Inteligencia Artificial
Resolver problemas mediante búsqueda

Primavera 2007

profesor: Luigi Ceccaroni

Resolución de problemas

- Se quiere:
  - Resolver automáticamente un problema
- Se necesita:
  - Una representación del problema
  - Algoritmos que usen alguna estrategia para resolver el problema definido en esa representación

Definición de un problema

- Si se abstraen los elementos de un problema se pueden identificar:
  - Un punto de partida
  - Un objetivo a alcanzar
  - Acciones a disposición para resolver el problema
  - Restricciones sobre el objetivo (p.e., de costo)
  - Elementos del dominio que son relevantes en el problema (p.e., conocimiento incompleto del punto de partida)

Representación de problemas

- Existen diferentes formas de representar problemas para resolverlos de manera automática
- Representaciones generales:
  - **Espacio de estados**. Un problema se divide en un conjunto de pasos de resolución desde el inicio hasta el objetivo.
  - **Reducción a sub-problemas**. Un problema se descompone en una jerarquía de sub-problemas.
- Representaciones para problemas específicos:
  - Resolución de juegos
  - Satisfacción de restricciones

Representación de problemas: estados

- Se puede definir un problema por los elementos que intervienen y sus relaciones.
- En cada instante de la resolución de un problema esos elementos tendrán unas características y relaciones específicas.
- Se denomina **estado** a la representación de los elementos que describen el problema en un momento dado.
- Se distinguen dos estados especiales: el **estado inicial** (punto de partida) y el **estado final** (en general, el objetivo del problema).
- ¿Qué descriptores incluir en el estado? Ej.: la localización

Modificación del estado: función sucesor

- Para poder moverse entre los diferentes estados se necesita una función sucesor (descripción de las posibles acciones).
- **Función sucesor** (o conjunto de operadores): función de transformación sobre la representación de un estado que lo convierte en otro estado
- La función sucesor define una relación de accesibilidad entre estados.
- Representación de la función sucesor:
  - Condiciones de aplicabilidad
  - Función de transformación

Espacio de estados

- El conjunto de todos los estados alcanzables desde el estado inicial conforma lo que se denomina **espacio de estados.**
- Representa todos los caminos que hay entre todos los estados posibles de un problema.
- El espacio de estados forma un grafo (o mapa) en el cual los nodos son estados y los arcos son acciones.
- La solución del problema está dentro de ese mapa.

Solución de un problema en el espacio de estados

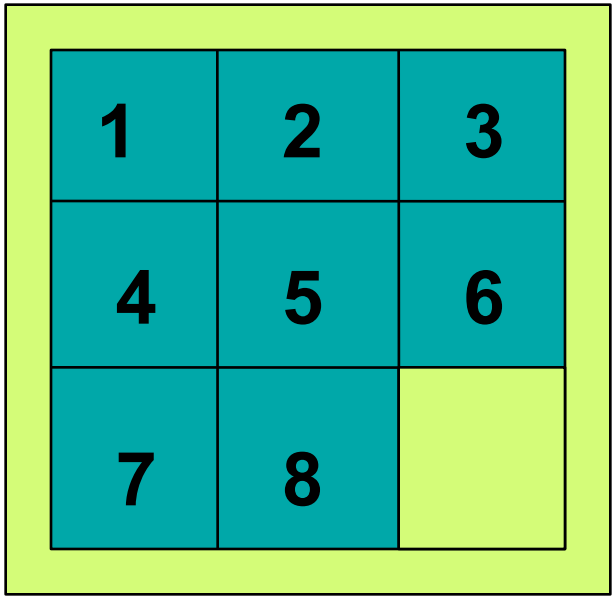
- **Solución:** Secuencia de pasos que llevan del estado inicial al final (secuencia de operadores) o también el estado final
- **Tipos de solución:** una cualquiera, la mejor, todas
- **Costo de una solución:** gasto en recursos de la aplicación de los operadores a los estados; puede ser importante o no según el problema y qué tipo de solución busquemos

Descripción de un problema en el espacio de estados

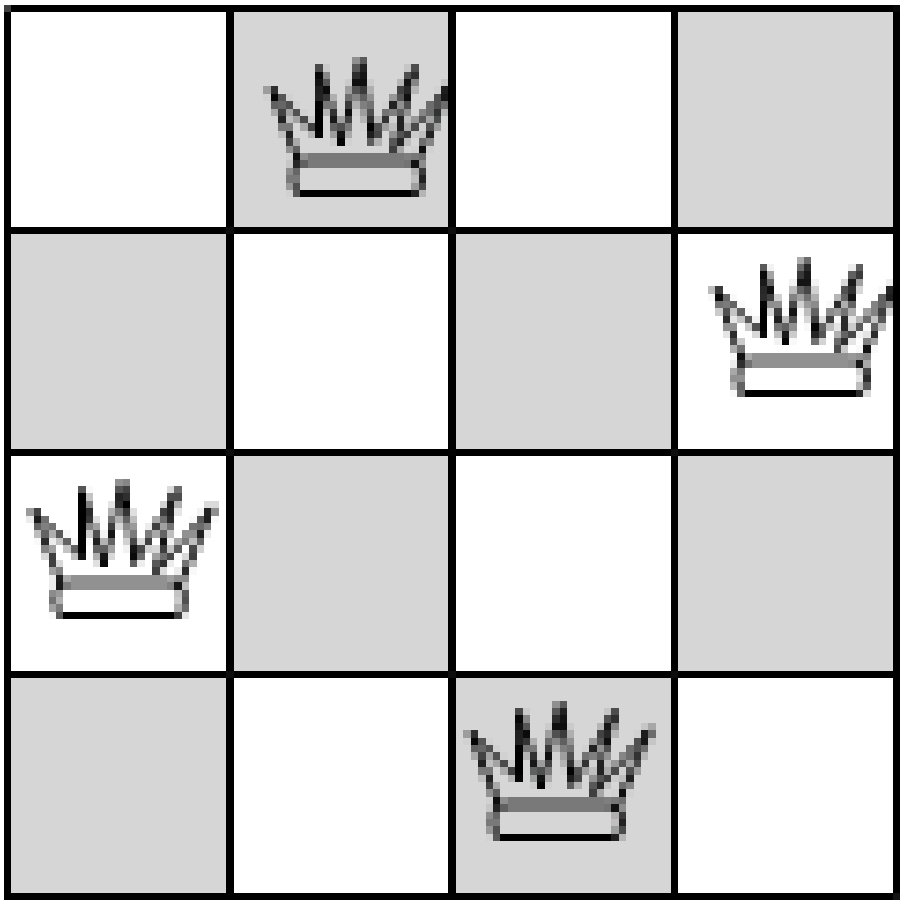
- Definir el espacio de estados (explícita o implícitamente)
- Especificar el estado inicial
- Especificar el estado final o las condiciones que cumple
- Especificar los operadores de cambio de estado (condiciones de aplicabilidad y función de transformación)
- Especificar el tipo de solución:
  - La secuencia de operadores o el estado final
  - Una solución cualquiera, la mejor (definición de costo), todas …

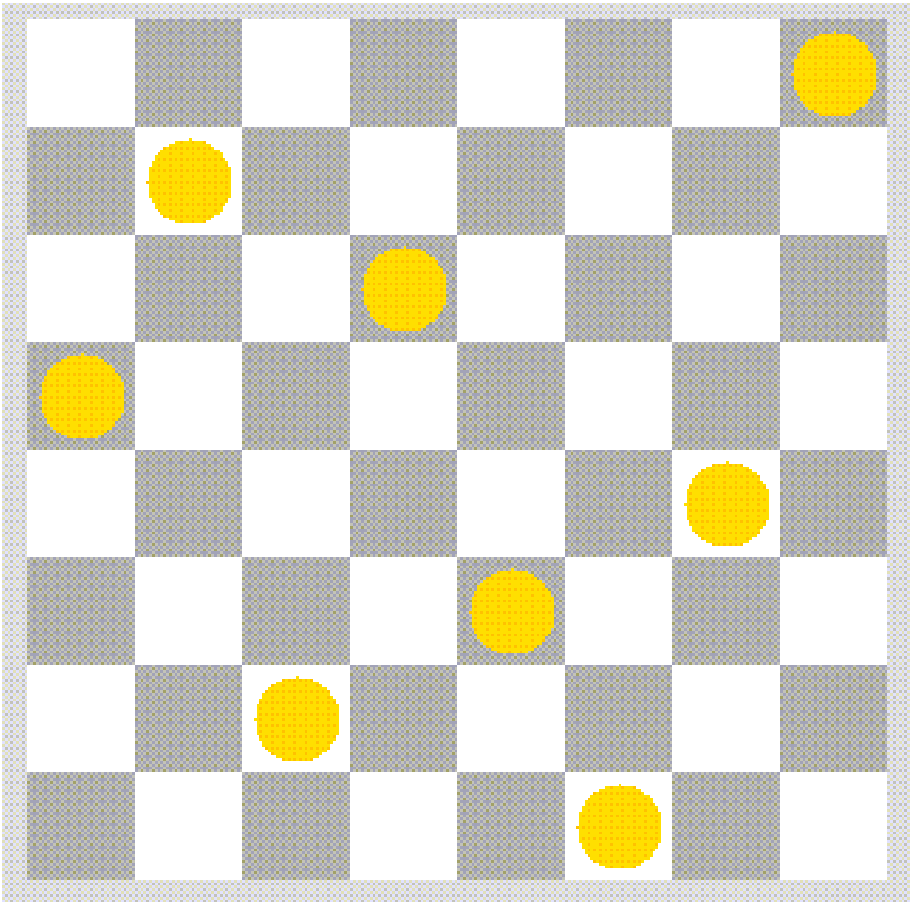
Ejemplo: 8 puzzle

- **Espacio de estados**: configuraciones de 8 fichas en el tablero
- **Estado inicial**: cualquier configuración
- **Estado final**: fichas en orden específico
- **Acción**: “mover hueco”
  - Condiciones: el movimiento está dentro del tablero
  - Transformación: “mover el hueco” a la *Izquierda*, *Derecha*, *Arriba* y *Abajo*
- **Solución**: Qué pasos + El menor número



Ejemplo: *n* reinas
(*n* = 4, n = 8)





Ejemplo: *n* reinas

- **Espacio de estados**: configuraciones de 0 a n reinas en el tablero con sólo una por fila y columna
- **Estado inicial**: configuración sin reinas en el tablero
- **Estado final**: configuración en la que ninguna reina se mata entre sí
- **Operadores**: colocar una reina en una fila y columna
  - Condiciones: la reina no es matada por ninguna ya colocada
  - Transformación: colocar una reina más en el tablero en una fila y columna determinada
- **Solución**: una solución, pero no importan los pasos

Ejercicios en grupos

- Ejercicio en grupos (15 m)
- Los alumnos se reúnen en los grupos base de 4. Cada grupo se divide en dos parejas. Cada pareja resuelve el problema usando un algoritmo de los dos propuestos. (entregable #2.1)
- Reunión de expertos (10 m)
- Explicación a los compañeros del grupo (10 m)
- Conclusión, resumen y planificación actividades fuera de clase (5 m)
- El profesor explica en detalle en que consisten las actividades fuera de clase a llevar a cabo. (entregable #2.2)

Búsqueda en el espacio de estados

- Se define una representación del espacio de estados para poder implementar algoritmos que busquen soluciones.
- La resolución de un problema con esta representación pasa por explorar el espacio de estados.
- Se empieza del estado inicial y se evalúa cada paso hasta encontrar un estado final.
- En el caso peor se exploran todos los posibles caminos entre el estado inicial del problema y el estado final.

Estructura del espacio de estados

- **Estructuras de datos**: árboles y grafos
- **Estados** = nodos
- **Operadores** = arcos entre nodos (dirigidos)
- **Árboles**: sólo un camino lleva a un nodo
- **Grafos**: varios caminos pueden llevar a un nodo

Algoritmo general de búsqueda en árboles (descripción informal)

función Búsqueda-Árboles*(problema,estrategia)* **devuelve** una solución o fallo
  inicializa el árbol de búsqueda usando el estado inicial del *problema*
  bucle hacer
    si no hay candidatos para expandir **entonces devolver** fallo
    escoger, de acuerdo a la estrategia, un nodo hoja para expandir [selección]
    si el nodo contiene un estado objetivo entonces devolver la correspondiente solución
    en otro caso expandir el nodo y añadir los nodos resultado al árbol de búsqueda

- Para grafos el algoritmo es equivalente.
- La estructura se construye a medida que se hace la búsqueda.
- La selección del siguiente nodo determinará el tipo de búsqueda (orden de selección o expansión).
- Es necesario definir un orden entre los sucesores de un nodo (orden de generación).

Algoritmo general: nodos

- Nodos abiertos:
  - Estados generados pero aún no visitados (o explorados)
  - Estados visitados pero aún no expandidos
- Nodos cerrados: estados visitados y que ya se han expandido
- Habrá una estructura para almacenar los nodos abiertos.
- Las diferentes políticas de orden de expansión y de inserción de los nodos generados en la estructura determinan el tipo de búsqueda.
- Si se explora un grafo puede ser necesario tener en cuenta los estados repetidos (esto significa tener una estructura para los nodos cerrados).

Características de los algoritmos

- **Completitud**: ¿encontrará una solución?
- **Complejidad temporal**: ¿cuánto tardará?
- **Complejidad espacial**: ¿cuánta memoria gastará?
- **Optimización**: ¿encontrará la solución óptima?

Algoritmo general de búsqueda (descripción formal)

```
función Búsqueda-Grafos(problema,frontera) devuelve una solución, o fallo
  cerrado ← conjunto vacío
  frontera ← Insertar(Hacer-Nodo(Estado-Inicial[problema]),frontera)
  bucle hacer
    si Vacia?(frontera) entonces devolver fallo
    nodo ← Borrar-Primero(frontera)
    si Test-Objetivo[problema](Estado[nodo]) entonces devolver Solución(nodo)
    si Estado[nodo] no está en cerrado entonces
      añadir Estado[nodo] a cerrado
      frontera ← Insertar-Todo(Expandir(nodo,problema),frontera)
```

- Variando la estructura de abiertos varía el comportamiento del algoritmo (orden de visita de los nodos).
- La función Expandir sigue el orden de generación de sucesores definido en el problema (si está definido).

Tipos de algoritmos

- Algoritmos de búsqueda ciega (o no informada)
  - No tienen en cuenta el coste de la solución en la búsqueda
  - Su funcionamiento es sistemático, siguen un orden de visitas y generación de nodos establecido por la estructura del espacio de búsqueda
  - Ejemplos:
    - primero en anchura
    - primero en profundidad
    - primero en profundidad con profundidad iterativa

Tipos de algoritmos

- Algoritmos de búsqueda heurística (o informada)
  - Utilizan una estimación del coste de la solución para guiar la búsqueda
  - No siempre garantizan el óptimo, ni una solución
  - Ejemplos:
    - ascensión de colinas,
    - primero el mejor,
    - A*,
    - A*PI

Búsqueda primero en anchura

No expandir nodos de nivel *n* hasta que todos los nodos de nivel *n-1* han sido expandidos

- Los nodos se visitan y generan por niveles.
- La estructura para los nodos abiertos es una cola (FIFO).
- Un nodo es visitado cuando todos los nodos de los niveles superiores y sus hermanos precedentes han sido visitados.
- Características:
  - **Completitud**: el algoritmo siempre encuentra una solución.
  - **Complejidad temporal**: exponencial respecto a la profundidad de la solución $O(r^{p+1})$.
  - **Complejidad espacial**: exponencial respecto a la profundidad de la solución $O(r^{p+1})$.
  - **Optimización**: la solución que se encuentra es óptima en número de niveles desde la raíz.

Búsqueda primero en profundidad

No expandir nodos de nivel *n* si hay todavía algún nodo de nivel > *n* pendiente de considerar

- Los nodos se visitan y generan buscando los nodos a mayor profundidad y retrocediendo cuando no se encuentran nodos sucesores.
- La estructura para los nodos abiertos es una pila (LIFO).
- Para garantizar que el algoritmo acabe debe (posiblemente) imponerse un límite en la profundidad de exploración.
- Características
  - **Completitud**: si se impone un límite de profundidad, el algoritmo encuentra una solución sólo si ésta existe dentro de ese límite.
  - **Complejidad temporal**: exponencial respecto a la profundidad del límite de exploración $O(r^m)$.
  - **Complejidad espacial**: en el caso de no controlar los nodos repetidos el coste es lineal respecto al factor de ramificación y el límite de profundidad O(r m). Si la implementación es recursiva el coste es O(m).
  - **Optimización**: no se garantiza que la solución sea óptima.

Búsqueda primero en profundidad

```
Algoritmo Búsqueda en profundidad limitada (límite: entero)
  Est_abiertos.insertar(Estado inicial)
  Actual= Est_abiertos.primero()
  Mientras no es_final?(Actual) y no Est_abiertos.vacía?() hacer
    Est_abiertos.borrar_primero()
    Est_cerrados.insertar(Actual)
    si profundidad(Actual) ≤ limite entonces
      Hijos= generar_sucesores(Actual)
      Hijos= tratar_repetidos(Hijos, Est_cerrados, Est_abiertos)
    fsi
    Est_abiertos.insertar(Hijos)
    Actual= Est_abiertos.primero()
  fMientras
fAlgoritmo
```

- La estructura de abiertos es ahora una pila.
- Se dejan de generar sucesores cuando se llega al límite de profundidad.
- Esta modificación garantiza que el algoritmo acaba.
- Si se tratan repetidos el ahorro en espacio es nulo.

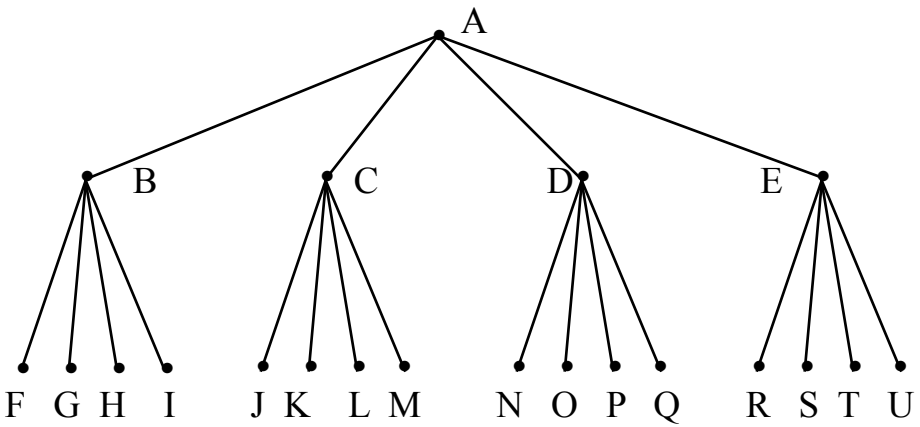
Tratamiento de nodos repetidos

- En anchura
  - Si el repetido está en la estructura de nodos cerrados, el camino actual tendrá una profundidad igual o mayor al repetido cerrado y se puede olvidar.
  - Si el repetido está en la estructura de nodos abiertos, el camino actual tendrá una profundidad igual o mayor al repetido abierto y se puede olvidar.
- En profundidad
  - Si el repetido está en la estructura de nodos cerrados, se guarda el camino actual si tiene una profundidad menor.
  - Si el repetido está en la estructura de nodos abiertos, se puede olvidar el camino actual; seguro que tiene una profundidad mayor.

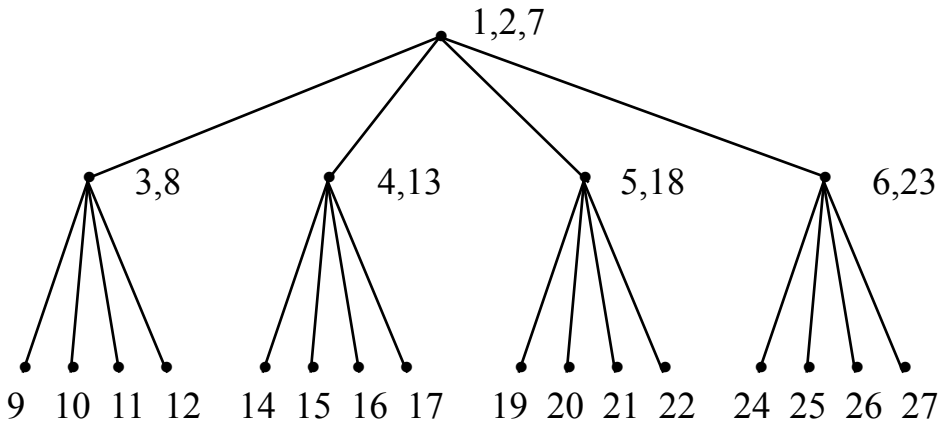
Profundidad iterativa (PI)

- PI combina la complejidad espacial de la búsqueda primero en profundidad con la optimización de la búsqueda primero en anchura.
- El algoritmo consiste en realizar búsquedas en profundidad sucesivas con un nivel de profundidad máximo acotado y creciente en cada iteración.
- Así se consigue el comportamiento de la búsqueda primero en anchura pero sin su coste espacial, ya que la exploración es en profundidad.
- Los nodos se regeneran a cada iteración.
- PI permite evitar los casos en que la búsqueda primero en profundidad no acaba (existen ramas infinitas).
- En la primera iteración la profundidad máxima será 1 y este valor irá aumentando en sucesivas iteraciones hasta llegar a la solución.
- Para garantizar que el algoritmo acabe si no hay solución, se puede definir una cota máxima de profundidad en la exploración.

Profundidad iterativa (PI)



Profundidad iterativa (PI)



Iteración	nodos
1	1
2	2,3,4,5,6
3	7, … 27

Búsqueda en profundidad iterativa

```
Algoritmo Búsqueda en profundidad iterativa (límite: entero)
  prof=1; Est_abiertos.inicializar()
  Mientras no es_final?(Actual) y prof<limite hacer
    Est_abiertos.insertar(Estado inicial)
    Actual= Est_abiertos.primero()
    Mientras no es_final?(Actual) y no Est_abiertos.vacía?() hacer
      Est_abiertos.borrar_primero()
      Est_cerrados.insertar(Actual)
      si profundidad(Actual) ≤ prof entonces
        Hijos= generar_sucesores(Actual)
        Hijos= tratar_repetidos(Hijos, Est_cerrados, Est_abiertos)
      fsi
      Est_abiertos.insertar(Hijos)
      Actual= Est_abiertos.primero()
    fMientras
   prof=prof+1
   Est_abiertos.inicializar()
  fMientras
fAlgoritmo
```

Profundidad Iterativa

- **Completitud**: el algoritmo siempre encontrará la solución.
- **Complejidad temporal**: la misma que la búsqueda en anchura. El regenerar el árbol en cada iteración solo añade un factor constante a la función de coste - $O(r^{p+1})$.
- **Complejidad espacial**: igual que en la búsqueda en profundidad - O(r m).
- **Optimización**: la solución es óptima igual que en la búsqueda en anchura.