Tema 1. Búsqueda en Espacios de Estados

Algoritmos de Búsqueda no Informada en Grafos y algunos Problemas

Objetivos

1. Conocer los fundamentos de los algoritmos de búsqueda y el papel que juegan en la Inteligencia Artificial
2. **Conocer el paradigma de Búsqueda en Espacios de Estados y los algoritmos básicos de búsqueda a ciegas** y sobre todo de búsqueda inteligente o heurística
3. Saber cómo **modelar problemas para resolverlos con Búsqueda en Espacios de Estados**, en particular cómo introducir conocimiento específico del dominio del problema

Contenidos

1. Introducción
2. Espacios de búsqueda
3. **Algoritmos de búsqueda no informada**
    1. Generalidades sobre los algoritmos de búsqueda
    2. Algoritmos de búsqueda en árboles
    3. **Algoritmos de búsqueda en grafos**
4. Algoritmos de búsqueda informada o heurística
5. Técnicas de diseño de funciones heurísticas

1.3.3. Algoritmos de Búsqueda no Informada
Búsqueda en Grafos

- La diferencia entre búsqueda en grafos y búsqueda en árboles es la gestión de nodos con estados repetidos
  - En búsqueda en grafos, no hay dos nodos con el mismo estado en el árbol de búsqueda.
  - En búsqueda en árboles, puede haber varios nodos, incluso infinitos, con el mismo estado.
- En búsqueda en grafos, cuando aparece un nodo con un estado repetido debemos quedarnos con el que represente el mejor camino desde el inicial al estado.
- Cuando el espacio de búsqueda es un grafo, se pueden utilizar algoritmos de búsqueda en grafos o en árboles.
- Si el espacio de búsqueda es un árbol, los algoritmos de búsqueda en árboles y en grafos son equivalentes.

El *Algoritmo Best-First-Search* *[Russel&Norvig, 2022]*
Búsqueda en grafos

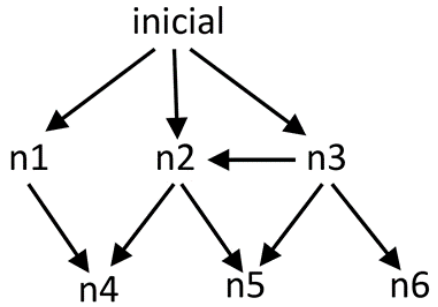
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

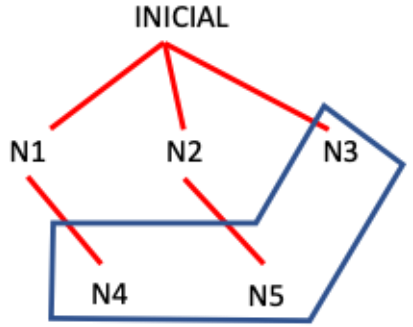
- **reached** es una tabla hash con clave: STATE y contenido: NODE
- Para cada estado visitado, **reached** contiene el nodo con ese estado que representa el mejor camino encontrado hasta el momento desde el INITIAL al estado
- Si en **frontier** hay varios nodos con el mismo estado, el único que “vale” es el que está en **reached**

Búsqueda en Grafos

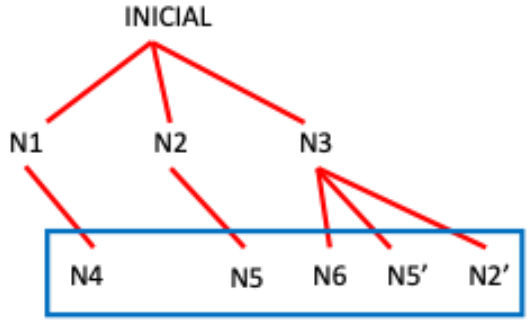
Gestión de estados repetidos. Ejemplo abstracto (I)

- Dados el siguiente espacio de búsqueda y la situación del árbol de búsqueda en una iteración intermedia



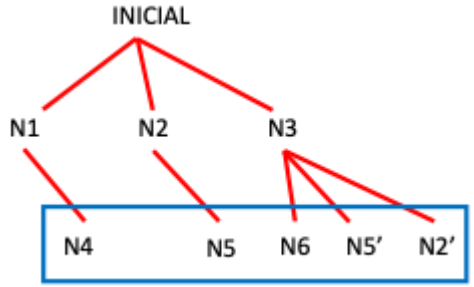


- Si el nodo elegido para expandir es N3 y los nuevos caminos a n2 y n5 son mejores que los anteriores, la nueva situación del árbol de búsqueda es



Búsqueda en Grafos

Gestión de estados repetidos. Ejemplo abstracto (II)



- El estado n6 es nuevo, se crea un nodo N6 y se inserta en **frontier**
- El estado n5 ya está en **frontier**, y por lo tanto en **reached**
  - Se inserta el nodo N5' en **frontier** con el mismo estado que N5 y posiblemente con distinto valor de f()
  - Se cambia el nodo para el estado n5 en **reached**. Ahora es N5'
- El estado n2 está en **reached** y no en **frontier** porque ya fue expandido
  - Se inserta el nodo N2' en **frontier**, y éste pasa a ser el nodo del estado n2 en **reached**
- La rama INICIAL → N2 → N5 debería eliminarse del árbol de búsqueda, pero esto es costoso. N5' es el nodo que "vale" para el estado n5
  - Un buen algoritmo de búsqueda no debería elegir para expandir N5 antes que N5', pero si lo hace el nodo N5 se descarta en ese momento ya que en **reached** está fichado N5' para el estado n5 con un camino al inicial menor que el del inicial a N5

Búsqueda en Grafos

Gestión de estados repetidos. Ejemplo de rutas en Rumanía

- Supongamos que la distancia entre Arad y Sibiu es 400 y que el estado Oradea se desarrolla después de Silbiu

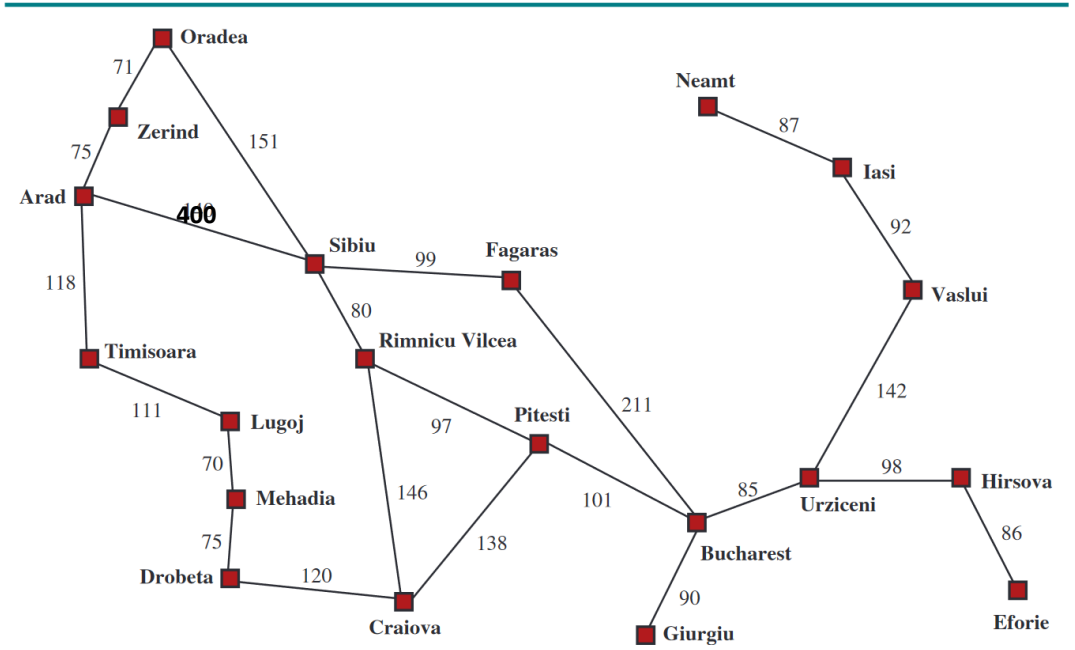


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

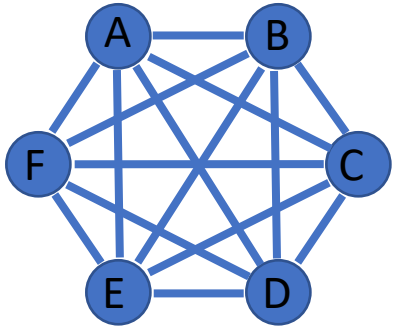
Problemas de Búsqueda

Modelado de espacios de estados

- **El problema del viajante de comercio**
- **El problema de las N-reinas**
- **El problema de los misioneros y caníbales**
- Cálculo de un árbol de expansión mínimo con grado limitado
- El problema de asignación cuadrática
- Coloreado de grafos
- Cálculo de la comunidad más pequeña en una red social
- . . . . .

El problema del viajante de comercio (TSP)
Enunciado

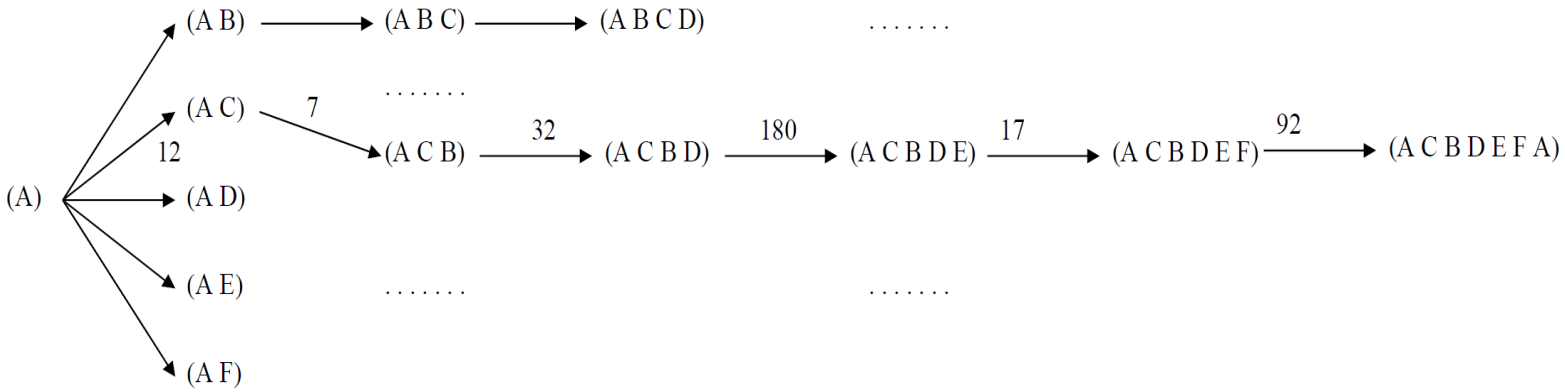
- En el TSP (*Traveling Salesman Problem*), se trata de calcular un recorrido sobre una serie de ciudades, con origen y destino en la ciudad A, visitando cada ciudad una sola vez, y con un coste mínimo
- Ejemplo (TSP simétrico con conexiones entre todas las ciudades)



	A	B	C	D	E	F
A		21	12	15	113	92
B			7	32	25	9
C				5	18	20
D					180	39
E						17

El problema del viajante de comercio (TSP)
Espacio de búsqueda (I)

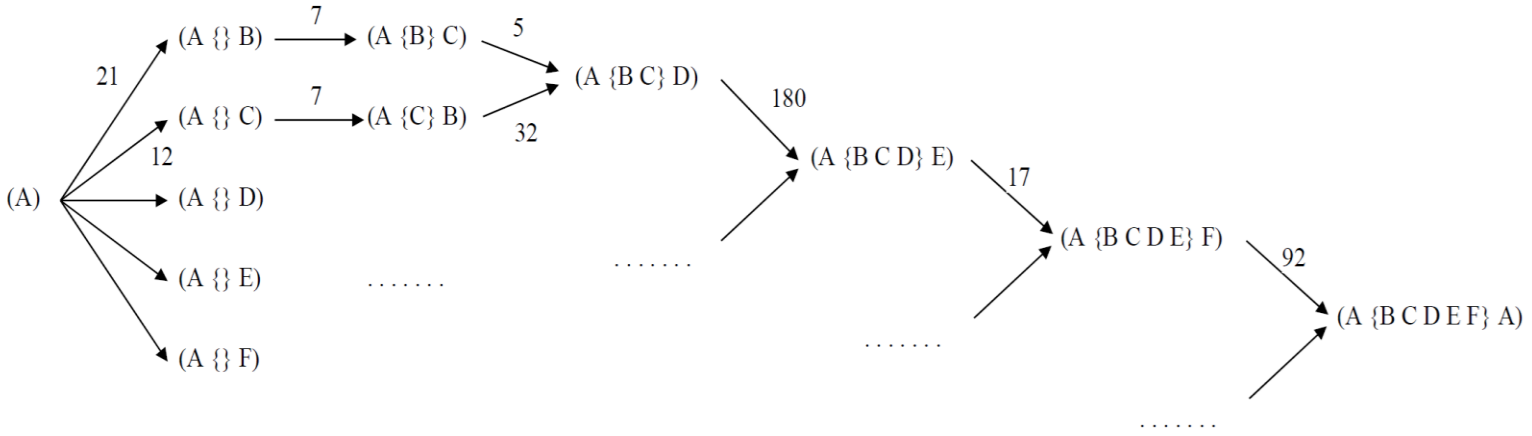
- El espacio de búsqueda se puede representar mediante un árbol



- **Un estado es una lista de ciudades en el orden en que se visitaron**
- Hay un número exponencial (factorial) de estados objetivo. Cada uno representa una solución diferente
- Los subproblemas que representan los estados (ABCD) y (ACBD) en realidad son el mismo

El problema del viajante de comercio (TSP)
Espacio de búsqueda (II)

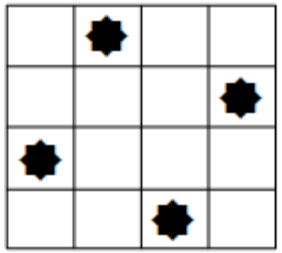
- Si dos estados representan el mismo subproblema, son el mismo estado al que se puede llegar por dos caminos distintos desde el inicial
- Con lo cual, lo razonable es representar el espacio de búsqueda con un grafo, en este caso



- **Un estado es el conjunto de ciudades visitadas y la ciudad actual**
- Solo hay un estado objetivo
- Ahora está claro que una solución es un camino desde el inicial al objetivo
- Cada estado representa un subproblema, pero no la forma de llegar desde el inicial hasta él

El problema de las N-reinas
Espacio de búsqueda

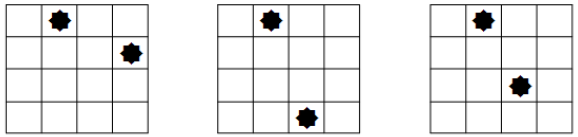
- El problema consiste en situar N reinas en un tablero de N×N de forma que no se ataquen
- Ejemplo con N = 4



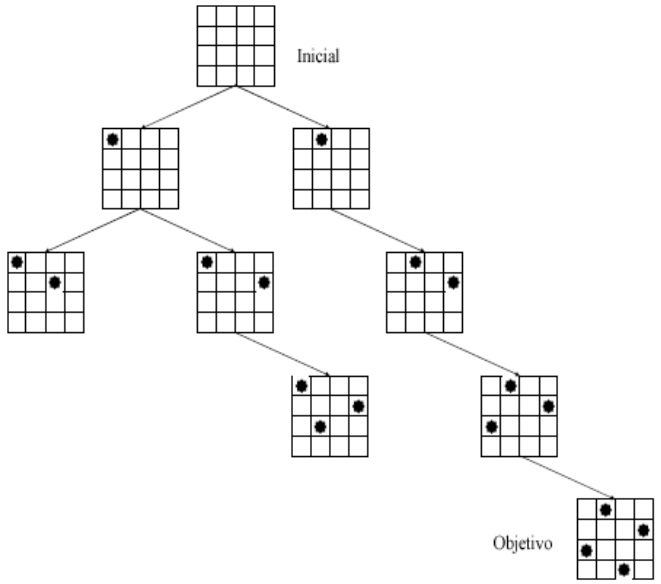
- **Estados**
  Situaciones con *k* reinas en el tablero sin que se ataquen, $0 \le k \le N$
  Inicial $k=0$; objetivos $k = N$ (hay varios)
- **Reglas**
  Colocar una nueva reina sin que se ataque con las que ya están
  Coste = 1
- **Solución**
  Basta con el estado en el que hay *N* reinas, no hace falta el camino

El problema de las N-reinas
Espacio de búsqueda

- Algunos estados posibles, con la interpretación anterior



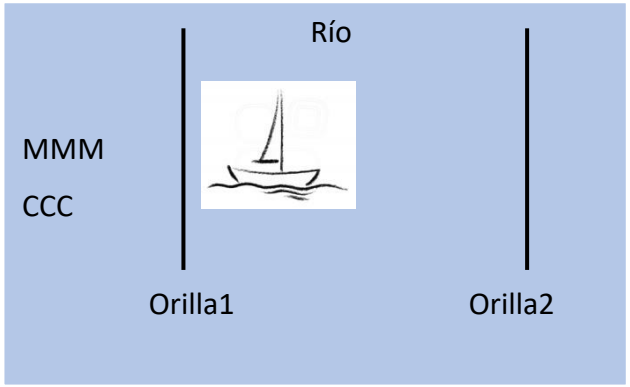
- Parece mejor ir colocando reinas en filas consecutivas y eliminar simetrías



El problema de los misioneros y los caníbales
Enunciado

- Se trata de llevar tres misioneros y tres caníbales desde una orilla del río a la otra en una barca, con las restricciones siguientes:
  - *La barca tiene capacidad para dos personas*
  - *La barca no puede viajar vacía*
  - *No puede haber más caníbales que misioneros en ninguna orilla, salvo que no haya misioneros*

Estado inicial



Estado objetivo