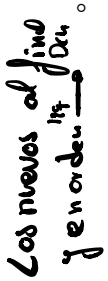
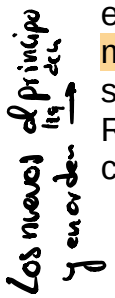
Tema 3: Búsqueda

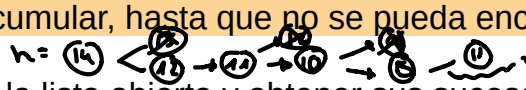
- Son aquello problemas en los que partido de un estado inicial, por el que comenzaremos, y debemos llegar mediante una seria de operaciones a un determinado estado, el estado final. Es importante antes de comenzar un problema de búsqueda conocer cuales son los estado posibles, y entre ellos debe estar el estado inicial y el final. La transición entre estos estados se realiza mediante una serie de operaciones. Se pueden representar como un grafo, en el que la raíz es el estado inicial y de el cuelgan todos los estados resultado de aplicar las operaciones. El numero de estados posibles aumenta exponencialmente cuando varia el numero de datos.
- **Lista abierta**: Lista de todos los nodos que aun no hemos explorado.
- **Parámetros importantes**:
  - **Factor de ramificación, b:** Numero de posibles nodos que pueden derivar de otro, posibles sucesores de un nodo. El factor de ramificación medio es la suma de todos los factores de todos lo estados dividido del número de estados. Ojo tener en cuenta retroceso en el árbol.
  - **Profundidad del árbol de búsqueda, d**: Cuantos operadores tenemos que aplicar para llegar a la solución, solo se puede calcular tras encontrarla, pero no antes.
- **Búsqueda no informada**: No tenemos información de donde esta la solución, solo sabemos cual es, por lo que tenemos que seguir un proceso mecánico lógico que recorra todos los estados. No tienen pesos las ramas, por lo que elegir una u otra es igual si llega a la solución.
  - **Búsqueda en amplitud**: Sigue el orden de un FIFO, el primero que entra es el primero en salir se ejecutan en orden de entrada, una cola. Si un estado vuelve a aparecer no lo ponemos, por que ya esta contemplado. Este método llega al estado final por medio de menos operadores y eso es mejor, es la solución optima. Como no hay pesos en las ramas cuando aparece la solución en una rama hemos acabado.
    - Completitud(hay solución y el factor de ramificación es finito en cada nodo), admisibilidad(si todos los nodos tienen el mismo coste, encuentra la solución optima), eficiencia y consumo de memoria exponencial.
  - **Búsqueda en profundidad:** Sigue el orden de un LIFO. El ultimo en entrar será el primero en ejecutar, una pila. Puede alcanzar la solución en menor tiempo, pero posiblemente lo hará mediante mas operaciones. Se puede poner un limite de profundidad de las ramas, para que no se llegue a la solución con tantas operaciones. No se ponen estados que ya hayan aparecido. Recorre una rama hasta que no puede avanzar mas, entonces retrocede hasta que pueda cansar por una rama.
    - Requiere backtracking(cuando retrocede al llegar al limite de profundidad, no encuentra soluciones en esa rama o solo aparecen duplicados), completitud(no asegura encontrar la solución, aunque la haya), admisibilidad(no asegura que sea la solución optima) y eficiencia(cuando la meta esta lejos del estado inicial o cuando hay problemas de memoria).
  - **Complejidad de amplitud y profundidad:**
    - Amplitud: Temporal y espacial exponencial O(b^d)
    - Profundidad: Temporal exponencial O(b^d) y espacial lineal O(b).
  - **Búsqueda de coste no uniforme**: Cuando aparecen costes en las transiciones. Se buscará el camino a la solución mas barato en costes.
    - **Dijkstra**: Amplitud pero ordenado ascendente. Consiste en ir escogiendo aquellas ramas que menor coste tengas, se ordena la lista abierta de menor a mayor y FIFO, pero si hay un estado repetido hay que conspirar su coste para ver si es mas barato. Cuando vamos avanzamos hay que ir arrastrando el coste acumulado de las ramas ya recorridas. Es como recorrer en profundidad pero escogiendo el mas barato. Termina cuando hemos comprobado que la solución encontrada es la mas optima.
    - **Ramificación y acotación:** En profundidad, seguimos una rama cogiendo el que menos coste hasta una meta, y termina cuando no encontramos otra rama con menor coste que la escogida, ponemos el coste de esa como limite de coste de la rama, ya que si lo supera será mejor la anterior, pero si la encuentra esa es la solución.





- **Pasos a seguir**:
  - Formalizar el problema:
    - Definir lo posibles estados, ademas cual será el inicial y el final o meta.
    - Operaciones que se pueden realizar en los estados.
  - Estimar la complejidad, el numero de estados diferentes del problema.
  - Algoritmo de búsqueda:
    - Orden al generar nodos.
    - Lista abierta, nodos generados y nodos expandidos.
    - Conocer las propiedades del algoritmo: Completitud, admisibilidad, complejidad temporal y espacial.

- **Búsqueda heurística:** Se tiene conocimiento parcial sobre un problema que nos permite resolverlo de una manera mas o menos eficiente. Si se tiene el conocimiento perfecto, se puede desarrollar un algoritmo exacto, pero si no se tiene conocimiento se hará una búsqueda no informada.
  - Hay que hallar una función heurística h(n), que permita hallar el coste estimado desde el nodo n hasta un nodo objetivo, y esta devuelve un valor real positivo. Se descubre resolviendo modelos simplificados del problema real, simplificando mecánicas como restricciones, pero no el propio problema. Es el coste optimo del problema relajado. Ejem: Distancias de Manhattan, para problemas con un tablero con coordenadas, h(n)=|x-xi|+|y-yi|, la suma de distancias.
  - **Búsqueda Escalada (Hill Climbing)**: Básicamente buscando el camino que genera menores h, solo fijándonos en la h y sin acumular, hasta que no se pueda encontrar uno mas bajo que el anterior, o encuentre el final.



Consiste en coger un nodo de la lista abierta y obtener sus sucesores, calculamos la función heurística de cada uno, y nos quedamos en la lista abierta solo con aquel que tiene menor coste(no se acumulan valore, es el puro coste heurístico) y ese coste ademas debe ser menor que el del padre. No hay backtracking, por lo que si ese único nodo que usamos no puede avanzar, hemos terminado. No se analizan los nodos repetidos, ya fueron descartados. En cuanto un nodo sucesor sea final, el proceso termina.
    - No completo(NO, ya que no siempre encuentra la solución), no admisible(NO, si no encuentra siempre la solución no será admisible) y eficiencia (es rápido y útil)
    - Posibles soluciones: Poder hacer backtracking, avanzar por un par de ramas a la vez, reinicio aleatorio o que siga avanzando aunque el sucesor tenga un mayor coste.

A*

~~**Búsqueda Mejor Primero**~~: Básicamente ir calculando para cada sucesor la suma del h propio y el coste en llegar (OJO, pero sin tener en cuenta los h anteriores, es la suma de transiciones pura), e ir avanzando por el que menor suma tiene.
Consiste en ir cogiendo de la lista abierta el nodo de menor coste y obtener sus sucesores, de los que calculamos su coste que es la suma de la función heurística de ese nodo y el coste desde el inicio hasta llegar a ese nodo, y ordenamos la lista abierta en orden ascendente en coste, pero no desechamos ningún nodo. Este algoritmo siempre encuentra la solución y es la mas optima. Los nodos recorridos pasan a la lista cerrada, que servirá para ver si un nodo repetido lo hemos encontrado con menor coste, por lo que se convertiría en mejor camino. Si el nodo final es escogido termina, pero no solo si aparece como un sucesor. f(n)=g(n)+h(n) El coste total es la suma del coste desde el inicio al nodo n y el coste desde n hasta el nodo final.
    - Al final se pueden calcular los valores reales, que se indican con un *.
    - Completitud(si existe solución la encuentra), admisibilidad(encuentra la solucion optima, si: sucesores finitos, los coste son mayores que 1 y h(n) es admisible, menor que el h*(n)) y complejidad exponencial.
    - Cuando mayor h(n) mejor informada esta la función heurística y menor numero de nodos sucesores aparecen.

Todo excepto Mejor primero, pueden terminar sin un sucesor el final. Mejor primero, espera hasta escoger el nodo.

Mejor primero (BF): Es hacer Dijkstra con h(n), manteniendo los hijos y cogiendo el hijo + barato. Termina cuando un nodo final es escogido por su coste.

No informada:

Amplitud — CA
Profundidad — II
} Terminan si en la lista aparece un final.

Ampl-gen) Dijkstra. —CA→ Termina cuando un final es escogido.

Prof-gen) Ramificación y Acotación. → Termina cuando no hay ramas que tengan menor coste que la que llega a un final.
↳ II

Heurística:

Prof-hen) Hill-Climbing —II→ Cuando aparece en la lista termina.

Ampl-hen) Mejor primero —C I→ Cuando un final es escogido, termina. (I→ no tiene en cuenta el coste g(n))

Ampl-hen+gen) A* → Cuando un final es escogido, termina.
↳ CA si h admisible