Inteligencia artificial.

**TEMA 1 Introducción:**

¿Qué es la IA?: la comprensión científica de los mecanismos sobre los que se basa el pensamiento y comportamiento inteligente, y su implementación en máquinas.

Comportamiento inteligente: Percepción, Razonamiento, Aprendizaje, Comunicación e Interacción con un entorno complejo (incluyendo otros agentes).

IA engloba: Representación del conocimiento, Razonamiento automático, Aprendizaje automático, Procesamiento de lenguaje natural, Percepción, Visión artificial y Robótica

Tabla

Descripción generada automáticamente

Definición operacional de inteligencia: Un sistema es inteligente si puede persuadir a otro sistema inteligente (por ejemplo, un humano) de su inteligencia.

Test de Turing:

Ventajas: Evita entrar en debates posiblemente estériles sobre ”¿qué es la inteligencia?” o “¿pueden las máquinas pensar?”, El test se sigue considerando una buena forma de medir el nivel de éxito que tienen los sistemas basados en IA en exhibir comportamiento inteligente Dificultades: Ambiguo, no reproducible, No constructivo, No se puede describir de manera matemática, Los esfuerzos actuales de la IA no se dirigen a pasar el test de Turing.

Ciencia cognitiva (psicología experimental + ciencias de la computación): Modelos de los procesos de pensamiento humanos.

Ventajas: Proporciona comprensión de inteligencia + cognición humana. Los modelos se diseñan para imitar el funcionamiento de la mente humana y son por ello inteligibles Dificultades: El hecho de que un dispositivo basado en IA actúe como un humano en una tarea que requiere inteligencia no significa que el dispositivo sea un modelo apropiado del proceso de pensamiento humano correspondiente. El mejor diseño artificial para un sistema inteligente no es necesariamente una imitación de la mente humana.

Lógica formal (matemáticas + ciencias de la computación): Proceso de razonamiento automático donde se deducen conclusiones correctas a partir de premisas correctas [representación + premisas + reglas de derivación]

Ventajas: Notación precisa y no ambigua de las afirmaciones sobre objetos del mundo y las relaciones entre ellos. Un algoritmo de resolución + búsqueda completa puede, en principio, resolver cualquier problema que pueda ser formulado en notación lógica. Si el problema no tiene solución, el algoritmo podría no parar nunca.

Dificultades: Formalización de conocimiento informal. Los requerimientos computacionales (espacio/tiempo) pueden hacer que los cálculos sean imposibles de hacerse en la práctica.

Agente racional

Sistema autónomo, capaz de percibir e interactuar con su entorno, de explorar (recopilación de información), aprender y adaptarse, de formular objetivos y diseñar planes para alcanzar esos objetivos. El agente es racional, en el sentido de que actúa para alcanzar el mejor resultado, o mejor resultado esperado, de acuerdo con una medida de rendimiento, condicionada a su conocimiento del mundo y dados los recursos computacionales existentes (limitados). Ventajas: Integra y extiende definiciones previas. La racionalidad limitada es más realista que la racionalidad absoluta.

Dificultades: Las exploraremos a lo largo de este curso.

**TEMA 2.1 búsqueda no informada:**

Búsqueda en el espacio de estados:

Solucionar un problema mediante búsqueda:

Formulación + búsqueda + ejecución

Formulación del problema:

Definir los estados de búsqueda (Deben incluir solo la información relevante para la solución del problema. No tienen por qué coincidir con los estados del sistema. Por ejemplo, en problemas con información incompleta, pueden corresponder a conjuntos de estados posibles).

Especificar el estado inicial.

Especificar las acciones que puede realizar el agente:

– Reglas para las acciones permitidas.

– Función sucesor:

Estado actual → Lista de estados directamente accesibles.

Definir los estados objetivo

– Definición extensiva: Lista

– Definición intensiva: Test de objetivo

Definir utilidad: Función de coste del camino.

**Camino**: Secuencia de estados conectados por acciones.

**Espacio de estados**: Conjunto de estados accesibles desde el estado inicial Se representa mediante un grafo conexo cuyos nodos ≡ estados, arcos ≡ acciones.

**Solución**: Camino desde el estado inicial al estado(s) objetivo(s). ´

**Solución** óptima: Solución con el mínimo coste.

Problemas sencillos I:

**Estados**: Disposiciones de 8 casillas numeradas de 1 a 8 + casilla vacía en un tablero de 3 x 3. **Estado inicial**: Una disposición dada (arbitraria).

**Acciones**: Desplazar una pieza adyacente a la casilla vacía, a esa casilla. El hueco estará ahora en donde estaba la pieza que se ha movido.

**Estado objetivo**: Una disposición ordenada, con la casilla vacía en el medio.

**Utilidad**: Coste unidad por movimiento

Problemas sencillos II:

Problema de las N reinas (formulación con estados completos)

**Estados**: Disposiciones de N reinas en un tablero de ajedrez de N x N casillas.

**Estado inicial**: Una disposición dada (arbitraria) de las N reinas en el tablero.

**Acciones**: Mover una reina a una casilla vacía.

**Estado final**: N reinas que no se atacan entre ellas.

**Utilidad**: Sólo es importante el estado final.

Problemas sencillos III:

Problema de las N reinas (formulación incremental)

**Estados**: Disposiciones de N reinas en un tablero de ajedrez de tamaño N x N.

**Estado inicial**: Un tablero vacío.

**Acciones**: A partir de un estado con n reinas, situar una reina nueva en una casilla vacía en la columna vacía más a la izquierda, que no ataque a las reinas previamente situadas.

**Estado final**: N reinas que no se ataquen entre sí.

**Utilidad**: Todos los caminos tienen igual longitud (N pasos) e igual coste.

Problemas sencillos IV:

Criptoaritmética.

**Estados**: Operaciones aritmética entre números codificados con letras y dígitos.

**Estado** **inicial**: Los números están codificados únicamente con letras

**Acciones**: Sustituir una letra por un dígito no usado.

**Estado final**: Todas las letras se sustituyen por dígitos de tal forma que el resultado de la operación aritmética es correcto.

**Utilidad**: Todas las acciones tienen coste unidad

Problemas sencillos V:

Debemos transportar 3 excursionistas incautos (Laura, Íker y Manel) de un lado a otro de un río utilizando una barca.

Inicialmente, junto a los excursionistas hay 3 zombis (zombi 1, zombi 2 y zombi 3) en la misma orilla.

La barca solo tiene capacidad para 2 criaturas además del barquero.

En ningún momento puede haber ni en la barca ni en cualquiera de las orillas más zombis que excursionistas

Problemas reales:

Buscador de rutas (por ejemplo para un viaje en avión)

**Estados:** Aeropuerto + hora actual.

**Estado inicial:** Aeropuerto de salida + hora de partida

**Acciones:** Estados resultantes de tomar uno de los vuelos programados disponibles desde el aeropuerto actual, de forma que la salida de dicho vuelo sea posterior a la hora actual + tiempo de tránsito del aeropuerto.

**Estado final:** Aeropuerto destino + franja horaria de llegada deseada.

**Utilidad:** Coste económico, tiempo total de viaje, número de escalas, comodidad, etc.

Problemas de planificación de rutas:

Problema del viajante de comercio: Encontrar un recorrido en el que se visiten N ciudades, de forma que cada ciudad se visite sólo una vez.

– **Estados**: Secuencia de ciudades visitadas.

– Estado inicial: Secuencia con un único elemento, el cual corresponde a la ciudad de partida.

– **Acciones**: Desplazarse desde la ciudad actual (la última en la secuencia de ciudades visitadas) a una ciudad aún no visitada.

– **Test objetivo**: La secuencia de ciudades visitadas que caracteriza el estado tiene longitud N. Esta condición basta para definir el circuito.

– **Utilidad**: Longitud del circuito formado por la secuencia de ciudades correspondiente al estado final, seguida de la ciudad de partida.

La solución es óptima si la longitud del circuito es la menor posible.

**Búsqueda en árbol [sin eliminación de estados repetidos]**

Entrada: problema = {nodo-raíz, expandir, test-objetivo} estrategia

function búsqueda-en-árbol (problema, estrategia)

;; devuelve solución o fallo

;; lista-abiertos contiene los nodos de la frontera del árbol de búsqueda

Inicializar árbol-de-búsqueda con nodo-raíz Inicializar lista-abiertos con nodo-raíz

**Iterar**

**If** (lista-abiertos está vacía**) then return** fallo

Seleccionar el nodo a explorar de lista-abiertos de acuerdo con estrategia (En la práctica, lista-abiertos se implementa como una cola de prioridad en la que los nodos están ordenados de acuerdo con estrategia. En tal implementación, se selecciona para su exploración el primer nodo de lista-abiertos)

**If** (nodo satisface test-objetivo)

**then return** solución (camino desde el nodo-raíz hasta nodo)

**else** eliminar nodo de lista-abiertos

expandir nodo

insertar nodos hijo (resultantes de la expansión) en lista-abiertos

Estrategias de búsqueda

Búsqueda no informada (ciega):

Par realizar la búsqueda se utiliza únicamente **la definición del problema**. Es decir, no se utiliza conocimiento experto específico para dicho problema.

Las distintas estrategias de búsqueda difieren en el **orden** en el que los nodos son expandidos.

– Búsqueda primero en anchura.

– Búsqueda de coste uniforme.

– Búsqueda primero-en-profundidad.

– Búsqueda de profundidad limitada.

– Búsqueda primero en profundidad con profundidad iterativa

– Búsqueda bidireccional.

Búsqueda informada (**heurística**): Se usa una heurística para guiar la búsqueda. La heurística proporciona, para cada uno de los estados de búsqueda, una estimación del coste del camino óptimo (de menor coste) desde dicho estado a un estado que cumpla el objetivo.

Rendimiento:

Completitud: ¿garantiza el algoritmo de búsqueda encontrar una solución?

Optimalidad : ¿garantiza el algoritmo encontrar una solución óptima (de menor coste)?

• Coste del camino: dado por la función de utilidad.

• Coste computacional de la búsqueda (análisis de complejidad)

– Complejidad temporal. ´

– Complejidad espacial (uso de memoria).

**El coste computacional de la búsqueda se expresa en términos de los siguientes parámetros**: – Factor de ramificación (b): número máximo de sucesores de cualquier nodo.

– Profundidad del nodo objetivo menos profundo (d)

– Profundidad máxima del árbol de búsqueda (m)

– Coste mínimo de una acción (ε)

Hipótesis:

– El factor de ramificación (b) es finito.

– Desde el punto de vista del análisis de complejidad, todas las operaciones tienen el mismo coste.

– La profundidad máxima del árbol de búsqueda (m) puede ser infinita.

– Costes no negativos y aditivos: Coste del camino = suma de costes de cada paso.

Búsqueda primero-en-anchura:

Expandir todos los nodos en una profundidad dada antes de expandir nodos en una profundidad mayor.

Implementación:

– Usar búsqueda-en-árbol utilizando una cola FIFO (first-in-first-out) para la lista de candidatos a expandir (lista-abiertos).

Los nodos nuevos generados se meten al final de la cola ⇒ los nodos menos profundo se expanden antes que los más profundos.

Rendimiento:

– Completo: Siempre encuentra el nodo objetivo que está a una profundidad menor.

– Óptimo sólo si el coste del camino es una función no decreciente de la profundidad (por ejemplo, todas las acciones tienen igual coste)

– Complejidad temporal y espacial: Exponencial en b.

Gráfico

Descripción generada automáticamente

Búsqueda de coste uniforme

Expandir nodo con el coste de camino más bajo

Implementación:

– Ordenar la lista de candidatos a expandir de acuerdo con sus costes de camino. Se expande primero el nodo con menor coste

Rendimiento: La búsqueda puede entrar en un bucle infinito si hay un bucle con coste cero en el espacio de estados.

– Completo y óptimo: Siempre encuentra el nodo objetivo con el menor coste de camino [C\*] sii el coste de cada paso es mayor que cero (es decir, si el coste siempre crece si la longitud del camino crece). ´

Texto

Descripción generada automáticamente– Complejidad: El caso más complejo espacialmente y temporalmente es

Si todos los pasos tienen igual coste, la búsqueda de coste uniforme es equivalente a la búsqueda primero-en-anchura.

Búsqueda primero-en-profundidad

Expandir primero los nodos más profundos de la frontera.

Implementación:

– Usar búsqueda-en-árbol con una cola LIFO (last-in-first-out), (pila) para implementar la lista de candidatos a ser expandidos Los nodos nuevos generados se introducen al principio de la cola ⇒ los nodos más profundos se expanden primero. ´

– Alternativa: Función recursiva que se llama a sí misma por cada uno de sus hijos.

Rendimiento:

– No es completa: podría no parar en árboles que no tienen una profundidad máxima.

– No es óptima

– Complejidad:

Asumamos que la profundidad máxima del árbol, m, es finita

• Complejidad temporal: Peor caso O(bm)

• Complejidad espacial: Se debe llevar cuenta del camino desde nodo raíz a nodo hoja + hermanos no expandidos de cada nodo del camino, es decir, (b·m+1) nodos

Búsqueda con vuelta atrás (backtracking)

Variante de la búsqueda con profundidad limitada, con uso eficiente de la memoria.

Sólo se genera un sucesor en cada expansión. En cada nodo parcialmente expandido se recuerda cuál es el siguiente sucesor para generar.

⇒ O (m) estados

Generar sucesor modificando el estado actual (en vez de copiar + modificar). Se deben poder deshacer modificaciones cuando se vaya hacia atrás para generar los siguientes sucesores.

⇒ un solo estado + O (m) acciones

Búsqueda primero-en-profundidad con límite de profundidad

Expandir primero los nodos más profundos del conjunto frontera hasta un límite de profundidad l.

Implementación:

– Igual que la búsqueda primero-en-profundidad, asumiendo que los nodos con profundidad igual a l no tienen sucesores.

Rendimiento:

**– No es completo** si l < d

– **Optimalidad**

**• No es óptimo** si l > d.

• **Óptimo** si l = d y los costes de cada paso son iguales.

– Complejidad:

**• Complejidad temporal**: Peor caso O(bl )

• **Complejidad espacial:** O(b·l)

Selección de la profundidad límite (l)

– Usar conocimiento específico del problema.

Por ej., el problema de las N reinas tiene como mucho N pasos en la formulación de estado-completo, y exactamente N pasos en la formulación incremental

– **Diámetro** del problema: Número máximo de pasos necesarios para alcanzar cualquier estado desde cualquier otro estado.

Búsqueda primero-en-profundidad con límite de profundidad

Expandir nodos hasta **un límite de profundidad máxima**. Una vez realizada la expansión hasta tal profundidad se **incrementa** el límite de profundidad y se repite la búsqueda.

Combina ventajas de búsqueda primero-en-anchura y búsqueda primero-en-profundidad

– **Completa** si el factor de ramificación (b) es finito.

– **Óptima** si los costes de cada paso son iguales.

– Complejidad:

Texto

Descripción generada automáticamente

Es la estrategia de búsqueda no informada que debe utilizarse cuando el espacio de búsqueda es grande y se desconoce el valor de d.

**Búsqueda uniforme con límite de cose, en la que se incrementa de manera iterativo:** Limitar el coste máximo del camino e incrementarlo a medida que se generan nodos (normalmente poco eficiente).

Búsqueda bidireccional

**Combinar búsqueda hacia delante** (del estado inicial al estado final) **y búsqueda hacia atrás** (del estado final al estado inicial)

Implementación:

– Se encuentra la solución cuando el nodo a expandir en una búsqueda está en el conjunto frontera del otro árbol de búsqueda.

– Puede usarse en combinación con cualquier estrategia de búsqueda.

Rendimiento:

Asumamos que ambas búsquedas son primero-en-anchura

– El chequeo de la pertenencia de un nodo puede realizarse en tiempo constante a través de una tabla hash.

– **Completa** si el factor de ramificación (b ) es finito.

– **Óptima** si los costes de cada paso son iguales.

– Complejidad:

**• Complejidad temporal:** O(bd/2)

• **Complejidad espacial:** Se debe mantener en memoria al menos un árbol de búsqueda O(bd/2)

Dificultades:

– La función predecesora debería ser computable de manera eficiente.

– Definición implícita de estados objetivo (por ej. N reinas)

Resumen de algoritmos de búsqueda no informada

Captura de pantalla de un celular

Descripción generada automáticamente

Estados repetidos

En caso de que no se detecte que se están generando estados repetidos, la eficiencia de la búsqueda se reduce.

Por ejemplo, búsqueda en una malla regular (presencia de ciclos en el grafo)

Aristas múltiples:

Diagrama

Descripción generada automáticamente

**Posibles soluciones**

Elegir una representación que evite repetición de estados.

Eliminación de estados repetidos: No expandir un nodo si ya ha sido generado con antelación

– Requerimientos extra de memoria.

– Óptimo en búsqueda con coste uniforme, y en búsqueda primero en anchura cuando el coste del paso es constante.

– No se garantiza encontrar el camino óptimo con otras estrategias.

Búsqueda en grafo [con eliminación de estados repetidos]

**Entrada**: problema = {nodo-raíz, expandir, test-objetivo} estrategia

**function** búsqueda-en-grafo (problema, estrategia)

;; devuelve solución o fallo

;; lista-abiertos: nodos terminales del árbol-de-búsqueda (generados pero no expandidos)

;; lista-cerrados: nodos generados que ya han sido considerados para ser expandidos

Inicializar árbol-de-búsqueda con nodo-raíz

Inicializar lista-abiertos con nodo-raíz

Inicializar lista-cerrados a una lista vacía

**Iterar**

**If** (lista-abiertos está vacía**) then return** fallo

Seleccionar el nodo a explorar de lista-abiertos de acuerdo con estrategia

**If** (nodo satisface test-objetivo)

**then return** solución (camino desde nodo-raíz a nodo)

eliminar nodo de lista-abiertos

**If** (nodo no está en lista-cerrados)

**then** añadir nodo a lista-cerrados

expandir nodo

insertar los nodos hijo (resultantes de la expansión) en lista-abiertos

Tipos de problemas

Problema con estados unívocamente definidos: Espacio determinista, completamente observable (las mediciones de los sensores del agente de búsqueda permiten determinar el estado real del entorno de manera unívoca) .

Problema conformado: No observable (problemas sin sensores).

Problema de contingencia No determinista, y posiblemente parcialmente observable. Problema de exploración Espacio de estados desconocido.

Problemas de contingencia y exploración:

Los algoritmos **estándar de búsqueda no son, en general, adecuados** para resolver este tipo de problemas.

El agente puede obtener **información** del estado de su entorno por medio de de sus sensores **después de realizar una acción.**

Propuesta: **alternar búsqueda y ejecución.**

Mundo de la aspiradora:

Estados físicos:

Agente: Aspiradora en uno de las dos estancias adyacentes.

Cada estancia puede estar sucia / limpia ζ

Acciones:

L / R: mover a la izquierda / derecha

S: Aspirar suciedad

NoOp: No hacer nada

Objetivo: Mundo limpio (estado 7 u 8)

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente

Problemas conformados

Si el agente no tiene sensores, entonces.

Los estados de búsqueda son estados de creencia: conjunto de los posibles estados físicos del mundo.

A través de una secuencia de acciones uno puede ser capaz de realizar acciones que modifique el entorno de forma que se alcance un estado objetivo.

Diagrama

Descripción generada automáticamente

Problemas de contingencia

La solución no puede ser formulada en general como una secuencia fija de acciones.

Forma

Descripción generada automáticamente**Plan de contingencia**: La solución puede ser dada como un árbol de decisión en el se elige una rama u otra dependiendo de las percepciones.

Ejemplo:

• Problema con sensores

Estado inicial: #5.

**Solución**: [Right, Suck]

• Problema conformado (sin sensores)

Estado inicial {1, 2, 3, 4, 5, 6, 7, 8}

**Solución**: [Right, Suck, Left, Suck]

Problema de contingencia.

• El aspirador es defectuoso: puede ensuciar la habitación cuando está en funcionamiento.

• Sensores: localización, suciedad.

• Estado inicial: #5.

• **Solución**: **Iterate**[Right, if suelo-sucio then Suck] [no interesa aspirar en caso de que el suelo esté limpio].

**TEMA 2.2 búsqueda informada:**

Las estrategias de búsqueda no informada (ciega) son generalmente muy ineficientes.

El uso de conocimiento específico sobre el problema (más allá de la definición del problema)

para guiar la búsqueda puede mejorar enormemente la eficiencia.

Versión informada de algoritmos de búsqueda general: búsqueda primero-el-mejor

Búsqueda avariciosa primero-el-mejor

Búsqueda A\*

Búsqueda heurística con memoria acotada

♣ IDA\* (A\* con profundidad iterativa)

♣ RBFS (búsqueda primero-el-mejor recursiva)

♣ MA\* (A\* con memoria acotada)

♣ SMA\* (MA\* simplificada)

Funciones heurísticas

Búsqueda local y optimización.

Búsqueda online y exploración.

**Búsqueda primero-el-mejor**

𝑓∗ (𝑛)= coste del camino óptimo (de menor coste) desde el nodo n a algún nodo que cumpla

el objetivo.

Explorar de entre los nodos en lista-abiertos el que tenga menor valor de 𝑓∗ (𝑛).

Implementación:

Usar búsqueda-en-árbol o búsqueda-en-grafo con una cola de prioridad para la lista de

candidatos para expandir (lista-abierta).

Los nodos nuevos que se generan se insertan en la cola ordenados de menor a mayor

valor para 𝑓∗ (𝑛)= ⇒ nodos con menor valor de 𝑓∗ (𝑛)= se expanden primero.

Rendimiento:

Por definición es óptima, completa y tiene la menor complejidad posible.

Problema 𝑓∗ (𝑛)= es generalmente desconocida. Podemos usar solamente estimaciones de la

distancia entre un nodo dado y el objetivo basadas en conocimiento experto que aprovecha

características específicas del problema.

Funciones de coste ¬

𝑔 (𝑛) : Coste real (no necesariamente óptimo) de la trayectoria seguida desde el

estado inicial hasta 𝑛.

ℎ (𝑛) : Heurística. Estimación del coste de la trayectoria óptima (de menor coste)

desde el nodo 𝑛𝑛 hasta un nodo objetivo.

𝑓(𝑛) = 𝑔(𝑛) + ℎ (𝑛) : Estimación del coste de la trayectoria seguida la trayectoria

desde el nodo inicial hasta un nodo objetivo pasando por el nodo (𝑛).

Búsqueda primero-el-mejor

Búsqueda en árbol o en grafo

Cola de prioridad ordenada por

(𝑓)∗ (𝑛): Búsqueda óptima o

𝑔 (𝑛): De coste uniforme

ℎ (𝑛): Búsqueda avariciosa

𝑓 (𝑛): Búsqueda A

Gráfico, Gráfico radial

Descripción generada automáticamenteEncontrar el trayecto más corto desde una ciudad hasta Bucarest

Función heurística (admisible, monótona) ℎ (𝑛) = distancia en línea recta desde la ciudad

Diagrama

Descripción generada automáticamentecorrespondiente a Bucarest.

**Búsqueda avariciosa: análisis de complejidad**

Diagrama

Descripción generada automáticamenteSi no se eliminan estados repetidos, la búsqueda primero-el-mejor avariciosa puede no llegar a encontrar una solución.

Ejemplo: Encontrar el itinerario entre “Iasi” y “Fagaras”:

Análisis del algoritmo

Completo solo si se eliminan estados repetidos.

No es óptimo

Complejidad: Temporal: 𝑂𝑂(𝑏𝑏𝑚𝑚), Espacial: 𝑂𝑂(𝑏𝑏𝑚𝑚) Todos los nodos se almacenan en memoria

[𝑏𝑏 = factor de ramificación]

[𝑚𝑚 = profundidad máxima del árbol de búsqueda]

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

**Heurística admisible**

La función heurística ℎ(𝑛) es admisible si nunca sobreestima el coste de alcanzar el objetivo (es decir, es una estimación optimista).

ℎ(𝑛) ≤ ℎ∗(𝑛), ∀n

ℎ∗(𝑛𝑛) = coste real del camino óptimo (es decir, con menor coste) desde n hasta algún nodo objetivo. Ejemplo: En el problema del mapa de carreteras, la distancia en línea recta es una función heurística admisible.

TEOREMA: Si se usa búsqueda-en-árbol, y h es admisible ⇒ A\* es completa y óptima Demostración:

Una captura de pantalla de un celular

Descripción generada automáticamente

**A\* + heurística admisible**

Si se usa búsqueda-en-grafo, A\* puede no ser óptima incluso si h es admisible: Se pueden generar soluciones subóptimas si el camino óptimo a un estado repetido no es el que primero se genera.

Diagrama

Descripción generada automáticamente

Solución: Descartar el camino con coste más alto.

Diagrama

Descripción generada automáticamenteAumenta la complejidad del algoritmo: necesita eliminar de lista-abierta el nodo con coste más alto y sus descendientes.

**Heurística monótona (o “consistente”)**

Una función heurística h(n) es monótona si se satisface la siguiente desigualdad triangular:

Texto

Descripción generada automáticamente

Ejemplo: Para el problema del mapa de carreteras, la distancia en línea recta es una función heurística monótona.

Si h es monótona ⇒ h es admisible [Ejercicio: Demostrar esto]

Hay heurísticas admisibles que no son monótonas

Diagrama

Descripción generada automáticamente

**A\* + heurística monótona**

TEOREMA: Si h es monótona ⇒ los valores de f(n) a lo largo del camino buscado por A\* son no decrecientes

Demostración:

Texto

Descripción generada automáticamente

TEOREMA: Si h es monótona ⇒ A\* usando búsqueda-en-grafo es completa y óptima. Demostración: Dado que f(n) es no-decreciente el primer nodo objetivo expandido debe ser el correspondiente a la solución óptima.

TEOREMA: Si h es monótona y A\* ha expandido un nodo n, se cumple g(n)=g\*(n) Demostración: Consideremos el problema de búsqueda relacionado con el mismo estado inicial y con nodo n como nodo objetivo.

Definamos la nueva heurística para este nuevo problema: h’(m) = h(m) – h(n), ∀ m / f(m) ≤ f(n).

Dado que la diferencia entre h y h’ es una constante:

Las búsquedas (A\* con h) y (A\* con h’) empezando desde el mismo estado inicial, exploran la misma secuencia de nodos antes de expandir n.

h’ es una heurística monótona para el nuevo problema.

Dado que A\* con una heurística monótona es completa y óptima, la búsqueda (A\* con h’) encuentra el camino óptimo entre el estado inicial y el nodo n. Este camino será también el óptimo para la búsqueda (A\* con h) ⇒ g(n)=g\*(n).

TEOREMA: A\* es óptimamente eficiente.

Para una heurística dada, ningún otro algoritmo expandirá menos nodos que los que expande A\* (excepto posibles empates) Demostración: si hay nodos que no se expanden entre el origen y la curva de nivel óptima, no está garantizado que el algoritmo encuentre la solución óptima.

Si h es una heurística monótona, la exploración se realiza en curvas de nivel con valores crecientes de f(n).

• En una búsqueda de coste uniforme [h(n) =0], las curvas de nivel son “concéntricas” alrededor del estado de partida.

• En heurísticas mejores estas curvas forman bandas que se extienden hacia el estado objetivo.

Diagrama

Descripción generada automáticamente

**Complejidad de A\***

Texto

Descripción generada automáticamente

Complejidad espacial:

Igual a la complejidad temporal (se mantienen todos los nodos en memoria).

Normalmente es el factor limitante.

Debido al gran requerimiento de memoria, A\* no es un algoritmo práctico para problemas grandes.

**Búsqueda IDA\***

Búsqueda A\* con profundidad iterativa Realizar una búsqueda primero-en-profundidad con una profundidad límite de f, e ir aumentando este valor. n0= nodo inicial

Sean

Texto, Carta

Descripción generada automáticamente

Texto, Carta

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama, Mapa

Descripción generada automáticamente

Diagrama, Mapa

Descripción generada automáticamente

**Comparación entre algoritmos de búsqueda**

Con el fin de comparar la eficiencia de distintos algoritmos de búsqueda, o de distintas heurísticas en búsqueda informada, se define para un factor de ramificación efectivo (𝑏):

𝑁: Número de nodos expandidos para alcanzar la solución.

𝑑 : Profundidad de la solución encontrada.

𝑏 : factor de ramificación de un árbol equilibrado de profundidad *d.*

Imagen que contiene Gráfico

Descripción generada automáticamente

Promediar los valores de 𝑏 obtenidos para diferentes realizaciones del problema; por ejemplo, para distintos estados iniciales.

Idealmente: 𝑏 lo más cercano posible a 1.

**Diseño de heurísticas: relajación**

Diseño de heurísticas por relajación del problema de búsqueda:

Definir un problema relajado eliminando restricciones del problema original.

Usar como heurística para el problema original la función de coste óptimo del problema relajado.

La heurística obtenida es admisible y monótona (por ser óptima para el problema relajado).

Comparación de heurísticas.

Texto, Carta

Descripción generada automáticamente

**Relajación del 8-puzle**

Problema original: La ficha en la posición A puede moverse a B si A es adyacente a B y B está vacía.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

**Comparación de algoritmos de búsqueda**

IDS: Búsqueda con límite de profundidad, en la que el límite se incrementa de manera iterativa (coste lineal en memoria, exponencial en tiempo)

Tabla

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

La dificultad de la búsqueda depende del perfil de la función de coste: Mínimo global / mínimo local. Mesetas: zonas en las que la función apenas varía. Crestas: zonas en las que la variación es abrupta. Puntos de silla (puerto de montaña). Discontinuidades.

**Búsqueda local (avariciosa)**

Meta: Minimizar la función de coste 𝐸:

1. Inicializa estado-de-búsqueda con el estado inicial.

2. Itera

i. Genera los sucesores de estado-de-búsqueda

ii. En caso de que los valores de la función de coste de los sucesores sean mayores que 𝐸(estado−de−búsqueda), devuelve estado-de-búsqueda.

iii. En caso contrario, actualiza estado-actual con el sucesor con el menor valor de la función de coste.

Características:

Requisitos de memoria muy reducidos: solo se almacena el estado de búsqueda actual.

Solo es completa si el espacio de búsqueda es finito.

No óptima: Converge a un mínimo local que podría coincidir con un mínimo global.

Permite encontrar soluciones aceptables en espacios de estados grandes o infinitos (ej. optimización continua), en los que la búsqueda exhaustiva no es factible.

**Búsqueda local: Variaciones**

Búsqueda local aleatoria:

1. Incializa el estado-de-búsqueda con el estado inicial.

2. Itera hasta convergencia

i. Genera un sucesor de estado-de-búsqueda de manera aleatoria.

ii. En caso de que 𝐸(sucesor) < 𝐸(estado−de−búsqueda) estado-actual <- sucesor Búsqueda local aleatoria en haz

1. Seleccionar aleatoriamente k estados iniciales.

2. Iterar hasta convergencia

i. Generar todos los sucesores de los k estados.

ii. Seleccionar de entre estos sucesores los mejores k estados.

Reinicio aleatorio: Una vez haya convergido el algoritmo de búsqueda desde un estado inicial, se vuelve a realizar la búsqueda desde otro estado inicial elegido de manera aleatoria.

Si hay una probabilidad no nula de que el óptimo sea elegido como estado inicial en alguna de las iteraciones, el algoritmo es asintóticamente (es decir, a medida que el número de reinicios tiende a infinito) completo y óptimo.

Texto

Descripción generada automáticamente

La dirección del gradiente es la de mayor variación local de la función.

𝛼 > 0 es la tasa de descenso.

Debe ser suficientemente pequeña como para que el algoritmo converja, pero suficientemente grande como para que lo haga en un tiempo razonable.

Puede ser modificada a medida que progresa el proceso de descenso.

Si no disponemos de la forma analítica del gradiente, podemos usar estimaciones numéricas.

Texto

Descripción generada automáticamente

Optimización con restricciones: lineales, cuadráticas, programación no lineal.

Cálculo del gradiente

Programado directamente.

Aproximado de manera numérica.

Computado mediante técnicas de diferenciación automática (disponibles, por ejemplo en Pytorch, Tensorflow, Julia, etc.)

Texto

Descripción generada automáticamente

La temperatura debe ser inicialmente elevada (etapa de exploración) y reducirse a medida que progresa la búsqueda (etapa de explotación).

Programa geométrico de temple

Mantener T constante durante un número de iteraciones (una “época”)

En cada época se reduce el valor de T multiplicándolo por 𝛽< 1 (por ejemplo 𝛽 = 0.99): T ← 𝛽T

**Algoritmo genético**

La solución óptima se obtiene como resultado de un proceso de evolución artificial en el que la función a maximizar cuantifica la capacidad de adaptación y supervivencia de los individuos de una población (fitness function)

Codificación del problema:

Un individuo se representa mediante un cromosoma. Por ejemplo, una cadena de bits. El cromosoma codifica una posible solución del problema.

Pseudocódigo

1. Inicializar aleatoriamente una población de M individuos.

2. Iterar hasta convergencia

i. Seleccionar de entre los de la población los m individuos (m < M) de acuerdo con el valor función a maximizar (fitness).

ii. Generar una nueva población de tamaño M a partir de los m individuos seleccionados mediante reproducción sexual. En este proceso se generan hijos cuyo material genético se obtiene por recombinación (cruce) del material genético de los padres (los individuos seleccionados en (i)).

iii. Introducir variaciones en el material genético de los individuos mediante mutaciones aleatorias.

Tema 2.3 búsqueda con adversarios

Búsqueda con adversarios:

Entorno multiagente simplificado en el que dos jugadores con objetivos divergentes alternan turnos en los que realizan movimientos (ejecutan acciones).

Los agentes sólo pueden realizar movimientos acordes a las reglas del juego.

Cada agente decide el movimiento a realizar de acuerdo con una estrategia.

Una vez concluido el juego, se valora el estado terminal de acuerdo función de utilidad para cada uno de los agentes.

Tipos de juegos

Número de jugadores: Dos jugadores (ej. backgammon, ajedrez, damas)  Varios jugadores: estrategias mixtas: competición / colaboración, posiblemente cambiante (por ejemplo alianzas estratégicas temporales).

Propiedades de la función de utilidad

Suma-cero: La suma de utilidades de los agentes es cero, independientemente del resultado del juego (ej. ajedrez, damas)

Suma constante: La suma de utilidades de los agentes es constante, independientemente del resultado del juego (equivalente a juegos de suma cero: normalización de la utilidad)  Suma variable: No son de suma cero. Pueden tener estrategias óptimas complejas, que a veces involucran colaboración. (ej. el dilema del prisionero)

Información de la que disponen los jugadores

Información perfecta (ej. ajedrez, damas, go)  Información parcial (ej. la mayoría de los juegos de cartas)

Elementos de aleatoriedad

Deterministas (ej. ajedrez, damas, go)

Estocásticos (ej. backgammon)

Tiempo ilimitado / limitado para realizar los movimientos (ej. ajedrez con reloj).

Movimientos ilimitados / limitados (ej. ajedrez con máximo de 40 movimientos).

Búsqueda con adversarios

Formalización del problema de búsqueda:

Estado inicial.

Función sucesor: [estado actual]  [lista de sucesores]

Test terminal: [estado actual]  True (el juego ha concluido) False (el juego no ha concluido)

Utilidad (función objetivo o de pago): Valoración numérica de los estados terminales que refleja.

Árbol del juego:

Nodo raíz: Estado inicial

Sucesores generados por secuencias movimientos legales realizados por cada uno de los jugadores de manera alterna.

Nodos hoja: Estados terminales del juego, etiquetados por el valor de la función de utilidad.

Abordaremos juegos del siguiente tipo:

Dos jugadores realizan movimientos por turnos hasta que se alcanza un estado terminal.

Suma cero: El estado terminal es etiquetado por el valor de la función de utilidad desde el punto de vista de uno de los jugadores (generalmente, del que inicia el juego).  Determinista.  Información perfecta.

Árbol de juego para tres en raya

Estado inicial: Tablero 3x3 vacío

Movimientos: Poner una ficha (Jugador 1: x, jugador 2: o) en una de las casillas vacías.  Test terminal: 3 fichas del mismo jugador están alineadasDiagrama, Dibujo de ingeniería

Descripción generada automáticamenteEstrategia óptima: Algoritmo minimax

Para determinar el movimiento óptimo en un estado no terminal del juego se define:

MAX: jugador que tiene el turno en ese estado.

MAX y MIN alternan sus movimientos: En el árbol de juego, los nodos de profundidad par (impar) corresponden al turno de MAX (MIN).

Formalización:

Estado inicial: Configuración inicial del tablero + identidad de MAX.

Función sucesor: sucesores(n)

Test terminal: terminal-test(n)

Función de utilidad: utilidad(n), donde n es un nodo terminal.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Tabla

Descripción generada automáticamenteLa estrategia minimax es óptima para MAX, asumiendo que MIN es un oponente infalible.  En caso de que MIN realice sus movimientos de acuerdo con una estrategia subóptima, podría haber una estrategia para MAX mejor que la minimax.

Completo sólo si el árbol del juego es finito (se puede dar el caso de que haya estrategias óptimas finitas incluso si el árbol de juego es infinito)

Óptima sólo si el oponente es óptimo (si el oponente es subóptimo, podemos usar sus debilidades para encontrar estrategias mejores. Peligroso).

Complejidad temporal exponencial O(bm); b: Factor de ramificación; m = profundidad máxima del árbol del juego

Complejidad espacial: Lineal si se usa búsqueda-primero-en-profundidad O(bm)Diagrama

Descripción generada automáticamente

• El mejor movimiento de MAX es A1  • La mejor respuesto MIN es A11  
  
Diagrama

Descripción generada automáticamente

Poda alfa-beta:

Observación: Para calcular el valor minimax de un nodo muchas veces no es necesario explorar exhaustivamente. El algoritmo realiza una búsqueda primero-en-profundidad desde un nodo dado. Si durante dicha búsqueda se encuentran los nodos m y n en diferentes subárboles, y el nodo m es mejor que el n, entonces, asumiendo decisiones óptimas, nunca se llegará al nodo n en el juego actual.

Asignar AN cada nodo un intervalo [, ] que contiene el valor minimax del nodo, y actualizar los límites del intervalo a medida que avanza la búsqueda

A es el valor de la mejor alternativa para MAX encontrada hasta el momento (es decir, la de mayor valor)  los valores de  nunca disminuyen en un nodo MAX

B es el valor de la mejor alternativa para MIN encontrada hasta el momento (es decir, la de menor valor)  los valores de  nunca aumentan en un nodo MIN.

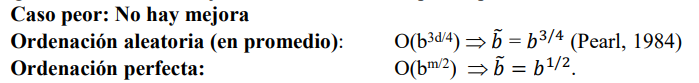
Reglas para parar la búsqueda:

a-cutoff: Parar la búsqueda en un nodo MIN cuyo valor de   valor de  de cualquiera de sus antecesores MAX.

b-cutoff: Parar la búsqueda en un nodo MAX cuyo valor de   valor de  de cualquiera de sus antecesores MIN.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente  
La poda no afecta al resultado final.  La eficacia de la poda depende del orden en el que se exploran los sucesores en el árbol de juego: Es buena si los mejores movimientos se exploran primero.  Caso peor: No hay mejora



El uso de heurísticas sencillas lleva a menudo a 𝑏෨ cercano al óptimo (ej. examinar primero los movimientos que fueron los mejor considerados en el anterior turno)

Diagrama

Descripción generada automáticamente

Decisiones imperfectas

Problema:

Con recursos limitados o juegos para los cuales el árbol de juego es infinito puede no ser posible realizar una búsqueda completa para determinar los valores minimax de los nodos.  Diagrama

Descripción generada automáticamente

Búsqueda con horizonte limitado

Para cada estado no terminal se define una función de evaluación: eval(n)

La función de evaluación es una estimación del valor minimax en dicho nodo (heurística).

Se define un test de corte para determinar cuándo parar la búsqueda y computar eval(n).

Propiedades de eval(n):

En los nodos terminales eval(n) debería coincidir con utilidad(n). En caso de que no coincida, debería conducir una ordenación de los nodos terminales en el mismo orden que utilidad(n).

En los nodos terminales eval(n) debería estar muy correlacionada con minimax(n).

eval(n) debería ser fácil computar.

Texto

Descripción generada automáticamente

La función de F puede puede ser o Determinada por conocimiento experto. o Aprendida a partir los resultados de partidas jugadas de manera automática (ej. AlphaZero).

Búsqueda de horizonte limitado

Modificar en búsqueda minimax:

If test-terminal(estado)

then return utilidad(estado)

else if test-corte(estado, profundidad)

then return eval(estado)

Búsqueda primero-en-profundidad con límit de profundidad: Terminar la búsqueda para un valor fijado de profundidad.

Búsqueda primero-en-profundidad con límite de profundidad que se incrementa de manera iterativa: Más robusto si hay limitaciones en el tiempo.

Efecto del horizonte: Desastre / éxito puede estar justo tras el horizonte de búsqueda.

No detener la búsqueda en posiciones activas. Es necesario una heurística para determinar si una posición es activa.

Extensiones singulares: Explorar con mayor profundidad si en la búsqueda se encuentran estados ventajosos.

Prepoda: Evitar explorar ramas de búsqueda que son claramente inferiores (peligro: podemos no darnos cuenta de que hay una jugada ventajosa en la rama no explorada).

Ajedrez:

Tiene aproximadamente 1040 nodos, b = 35.  Suponiendo que cada sucesor puede ser procesado en 1 s (10-6 s)

Una exploración completa tomaría 31026 años (la edad del universo es ~ 1010 años).  Si asumimos una limitación temporal de 1 minuto por movimiento, sólo podremos realizar una exploración completa hasta profundidad 5.

Con poda alfa-beta + ordenamiento perfecto, podemos realizar una exploración completa hasta profundidad 10.

Función de evaluación simple desde el punto de vista de blancas basada en asignar un valor a cada una de las piezas en el tablero.

peón=1; caballo y alfil=3; torre = 5; reina = 9. f =1\*(n-peones-blancos - n-peones-negros) + 3\*(n-alfiles-blancos - n-alfiles-negros) +3\*(n-caballos-blancos - n-caballos-negros) + 5\*(n-torres-blancas - n-torres-negras) +9\*(n-reinas-blancas - n-reinas-negras)

Funciones de evaluación más complejas que toman en cuenta otras características

control del centro

buena posición del rey

buena estructura de peones”. Es necesario computar los valores numéricos de tales características para cada uno de los estados de juego considerados.  Uso de librerías de movimientos (aperturas, finales de partida).

Diagrama

Descripción generada automáticamente con confianza baja

Juegos de azar  Se introduce un agente adicional correspondiente al proceso aleatorio  Turno de MAX = movimiento de RAND + movimiento de MAX 5

Turno de MIN = movimiento de RAND + movimiento de MIN

Movimiento de RAND = tirar una moneda, un dado, etc.

El árbol del juego tiene nodos MAX + nodos MIN + nodos RAND

Valor expectiminimax:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En caso de que se utilice búsqueda con horizonte limitado, este algoritmo sería óptimo en caso de que la función de evaluación fuera una transformación lineal positiva del valor expect minimax para cada estado.

La complejidad temporal es exponencial: O(bd n d ).

b = factor de ramificación de nodos MAX / MIN.

n = factor de ramificación de los nodos RAND.

d = límite en la búsqueda en profundidad.

Se puede realizar poda en los nodos RAND.