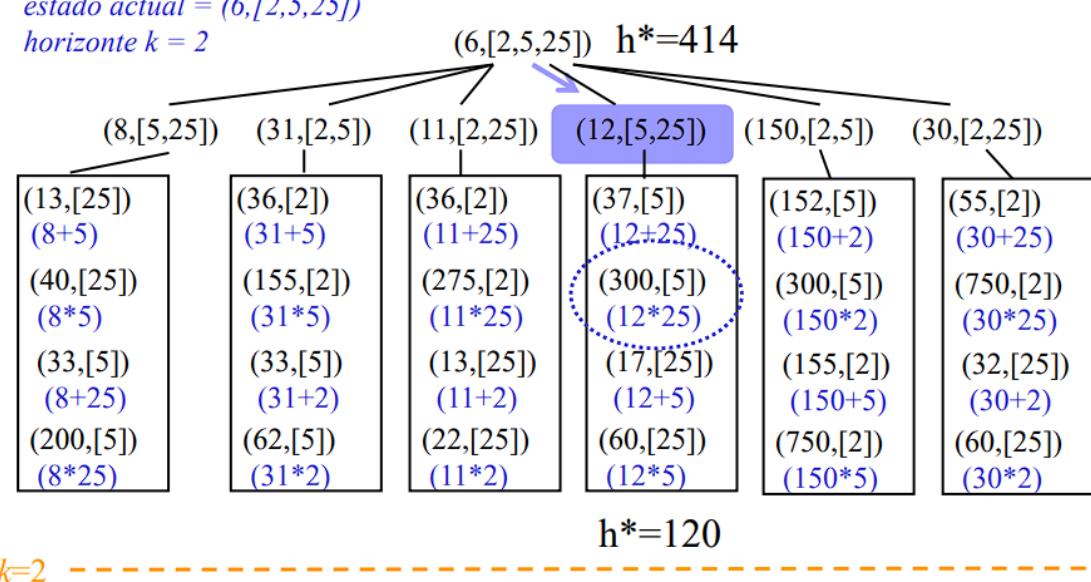
horizonte $k = 2$



método base: búsqueda en profundidad limitada

estado actual = $(6,[2,5,25])$

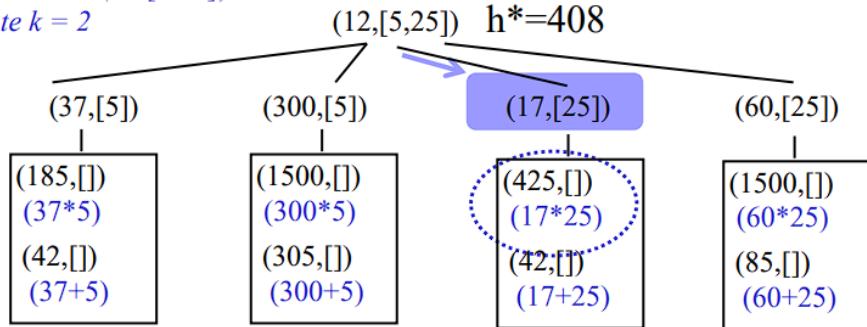
horizonte $k = 2$



método base: búsqueda en profundidad limitada

estado actual = $(12, [5, 25])$

horizonte $k = 2$



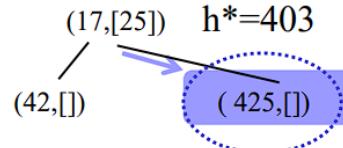
$h^* = 5$

$k=2$

método base: búsqueda en profundidad limitada

estado actual = $(17, [25])$

horizonte $k = 2$



$h^* = 5$

método base: búsqueda en profundidad limitada

estado actual = $(425, [])$

horizonte $k = 2$

$(425, [])$

BÚSQUEDA ON-LINE: COMENTARIOS

Comentarios:

- Aplicable en entornos inaccesibles o dinámicos
 - Ejemplos: Laberintos, Robot moviéndose en un entorno real, ...
- Útil en tareas de optimización, donde se quiere mejorar el estado continuamente
 - Ejemplo: Controlar los parámetros de procesos industriales, etc.

Los algoritmos de búsqueda on-line pueden ser empleados de forma “off-line”

- Buscar un plan completo para un problema:

- Solo simular las acciones, pero no efectuarlas en el entorno
- No es posible en entornos inaccesibles o dinámicos
- Ventaja: reducción de complejidad

Tema 5. – Búsqueda Multiagente

PROBLEMAS MULTIAGENTE

Hasta ahora, agentes individuales

- Un solo agente en el entorno.
- El agente controla qué acciones puede ejecutar.
- El agente es el único que interactúa con el entorno.
- Estado determinista o casi determinista
- Objetivo individual, encontrar u optimizar.

Problemas multiagente

- Múltiples agentes en el entorno.
- Cada agente controla sus acciones. Un agente no controla las acciones de los otros, aunque hasta cierto punto puede predecirlas.
- Todos los agentes interactúan con el entorno.
- Estado principalmente indeterminista.
- Objetivos individuales o colectivos, cooperativos o enfrentados.

TIPOS DE PROBLEMAS MULTIAGENTE

SEGÚN SUS OBJETIVOS

COOPERATIVOS

Los agentes trabajan en común para conseguir metas y objetivos.

Ejemplos: robots colaborativos, redes de sensores distribuidos...

COMPETITIVOS

Los agentes compiten unos con otros para conseguir metas individuales, o ganar una ventaja competitiva.

Ejemplos: juegos competitivos (ajedrez, go, ...), escenarios adversarios en seguridad, ...

MIXTOS

Algunos objetivos comunes, otros enfrentados. Equipos de agentes colaborando, enfrentados a otros equipos de agentes.

Ejemplos: juegos tipo captura la bandera, juegos deportivos, ...

SEGÚN EL NÚMERO DE AGENTES

PROBLEMAS PARA DOS AGENTES, BILATERALES

Involucra a dos agentes.

Ejemplos: ajedrez, damas, go.

PROBLEMAS PARA MÚLTIPLES AGENTES

Involucran más de dos agentes.

Ejemplos: Agentes para juegos de equipo, LoL, Counter Strike, drones coordinados.

SEGÚN EL AZAR INVOLUCRADO

Juegos con elementos de azar (backgammon, poker)

Juegos sin elementos de azar (damas, ajedrez)

SEGÚN LA INFORMACIÓN QUE MANEJAN LOS AGENTES

JUEGOS CON INFORMACIÓN PERFECTA

Los agentes tienen información completa del estado del juego y del resultado de las acciones de los otros agentes.

Ejemplos: damas, ajedrez.

JUEGOS CON INFORMACIÓN INCOMPLETA

Los agentes tienen información incompleta o imperfecta del estado del juego, o de la información privada de los otros agentes.

Ejemplos: poker, LoL, Counter Strike.

JUEGOS DE SUMA CERO

Un/os agente/s solo pueden ganar algo a costa de otro agente/s.

La suma de las ganancias de unos agentes y las pérdidas de los otros es “cero”.

Originario de la teoría económica de juegos.

Tipos de juegos de suma cero:

- Monopoly: competitivo, múltiples agentes, información incompleta, con elementos de azar.
- Damas, Ajedrez: competitivo, bilateral, información completa, sin elementos de azar.
- FIFA: mixto, múltiples agentes, información completa, sin elementos de azar.

Trabajaremos con juegos de dos jugadores, competitivos, con información perfecta, con y sin elementos de azar

MODELADO DE JUEGOS BIPERSONALES

Conocimientos mínimos **a priori** de los agentes **min** y **max**:

s_0	posición <i>inicial</i>
$\text{expand } s \mapsto \{s_1, s_2, s_3, \dots, s_n\}$	Acciones permitidas en la posición s Supone implícitamente que los jugadores se alternan al realizar las jugadas
$\text{terminal? } s \mapsto \text{true} \mid \text{false}$	Prueba si s es un nodo terminal
$U \ s \mapsto k, k \in \mathbb{R}$	Función de <i>utilidad</i> del juego, solo para los estados terminales s_t <ul style="list-style-type: none"> • Gana max $U(s) = +\infty$ • Gana min $U(s) = -\infty$ • Empate $U(s) = 0$

EJEMPLO: TRES EN RAYA

Características

- dos jugadores, **max** (X) y **min** (O).
- competitivo, uno gana o hay empate
- sin elementos de azar
- con información completa

X O X O X O O O X

ganador
max

Reglas

- tablero inicial vacío
- empieza colocando una ficha **max** y los jugadores se alternan para poner las fichas restantes
- gana **max** si consigue colocar tres X en línea
- gana **min** si consigue colocar tres O en línea
- si se ocupan todas las casillas sin que haya un ganador, se produce un empate

O O X O X O O X X

ganador
min

X O X O X O O X O

empate

Estados:

$$S = (e, l), e = \begin{bmatrix} v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 \\ v_7 & v_8 & v_9 \end{bmatrix}, v_i \in \{x|o|-\}, l \in \{\text{max}, \text{min}\}$$

Estado inicial:

$$s_0 = (\begin{bmatrix} - & - & - \\ - & - & - \\ - & - & - \end{bmatrix}, \text{max})$$

Función expandir:

$$\text{expand } s(e, l) \mapsto \{(e_1, l), \dots, (e_n, l)\}, v_i \in e_l, v_i = (-, l) \mapsto \begin{cases} (x, \text{min}) \Leftrightarrow l = \text{max} \\ (0, \text{max}) \Leftrightarrow l = \text{min} \end{cases}$$

Función terminal:

$$\text{terminal? } s = (e, l), \forall v \in e, v \neq -$$

Función de utilidad:

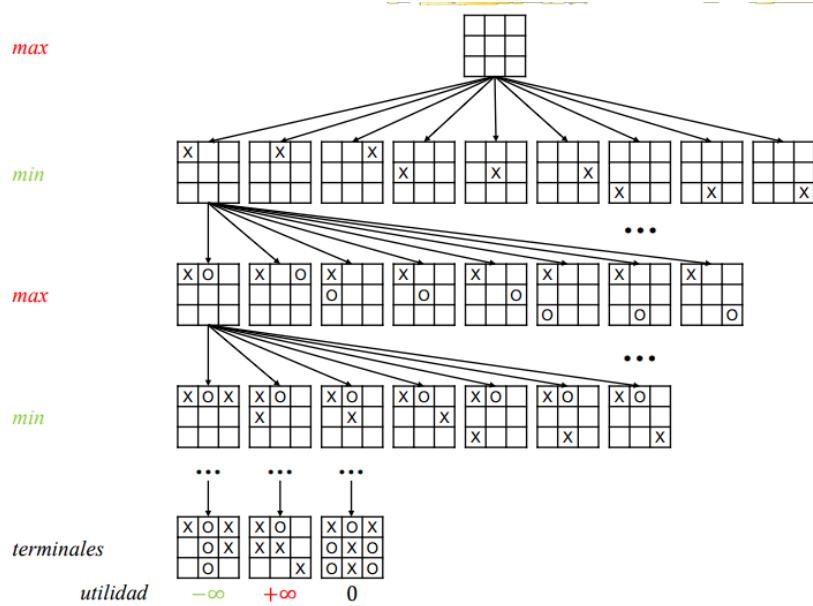
$$U(s) = k, s = (e, l), \{v_{l1} \in e, v_{l2} \in e, v_{l3} \in e\} \\ \in (\{v_1, v_2, v_3\}, \{v_4, v_5, v_6\}, \{v_7, v_8, v_9\}, \{v_1, v_4, v_7\}, \{v_2, v_5, v_8\}, \{v_3, v_6, v_9\}, \{v_1, v_5, v_9\}, \{v_3, v_5, v_7\}), \\ k = \begin{cases} \infty \Leftrightarrow v_{l1} = v_{l2} = v_{l3} = x \\ -\infty \Leftrightarrow v_{l1} = v_{l2} = v_{l3} = o \\ 0 \end{cases}$$

ÁRBOLES DE JUEGO

Sea un **conjunto de nodos** $N \in \mathbb{N} \times \mathbb{N}$ que representan los estados del tablero, un **conjunto de aristas** E que representan los posibles movimientos para un nodo, **etiquetas** $L = \{\text{max}, \text{min}\}$, el **nodo raíz** r , el **árbol de juego** $G = (N, E, L, r)$ es un grafo dirigido que cumple:

- r está etiquetado **max** están etiquetados **min**.
- todos los **sucesores** de **min** están etiquetados **max**.
- cada nivel del árbol representa un **ply** (una tirada o media jugada).
 - en los nodos **etiquetados max** es el **turno** de **max**.
 - en los nodos **etiquetados min** es el **turno** de **min**.
- un árbol de juego **completo** representa todas las posibles tiradas/movimientos del juego, **todas sus hojas son terminales**.
- un árbol de juego **parcial** es un subconjunto de un árbol completo.

EJEMPLO: ÁRBOL DE JUEGO PARA TRES EN RAYA



ESTRATEGIAS

Problema del agente max: ¿cómo determinar su mejor jugada?

- **max** podría aplicar métodos de búsqueda estándar, usando las posiciones en las que él gana como estados meta.
- pero **min** realizaría jugadas que llevarían a **max** a estados no planeados.

Estrategia general:

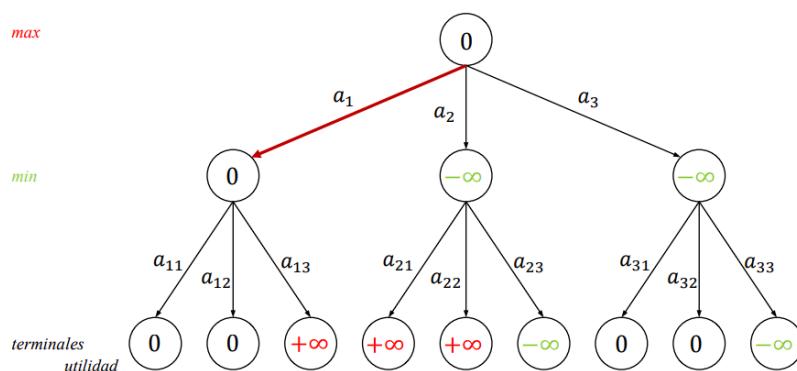
- **max** define el árbol completo, es decir, dando respuesta a todas las posibles jugadas de **min**.
- **max** escoge un subárbol del árbol de juego al hacer su movimiento.

Estrategia óptima (agente racional): ¿Cómo escoge **max**?

- la estrategia óptima es la que implica el mejor resultado **garantizado**.
- es un juego de suma cero con agentes racionales, **max** puede asumir que **min** hará lo **mejor** para sí mismo, que es lo **peor** para **max**.
- estrategia **minimax**: maximizar la utilidad mínima en cada jugada.

MÉTODO MINIMAX

1. Generar el árbol de juego completo
2. Aplicar la función de utilidad en cada nodo terminal
3. Propagar las utilidades hacia arriba
 - en los nodos **max**, usar la utilidad máxima de los sucesores.
 - en los nodos **min**, usar la utilidad mínima de los sucesores.
4. Los valores de utilidad llegan al nodo raíz (**max**).
5. La jugada óptima de **max** es la que lleva al sucesor de utilidad máxima.

EJEMPLO

5. La jugada óptima de **max** lleva al sucesor de utilidad máxima

Algoritmo Minimax (versión preliminar)**Algoritmo minimax (preliminar)**

```

1: procedure MAXVALOR(state)
2:   if terminal? state then
3:     return U state
4:   end if
5:   successorStates  $\leftarrow$  expand state
6:    $\alpha \leftarrow -\infty$ 
7:   for all successor in successorStates do
8:      $\alpha \leftarrow \max(\alpha, \text{MinValor}(\text{succesor}))$ 
9:   end for
10:  return  $\alpha$ 
11: end procedure
12:
13: procedure MINVALOR(state)
14:   if terminal? state then
15:     return U state
16:   end if
17:   successorStates  $\leftarrow$  expand state
18:    $\beta \leftarrow +\infty$ 
19:   for all successor in successorStates do
20:      $\beta \leftarrow \min(\beta, \text{MaxValor}(\text{succesor}))$ 
21:   end for
22:   return  $\beta$ 
23: end procedure

```

Algoritmo:

- funciones **MaxValor** y **MinValor** mutuamente recursivas
- *estado* es el estado actual
- α máximo de la utilidad de los sucesores de un nodo **max**
- β mínimo de la utilidad de los sucesores de un nodo **min**

DECISIONES IMPERFECTAS

Problema: crecimiento **exponencial** del árbol de juego.

- incluso en juegos muy simples, es imposible desarrollar el árbol de juego completo hasta todos sus nodos terminales

Solución: heurísticas

- sustituir la prueba terminal por una prueba suspensión (stop) que detiene la búsqueda aún sin llegar a una posición terminal:

- límite de profundidad fijo, posiciones “en reposo”, ...

- aplicar en lugar de la U una función de evaluación (e), que estime la utilidad esperada del juego correspondiente a una posición s determinada

- e debe coincidir con la función de utilidad u en los nodos terminales

- Ajedrez: $e(s) = \text{“suma de los valores materiales en } s\text{”}$

- Tres en Raya: $e(s) = \text{“nº de líneas abiertas para líneas } \max \text{ en } s\text{”}$

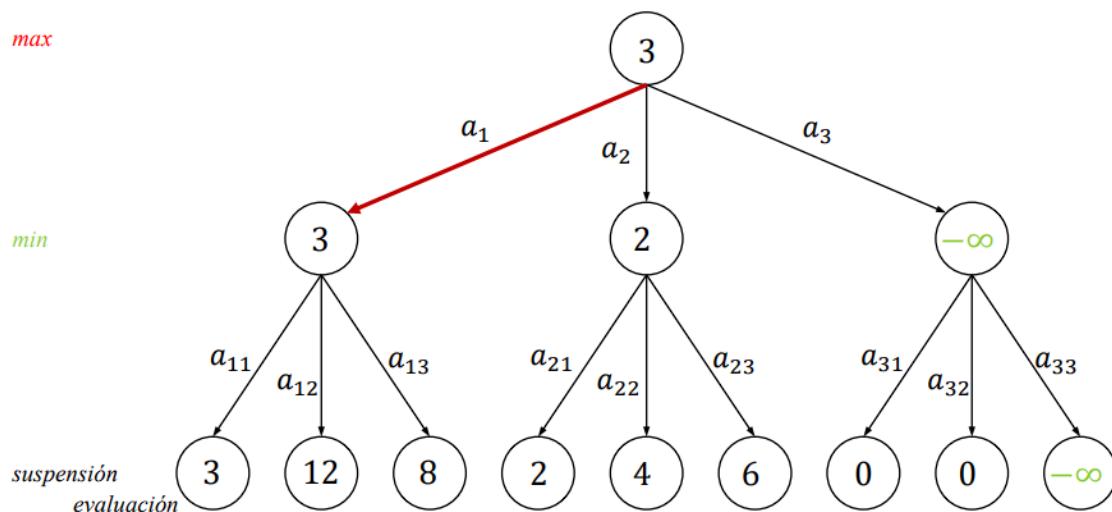
- “nº de líneas abiertas para líneas **min** en s ”

- suele ser función lineal ponderada: $e(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$

- heurísticas fuertes:

- las jugadas del Minimax ya no serán óptimas, pero dependerán de la calidad de las heurísticas.

EJEMPLO ESTRATEGIA MINIMAX



Algoritmo *Minimax* (versión definitiva)

Algoritmo minimax

```

1: procedure MAXVALOR(state)
2:   if stop? state then
3:     return e state
4:   end if
5:   successorStates ← expand state
6:   α ← -∞
7:   for all successor in successorStates do
8:     α ← max(α, MinValor(successor))
9:   end for
10:  return α
11: end procedure

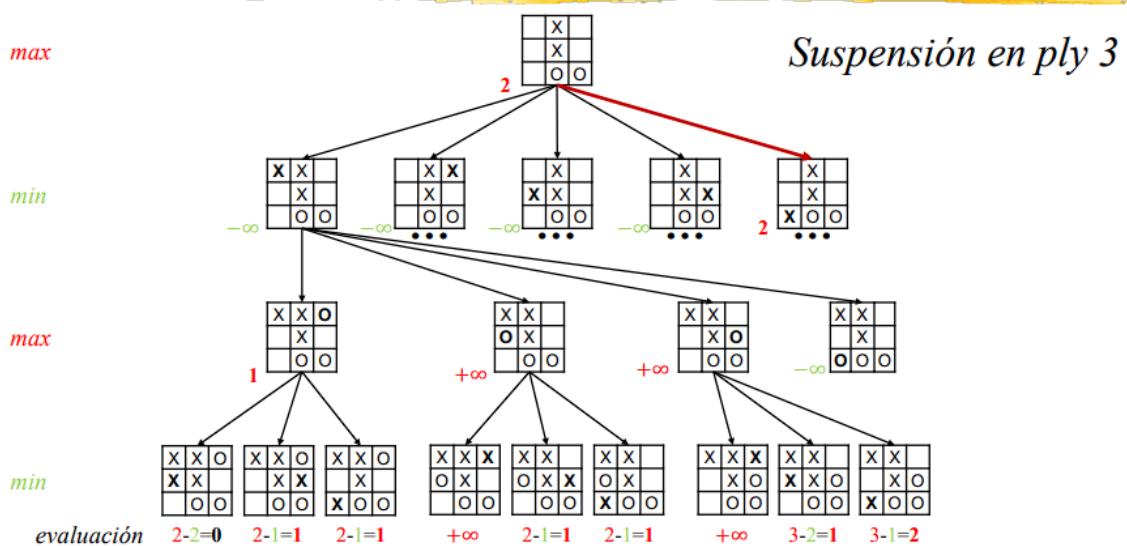
12:
13: procedure MINVALOR(state)
14:   if stop? state then
15:     return e state
16:   end if
17:   successorStates ← expand state
18:   β ← +∞
19:   for all successor in successorStates do
20:     β ← min(β, MaxValor(successor))
21:   end for
22:   return β
23: end procedure

```

Algoritmo:

- funciones *MaxValor* y *MinValor* mutuamente recursivas
 - *estado* es el estado actual
 - α máximo de la utilidad de los sucesores de un nodo max
 - β mínimo de la utilidad de los sucesores de un nodo min
 - *terminal* \mapsto stop
 - $U \mapsto e$

Ejemplo: Tres en Raya



RESUMEN MINIMAX

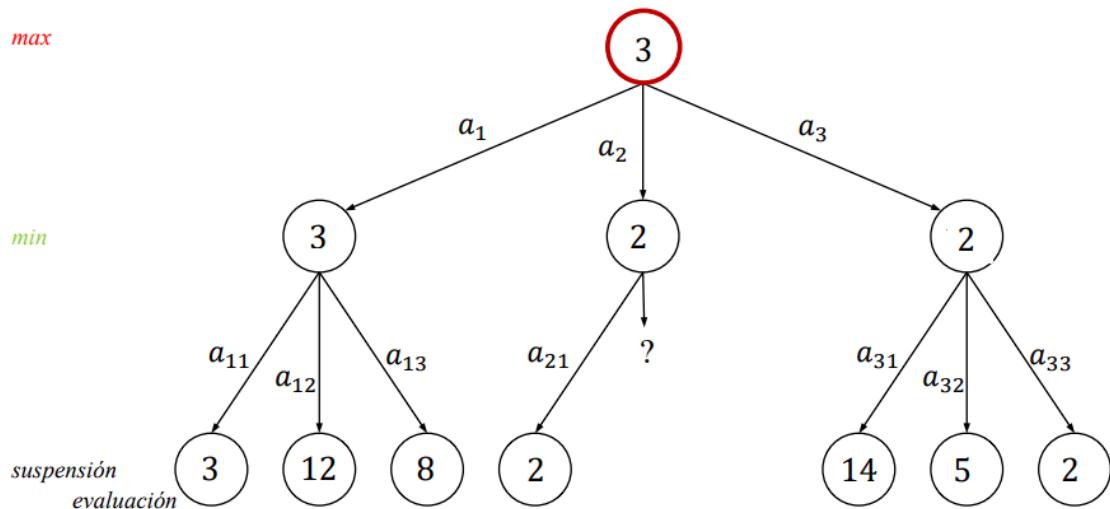
Complejidad:

- Minimax genera el árbol de juego hasta el nivel de suspensión
- Complejidad: $O(b^d)$ (factor de ramificación b; número de plies explorados d)

Extensiones del Minimax:

- mejorar la complejidad (poda $\alpha-\beta$: descartar nodos irrelevantes)
- juegos con elementos de azar (ExpectMinimax: hay un nuevo jugador “azar”)
- juegos con información parcial (búsqueda en estados de creencias)
- mejorar las heurísticas:
 - aprender funciones de evaluación y suspensión (p.e. por el “efecto horizonte”)
- relajar la suposición de racionalidad perfecta de min:
 - suponga que **max** está a punto de perder si **min** juega de forma óptima.
 - sin embargo, hay una jugada que hace ganar a **max**, si **min** hace un solo error.

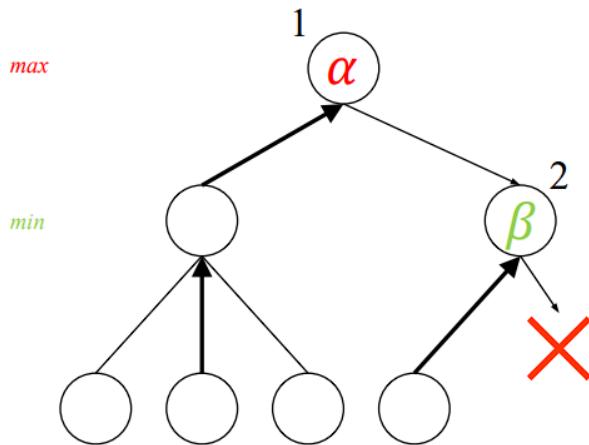
Poda $\alpha - \beta$



Intuición: a veces es posible estimar el valor de un nodo sin tener que evaluar todos sus sucesores

Poda α :

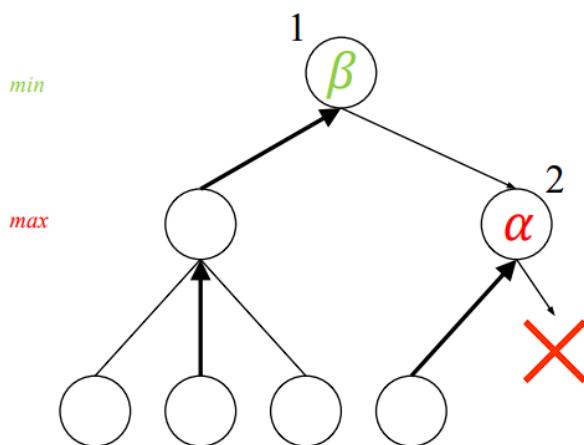
- α : evaluación más alta encontrada, *hasta el momento*, en un nodo **max**
- β : evaluación más baja encontrada, *hasta el momento*, en su sucesor directo **min**



- La e del nodo **min** será como mucho β , $e(2) \leq \beta$
- El valor del nodo **max** será $\alpha = \max(\alpha, \beta)$
- Si $\alpha \geq \beta$ el valor del nodo **max** no depende de la evaluación del nodo **min**, por lo que no es necesario explorar los sucesores restantes de **min**

Poda β :

- β : evaluación más baja encontrada, *hasta el momento*, en un nodo **min**
- α : evaluación más alta encontrada, *hasta el momento*, en su sucesor directo **max**



- La e del nodo **max** será como poco α , $e(2) \geq \alpha$
- El valor del nodo **min** será $\beta = \min(\beta, \alpha)$
- Si $\beta \leq \alpha$, el valor del nodo **min** no depende de la evaluación del nodo **max**, por lo que no es necesario explorar los sucesores restantes de **max**

Algoritmo Minimax con poda $\alpha-\beta$

Algoritmo minimax con poda $\alpha - \beta$

```

1: procedure MAXVALOR(state,  $\alpha, \beta$ )
2:   if stop? state then
3:     return e state
4:   end if
5:   successorStates  $\leftarrow$  expand state
6:   for all successor in successorStates do
7:      $\alpha \leftarrow \max(\alpha, \text{MinValor}(\text{succesor}, \alpha, \beta))$ 
8:     if  $\alpha \geq \beta$  then
9:       return  $\alpha$ 
10:      end if
11:    end for
12:   return  $\alpha$ 
13: end procedure
14:
15: procedure MINVALOR(state,  $\alpha, \beta$ )
16:   if stop? state then
17:     return e state
18:   end if
19:   successorStates  $\leftarrow$  expand state
20:   for all successor in successorStates do
21:      $\beta \leftarrow \min(\beta, \text{MaxValor}(\text{succesor}, \alpha, \beta))$ 
22:     if  $\beta \leq \alpha$  then
23:       return  $\beta$ 
24:     end if
25:   end for
26:   return  $\beta$ 
27: end procedure

```

Si $\alpha \geq \beta$ el valor del nodo max no depende de la evaluación del nodo min, por lo que no es necesario explorar los sucesores restantes de min

Si $\beta \leq \alpha$, el valor del nodo min no depende de la evaluación del nodo max, por lo que no es necesario explorar los sucesores restantes de max

RESUMEN MINIMAX CON PODA A-B

Análisis:

- El algoritmo Minimax con poda $\alpha-\beta$ siempre produce el mismo resultado que sin la poda
- La eficiencia de Minimax con poda $\alpha-\beta$ depende del orden en el que se exploran los nodos
- Complejidad (factor de ramificación b; número de plies explorados d)
 - Minimax sin poda $\alpha-\beta$: $O(b^d)$ (p.e. $10^4 = 10.000$ nodos)
 - Minimax poda $\alpha-\beta$ (mejor caso): $O(b^{(d/2)})$ (p.e. $10^2 = 100$ nodos)
 - Minimax poda $\alpha-\beta$ (caso medio): $O(b^{(3d/4)})$ (p.e. $10^3 = 1.000$ nodos)

JUEGOS BIPERSONALES CON ELEMENTO DE AZAR

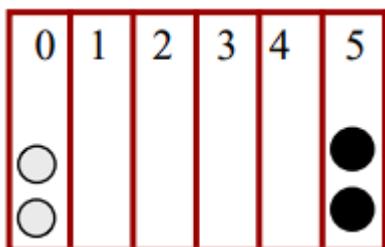
Algoritmo: ExpectMinimax

Idea:

- Utilizar el algoritmo Minimax
- Añadir un nuevo jugador: “azar” que se incluye en el árbol siempre que haya un evento independiente de los jugadores y cuyo resultado es aleatorio
- Los sucesores de un nodo “azar” son las posibles situaciones que podrían ser el resultado de este elemento de azar
 - p.e.: todos los posibles resultados de tirar un dado
 - Cada uno de los sucesores de un nodo “azar” tiene asociado la probabilidad de que este resultado ocurra
 - p.e.: en el caso del dado: $p(1) = 1/6, \dots, p(6) = 1/6$

EJEMPLO: BACKGAMMON SIMPLIFICADO

Estado inicial

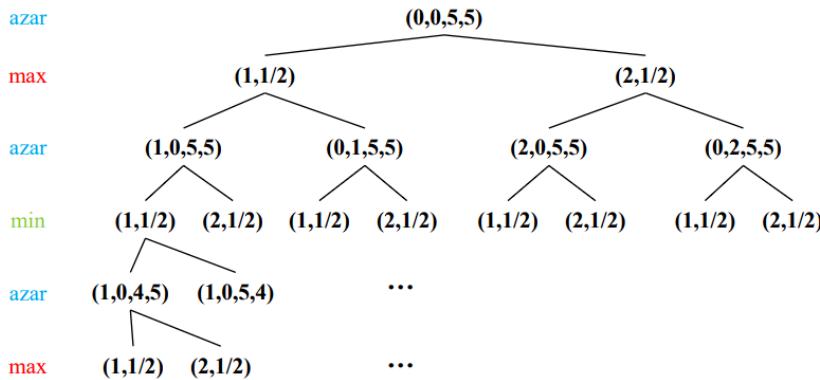


Objetivo:

- mover las fichas al lado opuesto (max= ○, al campo 5 y min= ● al campo 0)
- Reglas:
 - max empieza y los jugadores se van alternando sus jugadas
 - Cada jugada consiste primero en tirar una moneda; la cara tiene el valor 1 y la cruz el valor 2. Después se mueve una de las fichas 1 o 2 campos en la dirección deseada (dependiendo del resultado de la tirada de la moneda)
 - No es posible mover una ficha a un campo que tiene una ficha del oponente
 - Si un jugador no puede mover sus fichas pierde su turno (si puede, tiene que mover una ficha)
 - Gana el jugador que primero ha movido ambas fichas al campo deseado

- El elemento de azar ocurre antes de elegir la jugada

S: (x_1, x_2, y_1, y_2) , $x_1 = \text{blanca } 1, x_2 = \text{blanca } 2, y_1 = \text{negra } 1, y_2 = \text{negra } 2$
 azar: $(v, p), v = \{1,2\}, p = 1/2$



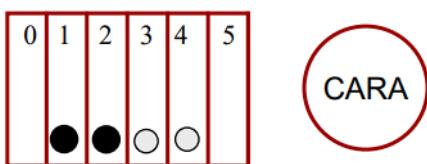
EXPECTMINIMAX

- Objetivo: elegir la mejor jugada para *max*
- ¿Cómo propagar los valores de utilidad/evaluación de los nodos hoja a los nodos superiores?
- Solución:

$$\text{ExpectMinimax}(n) = \begin{cases} e(n) \Leftrightarrow n \text{ suspension?} = \text{true} \\ \max_{s \in \text{expandir}(n)} \text{ExpectMinimax}(s) \Leftrightarrow l_s = \text{max} \\ \min_{s \in \text{expandir}(n)} \text{ExpectMinimax}(s) \Leftrightarrow l_s = \text{min} \\ \sum_{s \in \text{expandir}(n)} p(s) * \text{ExpectMinimax}(s) \Leftrightarrow l_s = \text{azar} \end{cases}$$

EJEMPLO: BACKGAMMON SIMPLIFICADO II

Situación actual: (toca a *max*)



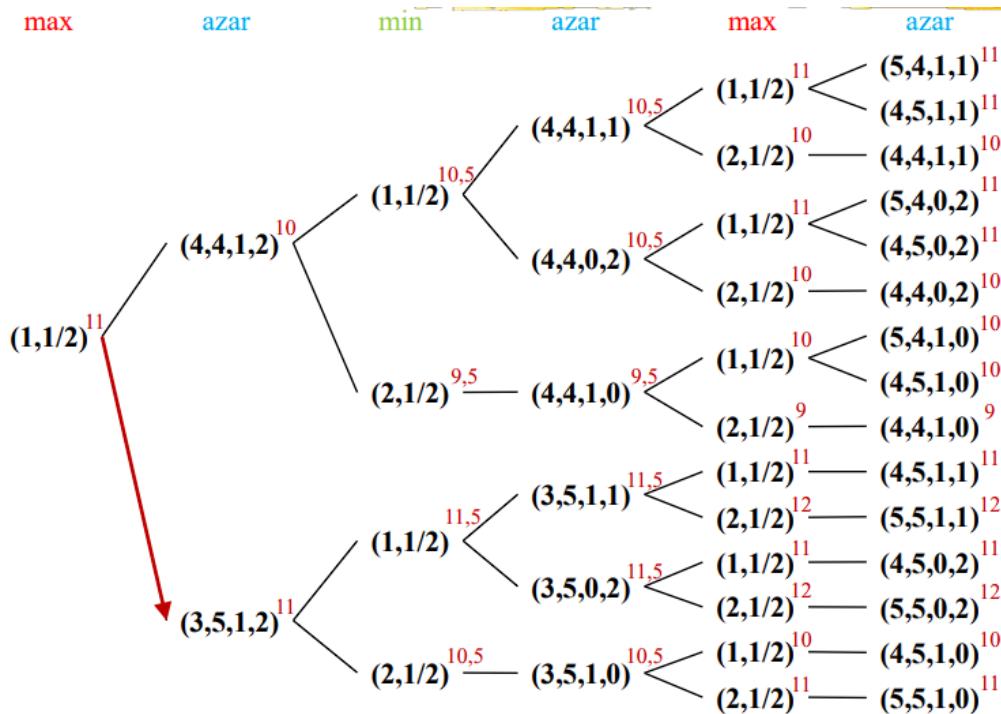
$(3,4,1,2)$; *max* tiene que mover una ficha (blanca) una posición

- Suponemos el algoritmo ExpectMinimax con un nivel de suspensión de 5
- Como función de evaluación se usa la siguiente: $e((a,b,c,d)) = a+b+c+d$

Valores altos de *a* y *b* son buenos para *max* porque indican que sus fichas están cerca de la meta (5)

Valores altos de c y d son buenos para **max** porque indican que las fichas de min están lejos de su meta (0)

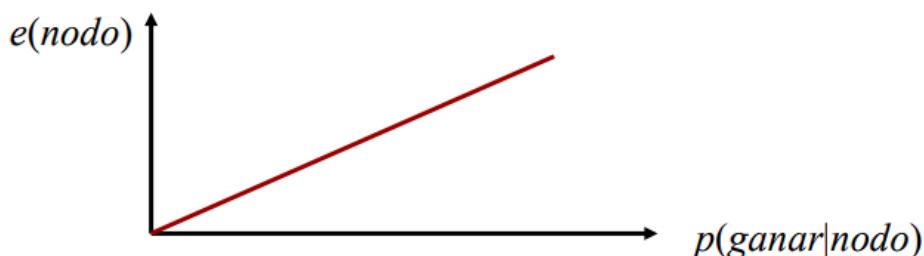
Para el estado actual: $e((3,4,1,2)) = 10$



FUNCIONES DE EVALUACIÓN

La función de evaluación e debe estimar la probabilidad de ganar

Caso ideal: e es una transformación lineal positiva de dicha probabilidad



Habitualmente, no es factible establecer una función e que cumple este criterio para todos los estados del juego (nodos)

Normalmente, una función e es más “exacto”, cuanto más cerca está el estado actual del juego de un estado terminal

Es preferible aplicar la búsqueda (Expect)Minimax en el máximo número de plies posibles, y sólo al final aplicar la función de evaluación e

COMPLEJIDAD EXPECTMINIMAX

Complejidad: proporcional al número de nodos en el árbol

Nivel de suspensión d (plys), factor de ramificación b (jugadas posibles), elementos de azar con n posibilidades

ExpectMinimax (incluyendo nodos azar): $O(b^d \cdot n^d)$

Ejemplo Backgammon: n=21 (2 dados) y b≈20

Número de <i>plys d</i> anticipados	Número de nodos en el árbol
1	$20*21=420$
2	176.400
3	74.088.000

Aplicación de la poda α - β al algoritmo ExpectMinimax:

Véase Russell/Norvig 3^a ed., sección 5.5.1

HOJA DE PROBLEMAS II

Ejercicio 1: Preguntas de teoría

1. ¿Cuáles de las siguientes afirmaciones acerca del algoritmo A* son verdaderas y cuáles son falsas?

- a) Si h^* es optimista, entonces el valor f^* crece de forma monótona en todos los caminos del árbol de búsqueda.
- b) Si h^* es optimista, entonces el algoritmo A* es óptimo.
- c) Si h^* es optimista, entonces $h^*(n) > g(n)$ en todos los nodos n del árbol de búsqueda.
- d) Si h^* no es optimista, entonces el algoritmo A* no es completo.

2. Para cada uno de los siguientes dominios, marque si es verdadero o falso el que las funciones heurísticas indicadas sean optimistas:

- a) En el 8-puzzle: la suma de los números de todas las piezas descolocadas en el 8-puzzle.
- b) En el problema de la col, la oveja y el lobo: el número de pasajeros que se encuentran en el lado incorrecto del río.
- c) En un laberinto discreto: la distancia de Manhattan entre las coordenadas del estado actual (posición del agente) y las coordenadas de la salida.
- d) En el problema de encontrar rutas: la raíz cuadrada de la distancia aérea entre la ciudad actual y la ciudad meta (la distancia es un número entero positivo).

3. Si se aplica el algoritmo A* con una función heurística h_1^* , tal que $h_1^*(n) = 0$ para todos los nodos n, entonces... (selecciona todas las correctas)

- a) ...la función h_1^* es optimista.
- b) ...todas las funciones heurísticas optimistas h_2^* son más informadas que h_1^* .
- c) ...el algoritmo A* expande los mismos nodos que la búsqueda de coste uniforme.
- d) ...no existe ninguna función heurística optimista h_2^* con la que el algoritmo A* expande menos nodos que con h_1^* .

4. Suponga que para un problema a resolver con el algoritmo A* se dispone de varias funciones heurísticas optimistas h_1^*, \dots, h_n^* , todas ellas de fácil evaluación. ¿Cuáles de las siguientes afirmaciones son correctas y cuáles son falsas?

- a) Si hay una función h_k^* que es más informada que otra h_i^* , entonces se puede prescindir de h_i^* en el proceso de búsqueda.
- b) Si para un nodo n se cumple que $h_i^*(n) < h_k^*(n)$, entonces se puede prescindir de h_i^* en todo el proceso de búsqueda.
- c) Para cada nodo n es conveniente elegir la función h_i^* de máximo valor.
- d) Si la función h_k^* es más informada que h_i^* , entonces $A^*(h_k^*)$ encuentra una mejor solución que $A^*(h_i^*)$.

Ejercicio 2: evaluación de heurísticas

El grafo que se muestra en la figura 1 muestra una red 2D donde los nodos representan estados y los arcos acciones y sus resultados. El estado inicial es $(0, 0)$, y el estado meta (x, y) , siendo x e y números enteros. Para un nodo genérico (u, v) , ¿cuáles de las siguientes funciones heurísticas son optimistas? Razoné brevemente su respuesta.

1. $h^*((u, v)) = |u - x| + |v - y|$
2. $h^*((u, v)) = |u - x| * |v - y|$
3. $h^*((u, v)) = 2 * \min(|u - x|, |v - y|)$
4. $h^*((u, v)) = |u| + |x| + |v| + |y|$
5. $h^*((u, v)) = \sqrt{|u - x|^2 + |v - y|^2}$

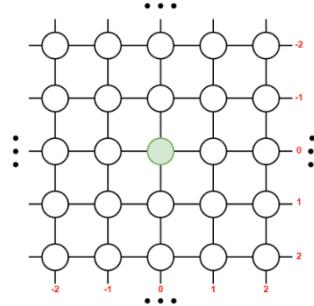
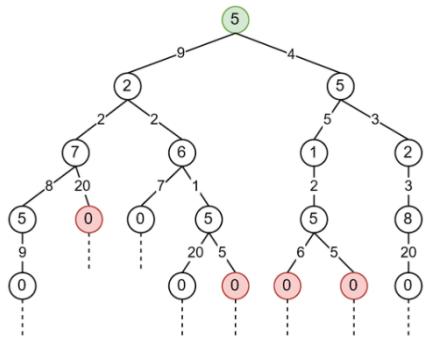


Figura 1

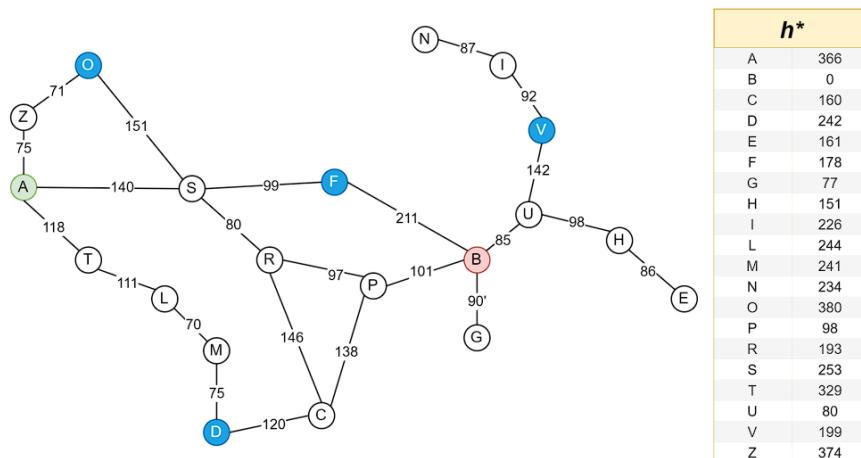
Ejercicio 3: búsqueda A*



Considera el siguiente subárbol de búsqueda mostrado en la figura 2. Los números asignados a cada arco representan los costes de las operaciones/acciones correspondientes. Los números en los nodos representan una estimación del coste del camino más corto de este nodo a un nodo meta. Los nodos meta están marcados en color rojo. Construye el árbol que expandiría el algoritmo A* aplicado a este problema, indica el orden en el que se expandirían los nodos, los valores de la función f^* y el nodo meta que el algoritmo encontraría.

Ejercicio 4: búsqueda A*

El grafo que se muestra en la figura 3 representa un esquema de un mapa de carreteras. Los nodos están etiquetados con el nombre de ciudades, y los arcos con la distancia por carretera entre dichas ciudades. La función h^* que se muestra al lado indica la distancia aérea entre cualquier ciudad y la ciudad de B. Suponga que nuestro agente dispone de un coche eléctrico que le permite recorrer 320km sin recargar. Sólo puede recargar en ciudades con estación de recarga pública, los cuales se indican en el mapa por los cuadrados rellenos (ciudades D, F, O, y V). Inicialmente el agente se encuentra en la ciudad A con la batería completamente cargada, y desea trasladarse a B por la ruta más corta posible (e.d. sin quedarse tirado por el camino con la batería vacía). Conoce la red de carreteras, la posición de las estaciones de recarga, y las características de su vehículo, por lo que en cada momento sabe a qué ciudades vecinas puede ir dado el estado de carga de su batería. El agente decide aplicar la búsqueda A* para resolver el problema.



1. Represente el problema como espacio de estados, definiendo el conjunto de estados S , el estado inicial s_0 , los estados meta (a través de la función $meta?$), el coste $c(s_i, s_j)$ para ir de una ciudad s_i a la ciudad vecina s_j , así como la función $expandir$ (para dichas definiciones, puede suponer que existe un predicado binario *adyacente* que indica que dos ciudades son vecinas, un predicado unario *carga* que indica si hay una estación de recarga en una ciudad, y una función binaria *d* que proporciona la distancia entre dos ciudades vecinas de acuerdo con el mapa).
2. Desarrolle el árbol de búsqueda que genera el algoritmo A* (no se filtran ciclos de ningún tipo). Indique el orden en el que se expanden los nodos, los valores de g , h^* , y f^* de cada nodo del árbol de búsqueda, así como el estado de la lista abierta en cada paso del algoritmo.
3. ¿Para la representación del apartado (a), la función heurística h^* (distancia área entre ciudades) sería optimista? Justifique brevemente su opinión.

Ejercicio 5: algoritmo A* (8-puzzle)

1	2	3
6	4	
8	7	5

estado inicial

1	2	3
8		4
7	6	5

estado meta

Considera el 8-puzzle cuyo estado inicial y estado meta se muestran en la figura 4. Desarrolla el árbol de búsqueda que expande el algoritmo A* utilizando las siguientes heurísticas. Evita ciclos generales, indique el orden de expansión de los estados y muestre en cada paso los valores de h^* , g , y f^* . Cuando puedas elegir entre varios nodos para ser expandidos, asume el “peor caso”.

1. $h^*(n)$ = número de piezas descolocados en n respecto al estado meta
2. $h^*(n)$ = suma de las distancias Manhattan de las piezas en n respecto al estado meta

Ejercicio 6: algoritmo A* (cuadrados latinos)

En el juego de los “cuadrados latinos” se parte de un tablero 3 X 3 vacío. En cada posición colocamos números del 1 al 9, ninguno de los cuales puede repetirse. El objetivo es tener el tablero completo, es decir, un número en cada posición de este, y es necesario que el valor de la suma de las filas, columnas y diagonales sea siempre el mismo valor: 15. En la figura 5 se puede ver un ejemplo en el cual tenemos el tablero completo, se han utilizado todos los números, pero no se consigue el objetivo. En este ejemplo sólo una diagonal y la última fila cumplen que la suma de sus números es 15.

1	9	2
7	5	6
8	4	3

1. Define una representación eficiente para el juego de los cuadrados latinos 3x3, especificando el conjunto de estados, el estado inicial, y las operaciones permitidas en cada estado.
2. Considera el estado inicial que se muestra al lado donde tenemos seis posiciones ocupadas, tres libres, y sólo una diagonal cumple que la suma de sus números es 15. Cada paso tiene coste uno. Asimismo, considere la siguiente función heurística definida para cualquier estado n del tablero:

$$h^*(n) = \text{el número de filas} + \text{el número de columnas} + \text{el número de diagonales}$$
 en el estado n que no cumplen que la suma de sus números es 15.

2		4
	5	3
6	1	

Por ejemplo, el valor de h^* de la configuración de la primera figura es 6, mientras que el valor de h^* para la configuración de la segunda figura es 7. Desarrolla el árbol de búsqueda que genera el algoritmo A* para este problema. Indique el orden en el que se expanden los nodos, los valores de g , h^* y f^* para cada nodo del árbol de búsqueda, y la evolución de la lista abierta.

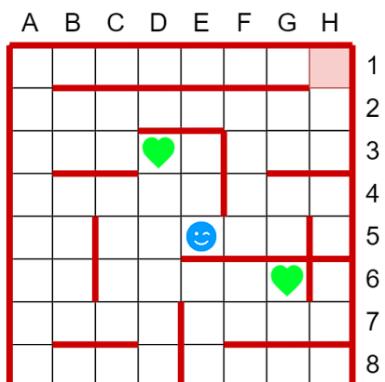
3. ¿La función h^* es optimista? ¿El algoritmo A* encuentra siempre la solución de menor coste? Razone brevemente sus respuestas.

HOJA DE PROBLEMAS III**Ejercicio 1: búsqueda por subobjetivos**

Dado un típico problema de buscar una ruta de un punto a otro en una red de carreteras, por ejemplo, encontrar una ruta para ir de Madrid a París. Describe cómo se podría emplear la búsqueda por subobjetivos a este problema.

Ejercicio 2: búsqueda por subobjetivos

Considera el juego representado en la figura 1. El avatar (la carita) tiene que moverse por el tablero hasta la salida (en la posición H1). Los movimientos se realizan de forma horizontal o vertical y siempre de un cuadrado a uno adyacente (si no hay ningún obstáculo). En el camino debe recoger todos los objetos corazones disponibles. Emplea el algoritmo de búsqueda por subobjetivos a este problema. ¿Qué subobjetivos fijarías? ¿Qué método de búsqueda base utilizarías? Aplica el algoritmo con los parámetros tu elección.



Ejercicio 3: búsqueda por subobjetivos

Considera el plano de la figura 2, correspondiente a una sala con obstáculos:

Un agente (por ejemplo un robot) tiene el objetivo de llegar desde la posición en la que se le posiciona en la sala hasta la puerta (en el cuadrado H1). Para ello el agente se puede mover de un cuadrado (identificado por sus coordenadas) a uno adyacente (no por la diagonal) siempre y cuando no haya ningún obstáculo en medio. Además, algunos campos tienen un “gasto de transito” (el número que figura en ellos).

Se supone que el agente conoce perfectamente la sala y sabe dónde están todos los obstáculos y los costes adicionales.

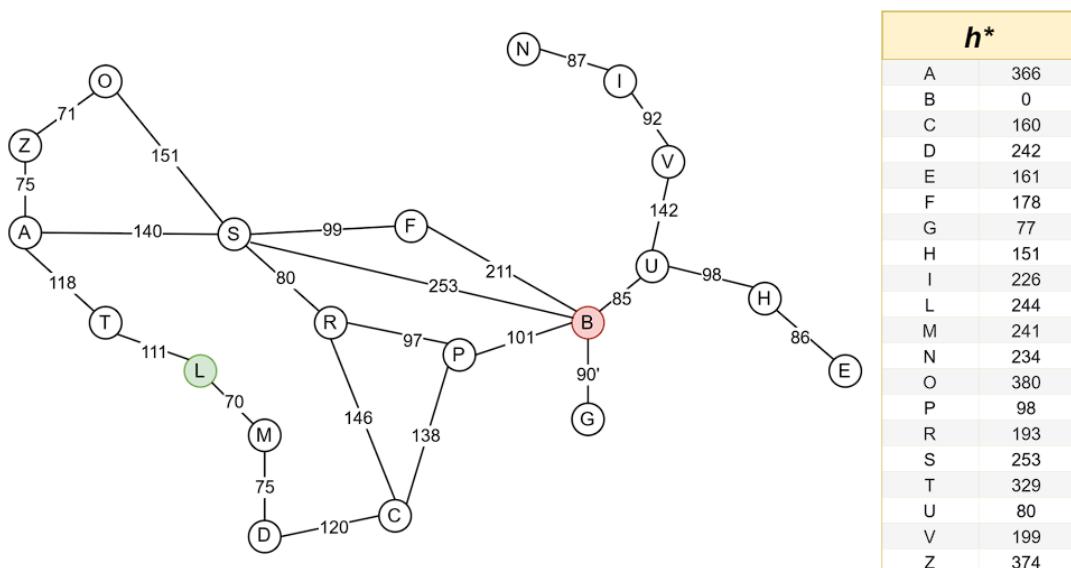
El agente quiere llegar lo más rápido a la salida, pero a la vez quiere pagar lo menos posible de tránsito. Concretamente, según las preferencias del agente, cada paso extra es igual de negativo que una unidad de “coste de tránsito”.

Supón que el agente se encuentra en la posición F2. La pregunta es ¿Qué camino debe utilizar el agente para salir?

1. Define los estados de este problema formalmente. En base a la definición de los estados, define formalmente una función heurística h^* para este problema. ¿Es tu función h^* optimista? ¡Justifique brevemente su propuesta!
2. Realiza una búsqueda por ascenso de colina en este problema. Indica la secuencia de pasos que realiza el agente e indica en cada paso el movimiento que el agente elige entre las posibles alternativas. Nota: Supón que el agente recuerda en cada momento su última acción y, de esta forma, evita ciclos simples.
3. Analiza los resultados del apartado anterior ¿Cómo se podría mejorar la solución? Realiza de nuevo la búsqueda con tu propuesta.

Ejercicio 4: búsqueda con horizonte

El grafo que se muestra en la figura 3 representa un esquema de un mapa de carreteras. Los nodos están etiquetados con el nombre de ciudades, y los arcos con la distancia por carretera entre dichas ciudades. Inicialmente nuestro agente se encuentra en la ciudad L, y procura trasladarse por carretera a la ciudad B por el camino más corto. Con tal fin, puede hacer uso de su conocimiento de la región, y en particular de la distancia aérea entre ciudades. La función h^* que se muestra al lado indica la distancia aérea entre cualquier ciudad y la ciudad de destino B.

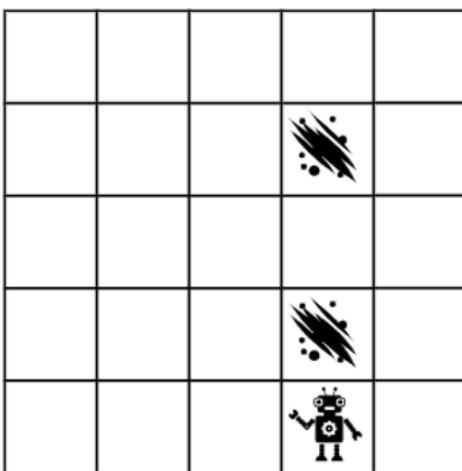


Supón que el agente, para encontrar el camino, utiliza una búsqueda con horizonte con un horizonte=3. Aplica la búsqueda con horizonte a este problema, desarrollando el(es) árbol(es) de búsqueda y indicando toda la información relevante para entender el desarrollo realizado. Describe brevemente tu aplicación de la búsqueda con horizonte en este problema.

Ejercicio 5: búsqueda con horizonte

La empresa iRobot quiere desarrollar un robot autónomo para limpiar hogares. Para esto, hay que desarrollar su proceso de toma de decisión para que, una vez encendido, pueda limpiar de manera autónoma la habitación donde se encuentra. Se supone que el robot viene con una micro-camara, que se instala en el techo de cada habitación, que proporciona una imagen de la habitación, como en figura 4.

El robot tiene 4 acciones posibles, Arriba, Abajo, Izquierda, Derecha. Aplica el algoritmo de búsqueda con horizonte, a partir del estado inicial de figura 3. Utiliza un **horizonte** $k = 2$, una **heurística** $h^*(s) = \text{número de manchas en el estado } s$, y la **búsqueda en profundidad limitada** como método base, filtrando todos los estados repetidos.

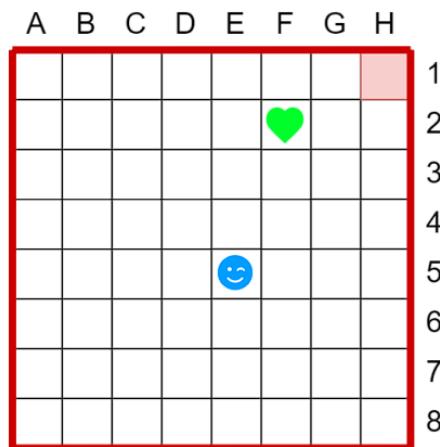


Ejercicio 6: búsqueda online

Considera el juego presentado en la figura 5 Un avatar (carita) tiene que pillar el corazón que se mueve en el tablero. En cada jugada ocurre lo siguiente: 1) el avatar realiza su movimiento, 2) el corazón puede moverse.

En los movimientos del jugador, éste puede decidir si realiza un movimiento de una o de dos casillas. Por otro lado, el corazón siempre realiza movimientos de 1 casilla (de forma aleatoria). El juego termina si el avatar ha pillado el corazón.

¿Qué algoritmo puedes emplear para decidir las jugadas del avatar? ¿Cómo aplicarías este algoritmo (qué parámetros utilizarías)? Aplica el algoritmo elegido a una posible instancia del problema (invéntate las posiciones de los objetos, así como los movimientos del corazón).



Ejercicio 7: problemas de optimización

En juego de los “cuadrados latinos” se parte de un tablero 3×3 vacío. En cada posición vamos colocamos números del 1 al 9, ninguno de los cuales puede repetirse. El objetivo es tener el tablero completo, es decir, un número en cada posición del mismo, y es necesario que el valor de la suma de las filas, columnas y diagonales sea siempre el mismo valor: 15.

En la figura 6 puede verse un ejemplo en el cual tenemos el tablero completo, se han utilizado todos los números, pero no se consigue el objetivo indicado. En este ejemplo sólo una diagonal y la última fila cumplen que la suma de sus números es 15. Este problema se puede resolver con un algoritmo A* en el cual se añade en cada paso una nueva cifra a un tablero inicialmente vacío.

Ahora, considera el siguiente juego relacionado con los “cuadrados latinos”: Varios jugadores tienen cada uno un tablero con las cifras ya puestas (en cualquier orden). Los jugadores tienen un tiempo limitado para intentar conseguir un tablero “ordenado”. Para ello, en cada paso, pueden intercambiar 2 cifras del tablero. Gana aquel jugador que, primero consigue “ordenar” el tablero, o, si ninguno lo consigue, él que obtiene un tablero con el mayor número de líneas (horizontales / verticales / diagonales) que sumen 15, cuando acabe el tiempo. Los jugadores no saben cuánto tiempo tienen disponible.

1. Si tuvieras que implementar las actuaciones de uno de los jugadores, explique cómo lo harías (¿Qué métodos y técnicas y parámetros utilizarías?). Argumenta las razones.
2. Aplica lo que has propuesto en el apartado anterior. Considera que el tablero inicial es el que se ha especificado en la figura 6.

1	9	2
7	5	6
8	4	3

HOJA DE PROBLEMAS IV**Ejercicio 1:**

Preguntas de teoría 1. ¿A cuáles de los siguientes juegos puede aplicarse la búsqueda Minimax?

- a. Parchís
- b. Juego de las damas**
- c. Pacman
- d. Tres en raya**

2. ¿Cuáles de las siguientes afirmaciones acerca del algoritmo Minimax son ciertas?

- a. El algoritmo Minimax realiza una exploración primero en anchura del árbol de juego. **FALSA**
- b. El algoritmo Minimax maximiza la utilidad mínima que puede conseguir el jugador max. **VERDADERA**
- c. Puede haber estrategias que funcionan mejor que Minimax Si el contrincante no es óptimo. **VERDADERA** hijos no son óptimos
- d. Puede haber estrategias que funcionan mejor que Minimax si el contrincante es óptimo. **FALSO** Si el contrincante es óptimo, Minimax es el mejor.
- e. Con la poda $\alpha - \beta$ se eliminan nodos que nunca serán alcanzados. **VERDADERA**

3. Con la poda $\alpha - \beta$ en el algoritmo Minimax...

- a. ...en algunos casos puede empeorar la calidad de la solución (i.e. es posible que se elija una jugada peor que el Minimax simple). **FALSA**
- b. ...puede mejorar la velocidad con la que se encuentra una solución (i.e. se elige una jugada más rápidamente que el Minimax simple). **VERDADERA**
- c. ...suele mejorar la calidad de la solución (i.e. se elige una mejor jugada que el Minimax simple). **FALSA**

d. ...produce siempre la misma solución (i.e. se elige la misma jugada que en el Minimax simple).

VERDADERA

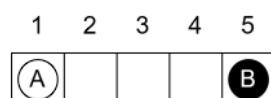
4. ¿Cuáles de las siguientes afirmaciones acerca del algoritmo ExpectMinimax son ciertas?

- a. Los nodos al azar representan una mala jugada del jugador min. **FALSO**
- b. Los nodos al azar representan una buena jugada del jugador min. **FALSO**
- c. Los nodos al azar representan un evento aleatorio. **VERDADERO**
- d. Si un nodo azar tiene k sucesores equiprobables entonces, al evaluar el árbol de juego, se pueden sumar las evaluaciones de los sucesores del nodo azar y dividir dicha suma por k.

VERDADERO

Ejercicio 2:

Algoritmo Minimax Considera el siguiente juego: Dos contrincantes (min y max) manejan una ficha en un tablero como en la figura 1. min y max mueven respectivamente las fichas B y A. Los dos jugadores mueven por turno, empezando por max.



Cada jugador debe mover su ficha a un espacio vacío adyacente en una u otra dirección. Si el adversario ocupa un espacio adyacente, entonces un jugador puede saltar sobre el adversario al siguiente espacio vacío, si existe. Por ejemplo, si A está sobre 3 y B está sobre 2, entonces max puede mover A hacia atrás al espacio 1.

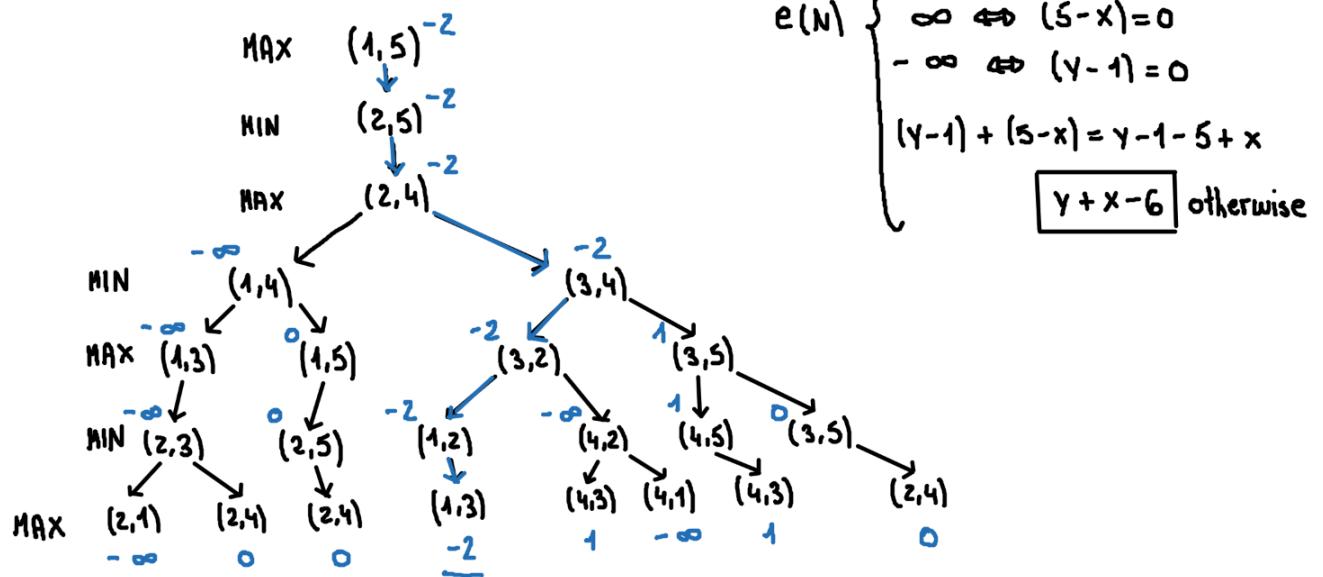
El juego termina cuando un jugador alcanza el extremo opuesto del tablero.

1. Define una función de evaluación $e(n)$, $\forall n \in N$.

2. Aplica el algoritmo **Minimax** con suspensión hasta el nivel 6, empezando con la situación de la figura 1. Especifica los valores calculados para cada nodo, como se propagan por el árbol de juego y determina la mejor jugada para max.

1. Función de evaluación $N(x,y)$ $x,y \in \{1,2,3,4,5\}$

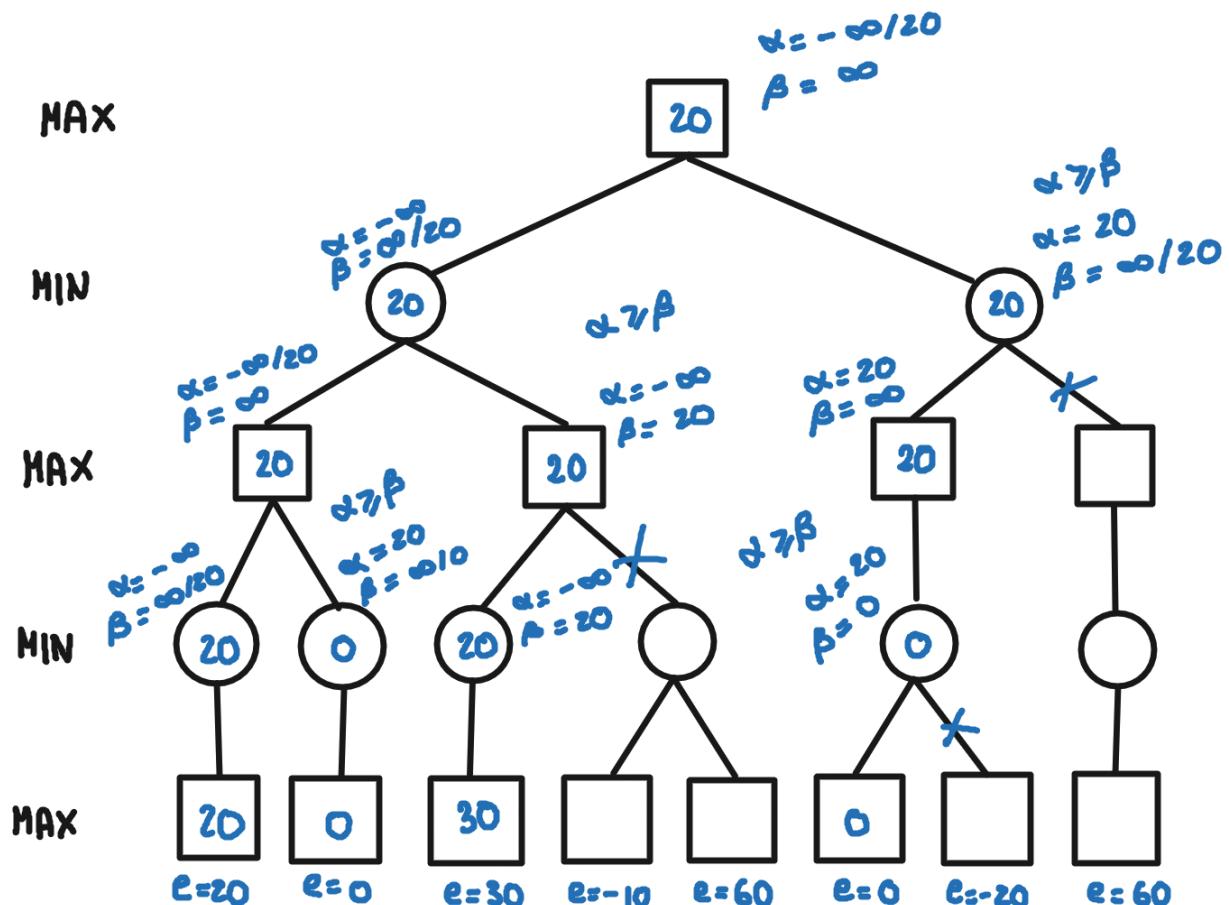
2.



Ejercicio 3:

Algoritmo Minimax con poda $\alpha - \beta$ Considera el árbol de la figura 2 de un juego de dos personas.

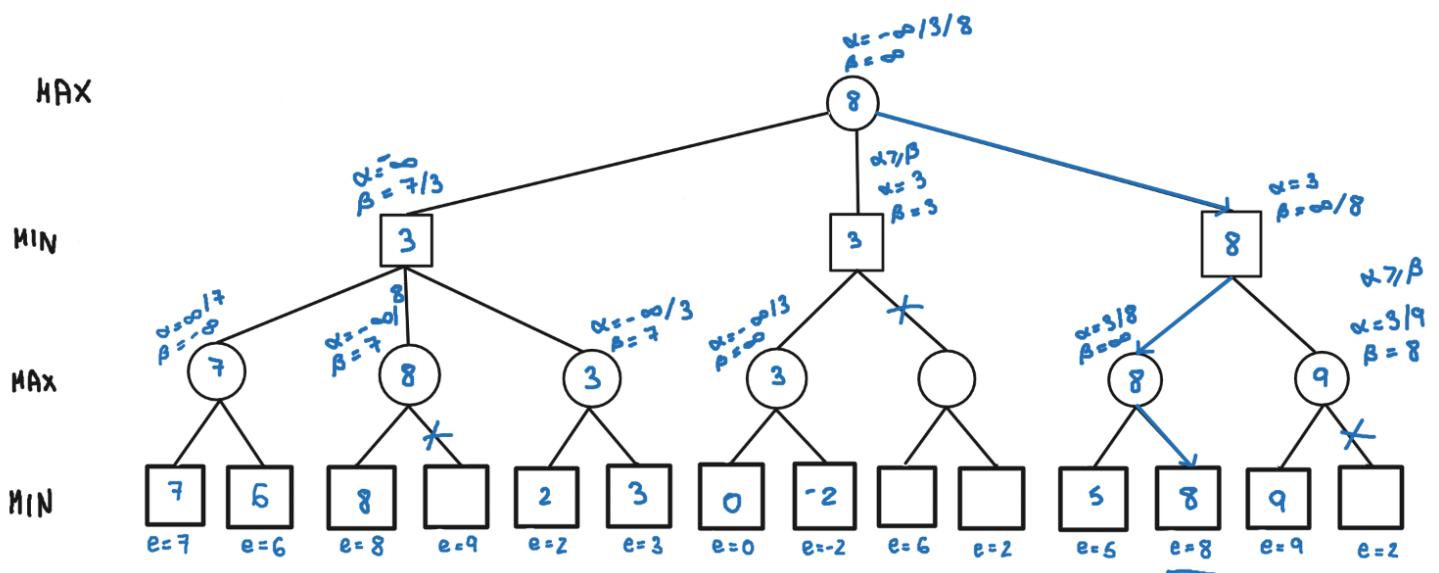
- Aplica el algoritmo **Minimax** con poda $\alpha - \beta$, propaga los valores de evaluación hasta el nodo raíz, marca la mejor jugada para max y marca todos los subárboles que se podan. Especifica en cada nodo los valores de α y β que propaga el algoritmo.



Ejercicio 4:Algoritmo Minimax con poda $\alpha - \beta$

Considera el arbol en la figura 3.

- Aplica el algoritmo **Minimax** con poda $\alpha - \beta$, propaga los valores de evaluación hasta el nodo raíz, marca la mejor jugada para max y marca todos los subárboles que se podan. Especifica en cada nodo los valores de α y β que propaga el algoritmo.



Ejercicio 5:

Algoritmo Minimax Dos conductores, el agente max y su contrario min, se plantean competir sobre un circuito entre diferentes ciudades, reflejado en el mapa en la figura 4 con las siguientes reglas:

El recorrido del circuito se hace por tramos, partiendo de la ciudad marcada como Salida.

Los jugadores, alternativamente eligen el tramo a recorrer entre aquellos que parten de la ciudad en la que se encuentran. Una vez elegido el tramo, ambos conductores lo recorren y el jugador que eligió el tramo se apunta los kilómetros de este. Por ejemplo, si están en la ciudad Salida, y el agente max elige ir a B, el agente max se apunta los 80 Kms del viaje. A continuación, el agente min debe elegir, partiendo de la ciudad B.

Un agente no puede ir de una ciudad a otra en la que haya estado antes, por lo que la competición acaba cuando el conductor al que le toca moverse no puede ir a ninguna ciudad no visitada anteriormente.

Gana el conductor que haya acumulado más kilómetros con los tramos recorridos.

1. Define una representación eficiente para el juego, incluyendo la función de evaluación adecuada para los nodos hoja.
2. Desarrolla el árbol de juego **Minimax** hasta ply = 4. Genera los sucesores de un nodo en orden alfabético, por ejemplo, el primer nodo sucesor de salida sería A y el segundo B. Propaga los valores de evaluación a través del árbol y marca la jugada que haría el agente max.
3. ¿Qué partes de árbol de juego no se expandirían si se aplicara poda $\alpha - \beta$?

$$C = \{A, B, C, D, E, F, G, S\}$$

$$R(\text{agente}) = (C_1, C_2, \dots, C_n)$$

$$K(a_1, a_2) = R_{a_1} - R_{a_2}$$

$$N = (C, R, K) \quad e(C, R, K) = \begin{cases} \infty \Leftrightarrow \text{expand } (c) \in R \wedge K > 0 \\ -\infty \Leftrightarrow \text{expand } (c) \in R \wedge K < 0 \\ K \Leftrightarrow \text{otherwise} \end{cases}$$

