Tema 3 - Búsqueda en espacios de estados

- Diseño de un agente deliberativo: búsqueda
  - Componentes de la descripción de un problema
    - Estado inicial y objetivo
    - Efecto de las acciones → modelo de transición
    - Espacio de estados
      - Conjunto de estados alcanzables desde el estado inicial mediante una secuencia de acciones
    - Grafo de estados
      - Elementos
        - Nodo → Estado
        - Arco → Acción
      - Tipos de grafos
        - Grafo explícito
          - Representa el espacio de estados explorados hasta el momento
          - Se almacena físicamente en memoria
          - Se construye conforme aplicamos acciones
          - Puede ser necesario aplicar en sentido inverso (razonamiento regresivo)
        - Grafo implícito
          - Representa el espacio de estados en su totalidad
          - No se almacena físicamente en memoria
    - Comprobación de objetivo
    - Costo de las acciones
    - Solución (óptima)
  - Agente deliberativo
    - Dispone de modelo del mundo que habita
    - Dispone de modelo de efectos de sus acciones sobre el mundo
    - Capaz de razonar sobre esos modelos para decidir cómo conseguir un objetivo
  - Búsqueda en espacio de estados
    - Espacio de estados
      - Representación del conocimiento a través de acciones del agente
    - Búsqueda en espacio de estados
      - Resolución de problema mediante proyección de distintas acciones
- Sistemas de búsqueda y estrategias
  - Procedimiento de búsqueda
    - Datos extraídos de base de datos inicial
    - Hasta que los datos satisfacen la condición de terminación
      - Selección de acción del conjunto de acciones que pueden ser aplicadas al conjunto de datos
      - Datos → resultado de aplicar acción a conjunto de datos
  - Estrategias de control
    - Estrategias irrevocables

- Grafo explícito formado por único nodo (en cada momento) con información completa del sistema

▼ Proceso

- Selección de acción
- Aplicación de acción sobre estado del sistema
- Se borra de memoria estado previo a la acción y se sustituye por el estado nuevo

▼ Estrategias tentativas

▼ Retroactivas (backtracking)

- Sólo se guarda un hijo de cada estado en memoria
- Se mantiene el camino desde el estado inicial hasta el actual
- Grafo explícito → realmente es una lista

▼ Posibles condiciones de parada del proceso

- Llegar al objetivo y no se desea encontrar más soluciones
- No más operadores aplicables al nodo raíz

▼ Posibles condiciones de vuelta atrás

- Se encuentra solución pero se quiere una solución alternativa
- Se alcanza un límite de nivel de profundidad o tiempo de exploración
- Se genera un estado que ya existía en el camino
- No existen reglas aplicables al último nodo del grafo

▼ Búsqueda en grafos/árboles

- Se guardan todos los estados

▼ Proceso

- Seleccionar un estado del grafo
- Seleccionar un operador aplicable sobre dicho estado
- Aplicar operador y obtener nuevo nodo
- Añadir arco old_nodo → nuevo_nodo
- Repetir

▼ Tipos de búsqueda

▼ Búsqueda en árboles

- Lista donde se almacenan todos los nodos generados y pendientes de explorar → Abiertos

▼ Búsqueda en grafos

- Lista de nodos generados y pendientes de explorar → Abiertos
- Lista de nodos ya explorados → Cerrados

▼ Infraestructura para los algoritmos de búsqueda

- Estado de un nodo → resultado de la acción del nodo padre
- Nodo padre → actualizado después de la acción
- Coste → coste asociado al nodo padre + coste de la acción del nodo padre

▼ Medidas del comportamiento de un sistema de búsqueda

▼ Completitud

- Hay garantía de encontrar la solución si ésta existe

▼ Optimalidad

- Hay garantía de encontrar la solución óptima

- Complejidad en tiempo
  - Cuánto tiempo se requiere para encontrar la solución
- Complejidad en espacio
  - Cuánta memoria se requiere para realizar la búsqueda

- Búsqueda sin información
  - Búsqueda en anchura
    - Características
      - Usa lista (FIFO) de nodos abiertos y cerrados
      - Consumo de memoria exponencial
      - Completo
        - Encuentra solución si existe
        - Factor de ramificación finito en cada nodo
      - Optimalidad
        - Si todos los operadores tienen mismo coste → encuentra solución óptima
      - Eficiencia
        - Buena (si las metas están cercanas)
    - Algoritmo
      - Nodo inicial con estado inicial y coste 0
      - ¿Nodo inicial cumple condición objetivo (resuelve el problema)?
        - Sí → se devuelve como solución
      - Inicialización de frontera
        - `frontier` → cola FIFO con el nodo inicial
        - `explored` → conjunto con el nodo inicial
      - Bucle principal
        - ¿ `frontier` está vacía?
          - Sí
            - No hay más nodos por explorar
            - No se ha encontrado solución
            - Se devuelve fallo
          - No
            - Se extrae de `frontier` el nodo que haya entrada primero
            - Se añade su estado al conjunto `explored`
        - Expansión del nodo
          - Para cada acción posible desde el estado actual
            - Se genera nuevo hijo
            - Si el estado del hijo **no** está en `explored` ni en la `frontier` (es decir, no ha sido visitado ni está pendiente de visitar), se continúa con él.
            - ¿Nuevo estado hijo cumple condición objetivo?
              - Sí
                - Se devuelve como solución
              - No
                - Se añade nodo hijo a `frontier` para ser explorado más adelante

- Búsqueda con costo uniforme
  - Igual que búsqueda en anchura pero el nodo guarda información sobre el coste del camino desde el nodo inicial
  - Algoritmo
    - Nodo inicial con estado inicial y coste 0
    - Inicialización de frontera
      - `frontier` → cola con prioridad (según su coste acumulado) con nodo inicial
      - `explored` → conjunto con el nodo inicial
    - Bucle principal
      - ¿`frontier` está vacía?
        - Sí
          - No hay más nodos por explorar
          - No se ha encontrado solución
          - Se devuelve fallo
        - No
          - Se extrae de `frontier` el nodo con menor coste acumulado
          - Si ese nodo cumple condición objetivo → se devuelve como solución
          - Se añade su estado al conjunto `explored`
      - Expansión del nodo
        - Para cada acción posible desde el estado actual
          - Se genera nuevo hijo
          - ¿El estado del hijo está en `frontier`?
            - Sí
              - Si el nuevo camino hacia él es más barato que el anterior, se reemplaza el nodo anterior en `frontier` por este nuevo nodo de menor coste
            - No
              - Si tampoco está en `explored`, se añade a `frontier`.
- Búsqueda en profundidad
  - Igual que búsqueda en anchura pero usa una pila (LIFO) en vez de una cola (FIFO)
- Búsqueda en profundidad retroactiva
  - Características
    - Técnica basada en recursividad
    - No es bueno cuando hay ciclos
    - Completitud
      - No asegura encontrar la solución
    - Optimalidad
      - No asegura encontrar la solución óptima
    - Eficiencia
      - Si las metas están alejadas del estado inicial
      - Si hay problemas de memoria
  - Algoritmo
    - Función principal recibe un problema y un límite de profundidad `limit`

- ▼ Función recursiva `RECURSIVE-DLS` pasando el nodo inicial del problema y el límite
  - ▼ Caso base 1 (meta alcanzada)
    - Si el estado del nodo actual cumple el objetivo, devuelve la solución
  - ▼ Case base 2 (se alcanzó el límite de profundidad)
    - Devuelve `cutoff` (se ha recortado la búsqueda de este nodo)
  - ▼ Caso recursivo (continuar explorando)
    - Se inicializa marcador `cutoff_occurred` como falso
    - Se exploran todas las acciones posibles desde el estado actual
    - ▼ Para cada acción
      - Se genera nuevo hijo
      - ▼ Se llama recursivamente a `RECURSIVE-DLS` con este hijo y límite `limit` - 1
        - Si la llamada a `RECURSIVE-DLS` devuelve `cutoff` y se marca `cutoff_occurred = true`
        - Si devuelve una solución, se devuelve
    - ▼ Después de explorar todas las opciones
      - Si `cutoff_occurred = true`, devuelve `cutoff`
      - Si `cutoff_occurred = false` pero no se encontró una solución, devuelve `failure`

▼ Descenso iterativo

▼ Algoritmo

- Función principal recibe un problema y devuelve una solución o fallo
- ▼ Bucle de profundización progresiva
  - Bucle donde profundidad `depth` comienza en 0 y se incrementa indefinidamente
  - ▼ En cada iteración se llama a función `DEPTH-LIMITED-SEARCH` limitando la profundidad de búsqueda al valor actual `depth`
    - ▼ ¿La búsqueda devuelve solución?
      - ▼ Sí
        - Se devuelve
      - ▼ No
        - Se incrementa la profundiad
        - Se repite el proceso

▼ Búsqueda bidireccional

- Búsqueda desde estado inicial y desde el estado final
- Esperanza de poder encontrar en el medio

▼ Búsqueda con información

▼ Heurística

▼ ¿Se tiene conocimiento perfecto?

- Si → algoritmo exacto
- No → búsqueda sin información

- Conocimiento parcial sobre un problema que permite resolver problemas eficientemente
- Método para resolver problemas
- No garantiza solución óptima pero produce resultados satisfactorias

▼ Métodos de escalada

- Búsqueda local consistente en seleccionar la mejor solución en el vecindario de una solución inicial y viajar por las soluciones hasta encontrar un óptimo

▼ Escalada simple

▼ Algoritmo

- Estado activo → E
- Función heurística → f()

▼ Mientras E no sea el objetivo y queden nodos por explorar

- Seleccionar operador A para aplicar a E → A(E)
- Evaluar función heurística en E aplicándole el operador → f(A(E))
- Si f(A(E)) es mejor solución que f(E) → se sustituye E por A(E)

- Se puede quedar atascado

▼ Escalada por máxima pendiente

▼ Algoritmo

- Estado activo → E
- Función heurística → f()

▼ Mientras E no sea el objetivo y queden nodos por explorar

- Para todos los operadores A, obtener A(E)
- Evaluar función heurística en E para todos los A(E)
- Seleccionar E_max tal que f(E_max) = max{f(A(E))}

▼ ¿f(E_max) > f(E)?

- Sí → E = E_max
- No → Devolver E

▼ Características

- Contempla todos los vecinos que se pueden generar a partir del estado actual
- Escoge el mejor de todos los vecinos evaluando la heurística
- Se puede seguir quedando atascado

▼ Completitud

- No tiene por qué encontrar la solución

▼ Admisibilidad

- No siendo completo, aún menos será admisible

▼ Eficiencia

- Rápido y útil si la función es monótona y (de)creciente

▼ Variaciones estocásticas

▼ Escalada estocástico

- Distribución de probabilidad
- Sorteo entre los descendientes que mejoran al actual

▼ Escalada de primera opción

- Genera sucesores aleatoriamente hasta que genera uno mejor que el actual
- Buena estrategia cuando un estado tiene muchos sucesores

▼ Escalada de reinicio aleatorio

- Cuando está cerca de atascarse genera una función aleatoria para continuar el proceso
- Útil cuando las características del problema permiten generar soluciones aleatorias fácilmente

- Enfriamiento simulado
  - Algoritmo
    - Parámetros del algoritmo
      - `problem` → el problema que se desea resolver.
      - `schedule` → función que determina la temperatura en cada instante `t`
    - Inicialización
      - Nodo inicial `current` → representa estado actual
    - Bucle principal
      - Temperatura actual `T` como resultado de la función `schedule(t)`
      - Si `T` = 0
        - Sistema enfriado completamente
        - Devuelve `current` como resultado final
      - Selección aleatoria de estado `next` sucesor de `current`
      - Cálculo de diferencia de valor entre `next` y `current` (ΔE)
      - Decisión de movimiento
        - Si ΔE > 0
          - `next` mejor que `current` → `curent` = `next`
        - Si ΔE ≤ 0
          - `next` peor que `current`
          - Podría aceptarse, pero con una probabilidad que depende de
            - Cómo de malo es `next` (valor de ΔE)
            - Temperatura `T` → $e^{\Delta E/T}$
          - Menor temperatura → menor probabilidad de aceptar estado peor
  - Características
    - Método de búsqueda local
    - Permite visitar soluciones peores que la actual para evitar óptimos locales
    - La solución inicial se puede generar de forma aleatoria o por conocimiento experto
    - La actualización de temperatura es heurística
      - Métodos
        - $T = \alpha \times T$, con α en (0, 1)
        - $T = 1/(1 + k)$, con k = nº de iteraciones del algoritmo hasta el momento
      - Número de vecinos a generar
        - Fijo → N(T) = constante
        - Dependiente de la temperatura → N(T) = f(T)
    - Temperatura inicial `Ti` y temperatura final `Tf` son parámetros de entrada
    - Método probabilístico → capacidad para salir de óptimos locales
    - Eficiente
    - Fácil de implementar
    - Requiere mucho ensayo y error
- Algoritmos genéticos
  - Características

- Objetivo → encontrar solución cuyo valor de función objetivo sea óptimo
- No necesitan nodo/estado inicial

▼ Conceptos

- Cromosoma → vector representación de una solución
- Gen → característica concreta del cromosoma
- Población → conjunto de soluciones al problema
- Adecuación al entorno → valor de función objetivo
- Selección natural → operador de selección
- Reproducción → operador de cruce
- Mutación → operador de mutación
- Cambio generacional → operador de reemplazo

▼ Proceso

- Generar y evaluar población inicial
- Selección de individuos padres
- Cruce de padres
- Mutación de hijos
- Evaluación de nuevos individuos
- Reemplazo generacional
- Selección de individuos padres (y repetir)

▼ Búsqueda primero el mejor

▼ Greedy

- Expandir al hijo más cercano al objetivo mediante función de evaluación
- Si un nodo no tiene hijos → coste infinito
- g(n) → valor desde el nodo inicial hasta el actual
- h(n) → estimación del valor desde el nodo actual hasta el objetivo

▼ A*

▼ Características

- Estrategia de búsqueda sobre grafos

▼ Heurística → f(n) = g(n) + h(n)

- h(n) → estimación al objetivo

▼ Uso de abiertos y cerrados

- Abiertos → cola con prioridad ordenada según f(n)

▼ Completitud

- Si existe solución la suele encontrar

▼ Admisibilidad

▼ Si existe solución óptima la suele encontrar

- Requiere condiciones muy generales
- Requiere $h(n) \leq h^*(n)$

▼ Estructura de un nodo

- Estado
- Hijos

- Mejor padre
- g
- h

▼ Algoritmo

- Abiertos → nodo inicial
- Cerrados → vacío

▼ Bucle principal (hasta que se encuentra solución o hasta que abiertos queda vacío)

- Selección de mejor nodo de abiertos

▼ ¿Es nodo objetivo?

- Sí → devolver nodo
- No → expandir dicho nodo

▼ Para cada nodo sucesor

▼ ¿Está en abiertos o en cerrados?

- Abiertos → insertarlo manteniendo información de mejor padre
- Cerrados → insertarlo manteniendo información de mejor padre y actualizar información de descendientes
- Otro caso → insertarlo como nuevo nodo

▼ Nodos repetidos

- Repetido en abiertos → comprobar si nuevo camino es mejor que anterior

▼ Repetido en cerrados

▼ Caso 1 → nuevo padre no es mejor que padre anterior

- Fin

▼ Caso 2 → nuevo padre es mejor que padre anterior

- Actualizar enlace al padre
- Actualizar nuevo valor de coste del camino
- Propagar información a los nodos hijos

▼ Casos particulares

▼ h(n) = 0

- Coste de cada arco = 1 → comportamiento igual a búsqueda en anchura
- Otro caso → comportamiento igual a búsqueda de coste uniforme

▼ g(n) = 0

- Comportamiento igual a búsqueda greedy

▼ Búsqueda dirigida

▼ Características

- Variación de A* que limita el factor de ramificación en problemas complejos

▼ Cada vez que expande un nodo

- Se generan sucesores
- Se evalúan los sucesores con la función heurística f
- Se eliminan los sucesores con peor valor de f
- Se queda un número fijo de sucesores

- No siempre obtiene información necesaria del estado del entorno
- Las acciones no siempre conocen sus efectos

- Puede haber otros procesos o agentes en el mundo
- Mientras se construye el plan el mundo puede cambiar
- Se le puede pedir actuar antes de que complete la búsqueda de un estado objetivo
- Consume mucha memoria

▼ Heurísticas sobre el proceso de búsqueda

▼ Búsqueda orientada a subobjetivos

- Trata de identificar subobjetivos que forman parte del camino al objetivo
- Avanza desde el estado inicial hasta el próximo subobjetivo
- Búsquedas locales

▼ Búsqueda con horizonte

- Profundidad máxima con la que se realiza la búsqueda
- A veces requiere cambiar el criterio de búsqueda del objetivo

▼ Búsqueda jerárquica

- Operadores primitivos y abstractos
- Proceso de búsqueda para encontrar solución a nivel alto
- Soluciones de nivel alto utilizadas como subobjetivos para encontrar soluciones a nivel primitivo

▼ Grafo Y/O

▼ Nodos Y

- Para resolver problemas asociados a su padre → necesario resolver todos los componentes que lo suceden
- Resuelve todos los hijos y combina las soluciones
- Soluciona el nodo y devuelve la solución

▼ Nodos O

- Para resolver problemas asociados a su padre → basta con resolver uno de los posibles operadores
- Ir resolviendo hijos hasta ver si devuelve la solución

▼ Nodo terminal

- Resolver subproblema asociado y devolver solución

Algoritmo	Tipo	Estrategia	Completo	Óptimo	Uso de memoria	Heurística	Observa clave
Anchura (amplitud)	Sin información	Expande nodos por nivel (cola FIFO)	✅ Sí	✅ Sí (si costes iguales)	❌ Alto	❌ No	Muy cos memoria solución poco prc
Coste uniforme	Sin información	Expande nodo con menor coste g(n)	✅ Sí	✅ Sí	❌ Alto	❌ No	Útil con diferente lento si h muchos caminos similares
Profundidad	Sin información	Explora camino hasta el fondo (pila)	❌ No	❌ No	✅ Bajo	❌ No	Puede c bucles o caminos infinitos
Profundidad retroactiva	Sin información	Llamada recursiva analizando hijos	❌ No	❌ No	✅ Bajo	❌ No	No es bu cuando h ciclos

Descenso iterativo	Sin información	Repite profundidad limitada creciente	✅ Sí	✅ Sí (si costes iguales)	✅ Bajo	❌ No	Combina mejor de amplitud profundi
Búsqueda bidireccional	Sin información	Búsqueda desde inicio y objetivo	✅ Sí	✅ Sí (si simétrica)	❌ Alta	❌ No	Muy efic se conoc inicio y f necesita operado reversibl
Escalada simple	Con información local	Explora vecino más prometedor	❌ No	❌ No	✅ Muy bajo	✅ Sí (h(n))	Rápida p atasca fácilmen óptimos
Escalada de máxima pendiente	Con información local	Contempla los vecinos que se puedan generar a partir del estado actual y escoge el mayor valor heurístico	❌ No	❌ No	✅ Muy bajo	✅ Sí (h(n))	Rápido y la funció monóton (de)crec
Escalada con reinicio aleatorio	Con información local	Reinicia aleatoriamente tras bloqueo	❌ No	❌ No	✅ Muy bajo	✅ Sí (h(n))	Mejora la probabili salir de ó
Enfriamiento simulado	Con información local	Actualización de temperatura para ir acotando las posibles soluciones	✅ Sí	❌ No	✅ Bajo	✅ Sí (h(n))	Permite solucion peores p salir de ó locales
Primero el mejor (greedy)	Con información	Usa solo h(n)	❌ No	❌ No	❌ Alto	✅ Sí (h(n))	Muy rápi pero sen óptimos
Búsqueda dirigida	Con información	Conserva los *k* mejores nodos por h(n)	❌ No	❌ No	✅ Bajo	✅ Sí (h(n))	Rápida y eficiente completa óptima; u espacios enormes
A*	Con información	Usa f(n) = g(n) + h(n)	✅ Sí	✅ Sí (si h admisible)	❌ Alto	✅ Sí (h(n))	Equilibric coste rea estimaci
Regresión	Con información	Desde el objetivo hacia el inicio	✅ Sí	✅ Sí (depende del dominio	❌ Alta	✅ A veces	Muy útil planifica resolució inversa