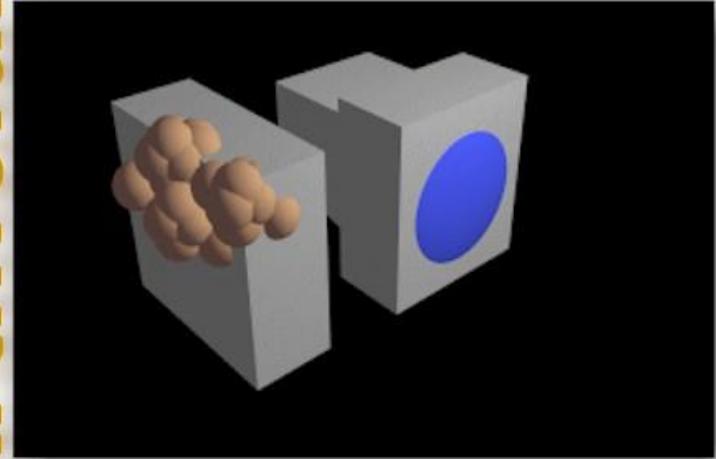
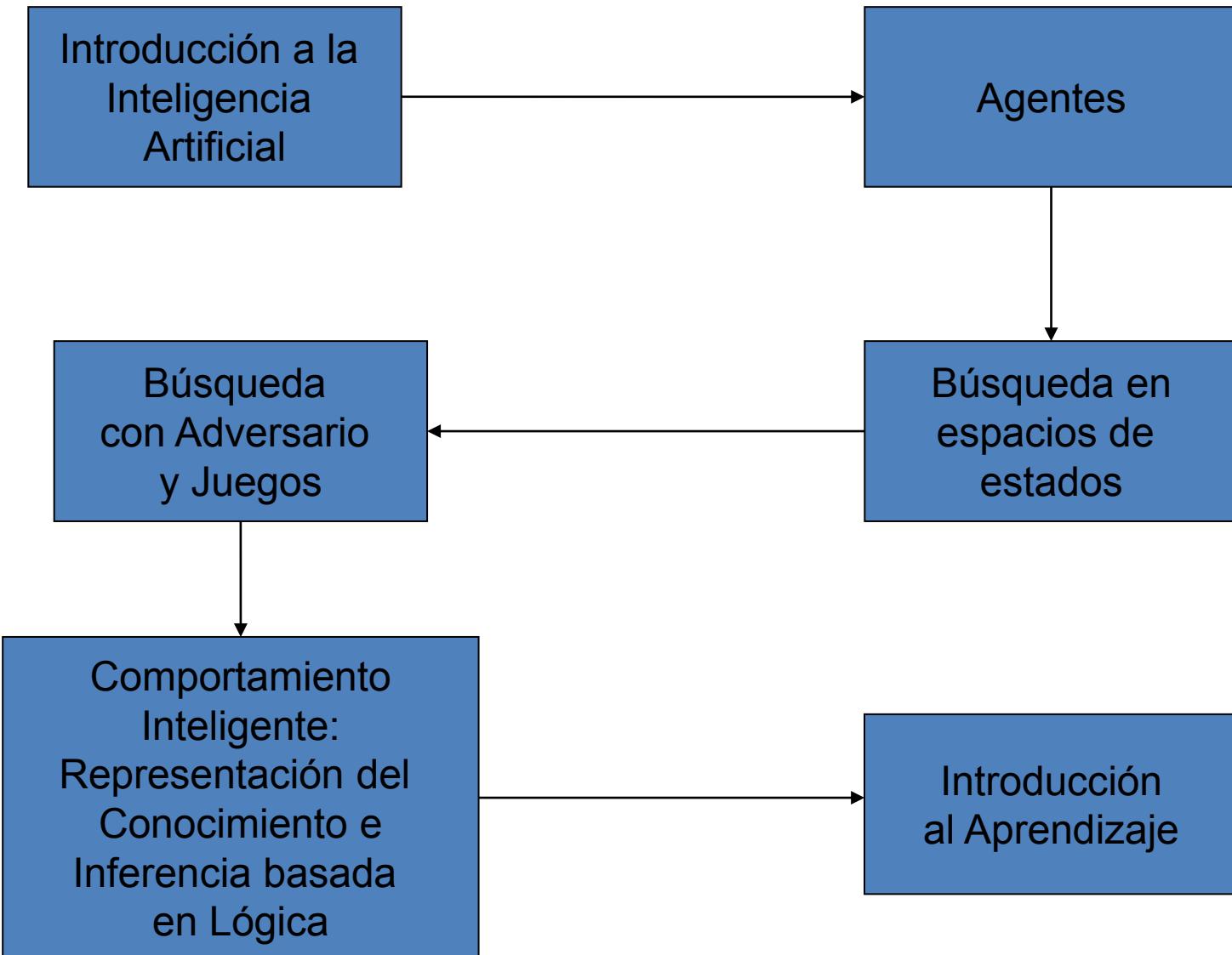


Tema 3: Búsqueda en Espacios de Estados



**Inteligencia
Artificial**





Objetivos

- Entender que la resolución de problemas en IA implica definir una representación del problema y un proceso de búsqueda de la solución.
- Conocer la representación de problemas basados en estados (estado inicial, objetivo y espacio de búsqueda) para ser resueltos con técnicas computacionales.
- Conocer las técnicas más representativas de búsqueda no informada en un espacio de estados (en profundidad, en anchura y sus variantes), y saber analizar su eficiencia en tiempo y espacio.
- Analizar las características de un problema dado y determinar si es susceptible de ser resuelto mediante técnicas de búsqueda. Decidir en base a criterios racionales la técnica más apropiada para resolverlo y saber aplicarla.
- Entender el concepto de heurística y analizar las repercusiones en la eficiencia en tiempo y espacio de los algoritmos de búsqueda.
- Conocer las técnicas más representativas de búsqueda informada en un espacio de estados (búsqueda local, algoritmo A*).

Estudia el tema en ...

- Nils J. Nilsson, “*Inteligencia Artificial: Una nueva síntesis*”, Ed. McGraw Hill, 2000. pp. 53-62, 125-146, 163-174
- S. Russell, P. Norvig, *Artificial Intelligence: A modern Approach*, Tercera Edición, Ed. Pearson, 2010.

Contenido

- Diseño de un agente deliberativo: búsqueda
- Sistemas de búsqueda y estrategias
- Búsqueda sin información
- Búsqueda con información
- Problemas descomponibles y búsqueda

Contenido

- **Diseño de un agente deliberativo:** búsqueda
- **Sistemas de búsqueda y estrategias:** estrategias irrevocables, retroactivas (backtracking), búsqueda en grafos
- **Búsqueda sin información:** Búsqueda en anchura, Búsqueda en profundidad, Búsqueda con costo, Descenso Iterativo
- **Búsqueda con información:** Heurísticas, métodos de escalada, enfriamiento simulado algoritmos genéticos, búsqueda primer mejor greedy, algoritmo A*, búsqueda dirigida
- **Problemas descomponibles y búsqueda:** grafos Y/O

Contenido

- **Diseño de un agente deliberativo: búsqueda**
- Sistemas de búsqueda y estrategias
- Búsqueda sin información
- Búsqueda con información
- Problemas descomponibles y búsqueda

Diseño de un agente deliberativo

- El agente dispone de un modelo del mundo en el que habita
- El agente dispone de un modelo de los efectos de sus acciones sobre el mundo
- El agente es capaz de razonar sobre esos modelos para decidir qué hacer para conseguir un objetivo

La búsqueda en un espacio de estados

- **Espacio de estados** – Representación del conocimiento a través de las acciones del agente
- **Búsqueda en el espacio de estados** – Resolución del problema mediante proyección de las distintas acciones

El mundo de bloques

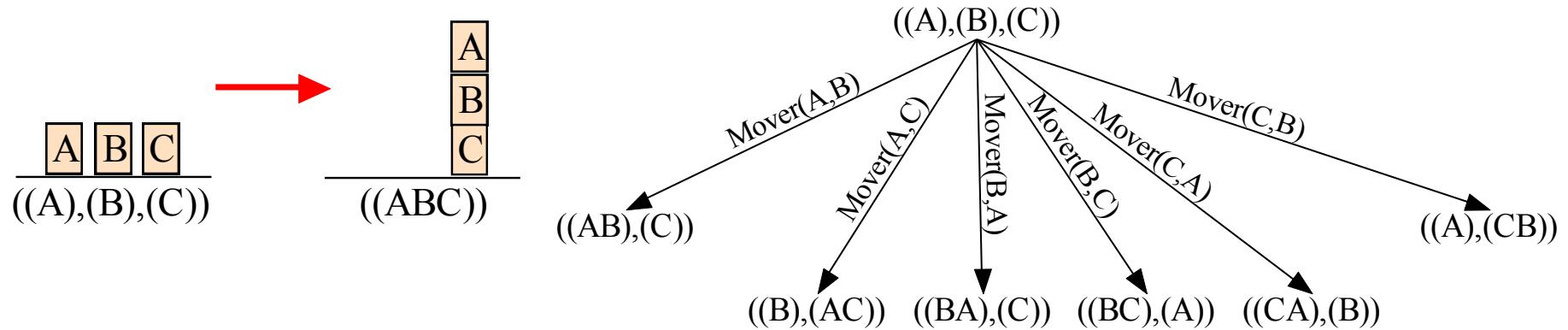
- Supongamos un mundo cuadriculado con 3 bloques A, B, C.
- Inicialmente, todos los bloques están en el suelo.
- El objetivo es apilar los bloques de modo que A quede sobre B, B quede sobre C, y C esté en el suelo.
- En cada momento, se dispone de la operación **mover(x,y)** para poner x sobre y, donde x={A, B, C} e y={A, B, C, Suelo}.
- En cada momento, se conoce el estado del sistema. Lo modelamos con una secuencia de listas de objetos sobre objetos. Inicialmente, el estado es **((A), (B), (C))** y se desea llegar al estado **((ABC))**.
- Asumimos que se descartan los **operadores imposibles mover(A,A), mover(B,B), mover(C,C), etc.**, para cada estado.

Descripción de un problema

- Estado inicial
- Estados
- Acciones
- Objetivo
- Costo de las acciones

El mundo de bloques

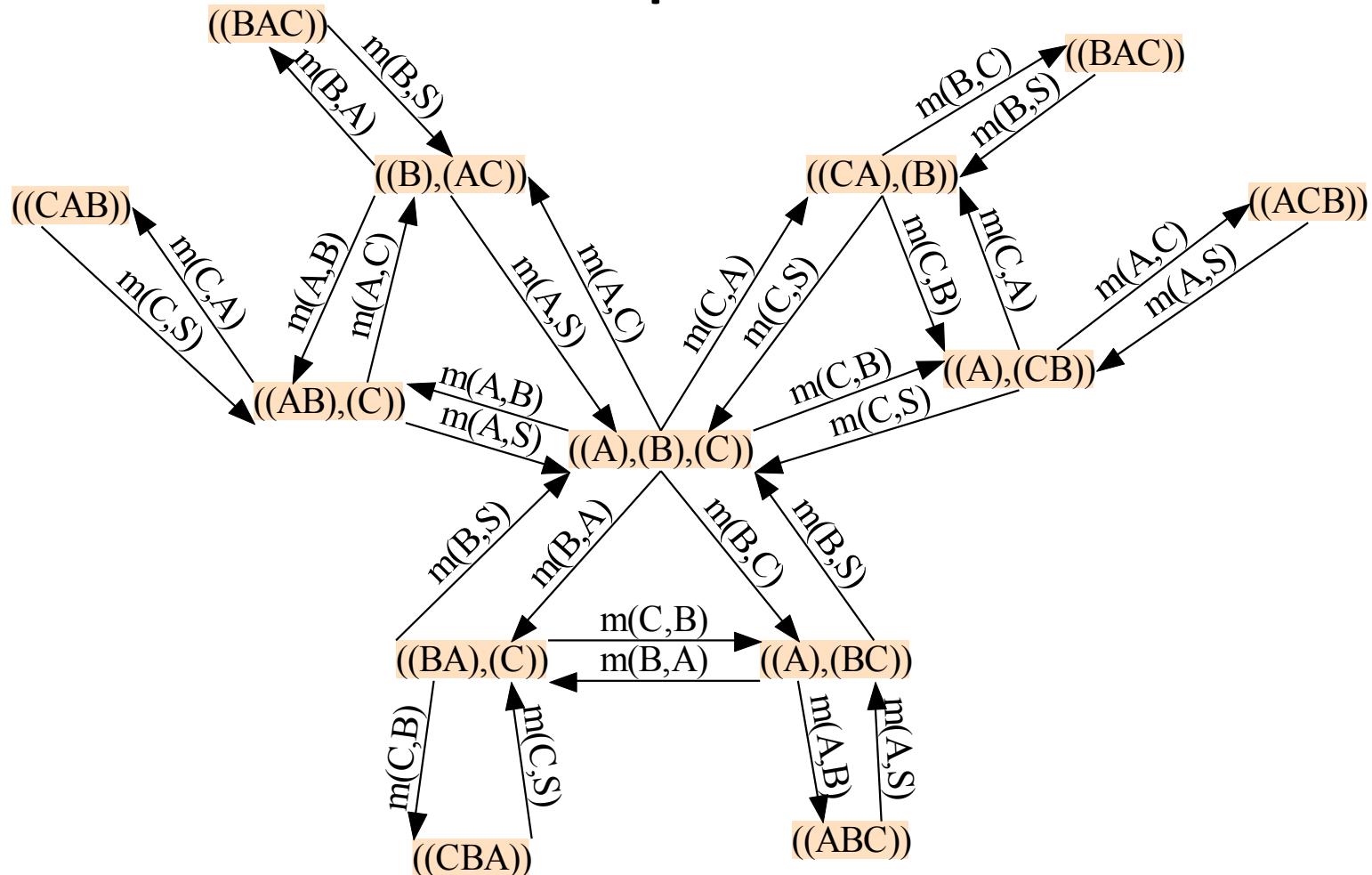
- Una estructura de **grafo dirigido** puede ser útil para buscar secuencias de acciones que nos lleven al objetivo final.
- En esta estructura, **un nodo representa un estado del sistema** y **un arco una posible acción**. La acción, aplicada al estado que representa al nodo origen, producirá el estado del nodo destino.
- Se denomina **grafo de estados**.



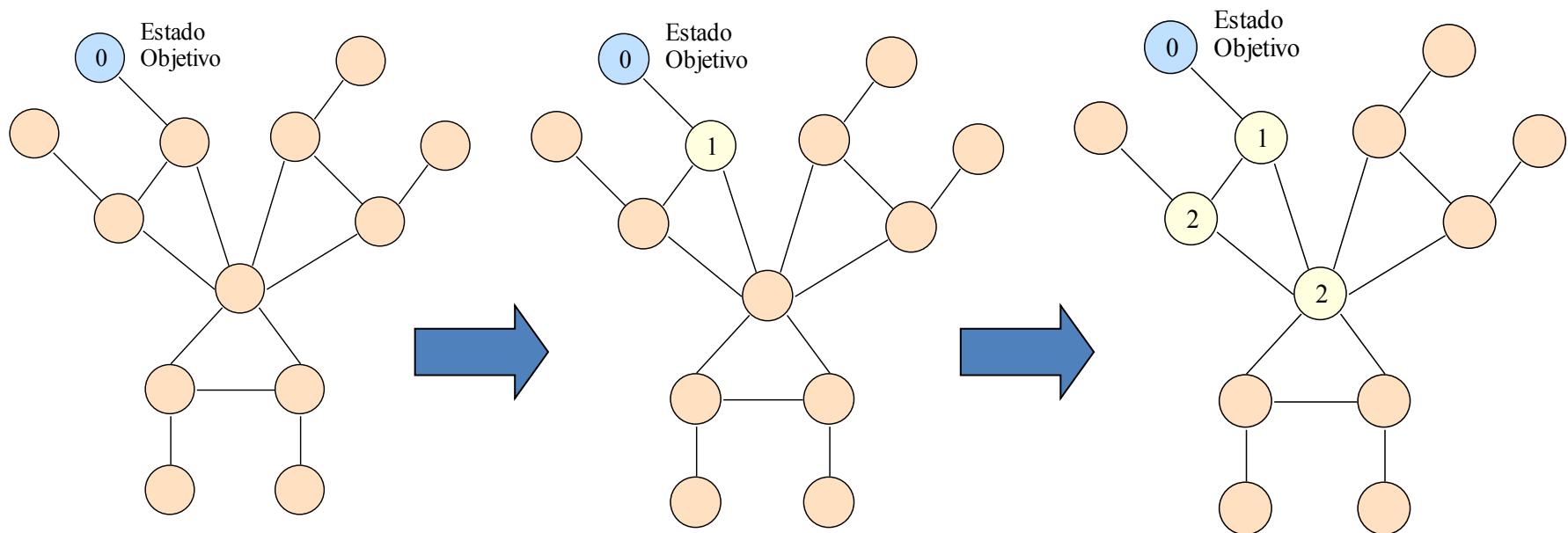
El mundo de bloques

- A la secuencia de acciones que lleva al agente desde un **estado inicial** hasta un **estado destino** se denomina **plan**.
- La búsqueda de dicha secuencia se denomina **planificación**.
 - Grafos explícitos. → *Viajante de comercio con 5 ciudades*
 - Grafos implícitos. → *Problemas con una gran cantidad de estados generan un grafo imposible de manejar*

Espacio de estados en el mundo de bloques

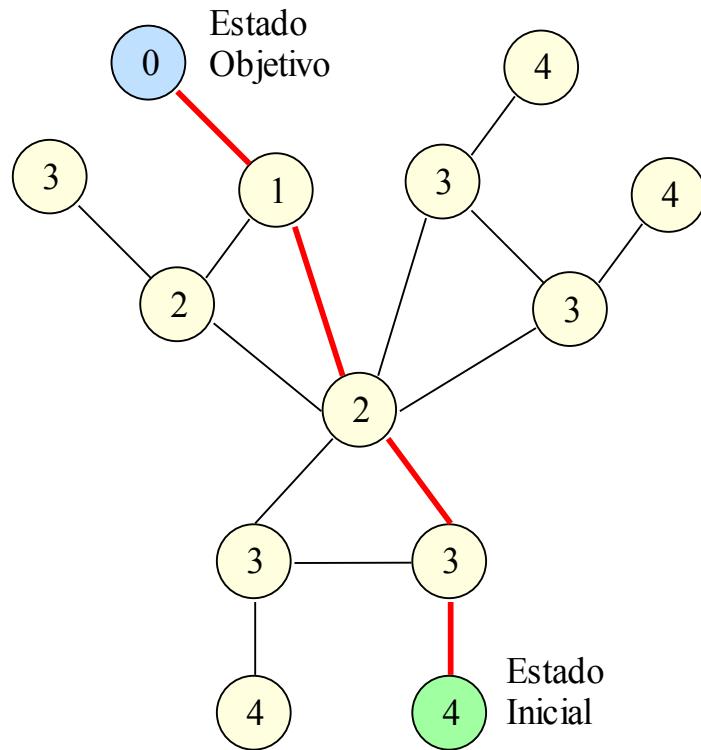


Búsqueda



Búsqueda

- Ejemplo de planificación en el mundo de los bloques:
Planificación de acciones.



Estado Inicial: **((ABC))**

Acción 1: **Mover(A, Suelo)**

Acción 2: **Mover(B, Suelo)**

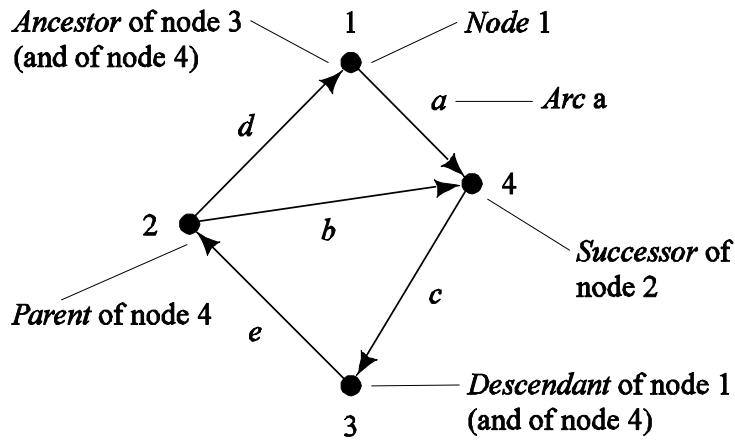
Acción 3: **Mover(A, C)**

Acción 4: **Mover(B, A)**

Estado objetivo alcanzado: **((BAC))**

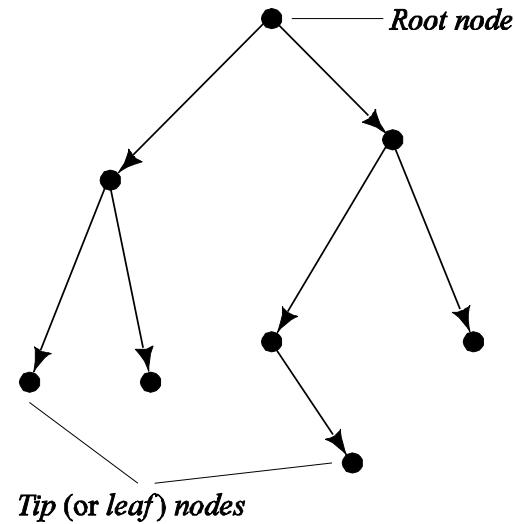
Búsqueda en grafos/árboles

Graph notation



$c(a)$, alternatively $c(1, 4)$, is the *cost* of arc *a*
 (d, a) , alternatively $(2, 1, 4)$, is a *path* from node 2 to node 4

Tree notation



© 1998 Morgan Kaufman Publishers

Parámetros importantes:
Factor de ramificación
Profundidad del árbol de búsqueda

Búsqueda en grafos/árboles

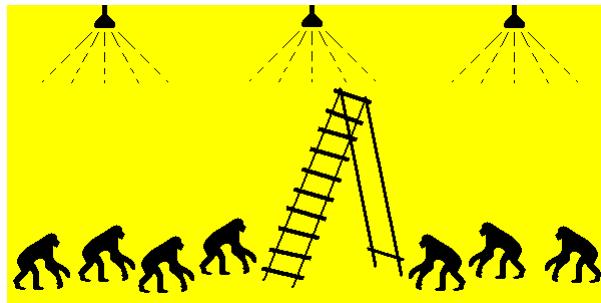
```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier



---


function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
            only if not in the frontier or explored set
```

Ejemplo de agente deliberativo: Problema del mono y los plátanos

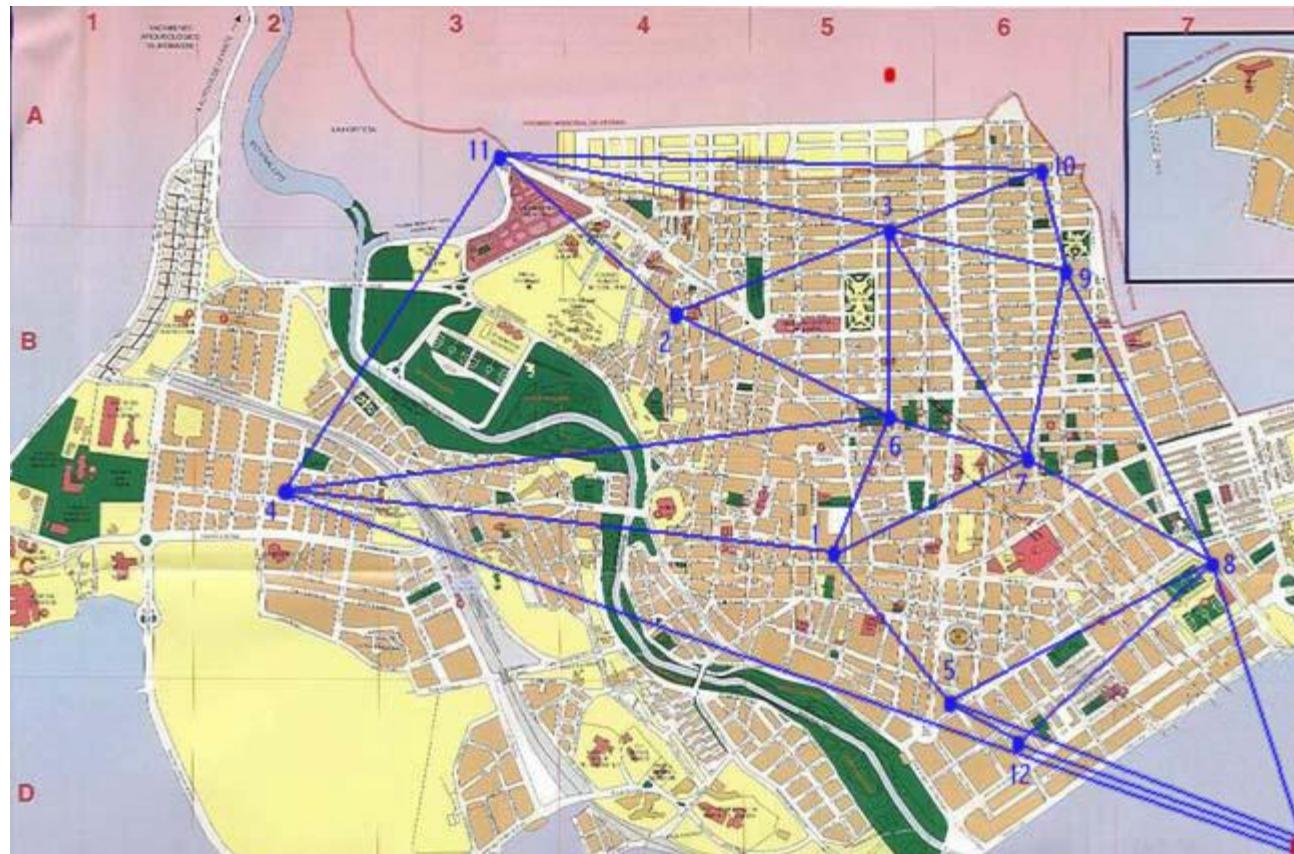


- “Un mono está en la puerta de una habitación. En el centro de la habitación hay un plátano colgado del techo, pero no puede alcanzarle desde el suelo. En la ventana de la habitación hay una caja, que el mono puede mover y a la que puede encaramarse para alcanzar el plátano.”
- El mono puede realizar las siguientes acciones: *desplazarse de la puerta al centro, del centro a la ventana y viceversa; empujar la caja a la vez que se desplaza; subirse y bajarse de la caja; coger el plátano.*
- El problema consiste en encontrar una secuencia de acciones que permita al mono coger el plátano.”

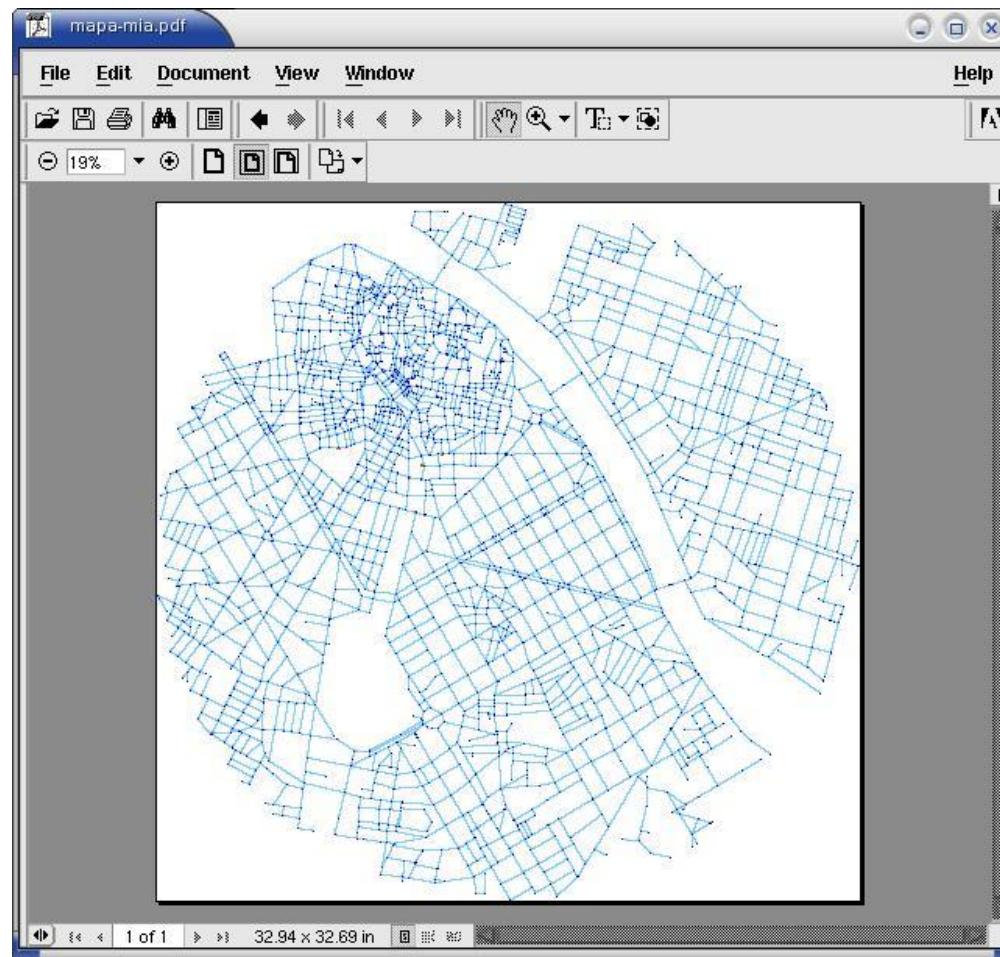
Problema del mono y los plátanos

- Como estado se puede utilizar una lista con cuatro elementos (X, Y, W, Z) donde:
 - X : posición del mono en la habitación (puerta, centro, ventana).
 - Y : situación del mono respecto a la caja (suelo, caja).
 - W : posición de la caja en la habitación (puerta, centro, ventana).
 - Z : posesión del plátano (tiene, no_tiene).
- Para describir todas las posibles acciones del mono, se necesitan seis operadores: andar, empujar, subir, bajar, coger y soltar. Sin embargo, para el problema que nos ocupa, son suficientes cuatro operadores: andar, empujar, subir y coger; nos limitaremos a estos 4 operadores .
- El operador andar se puede definir como:
 - $(X, \text{suelo}, W, Z) \longrightarrow (Y, \text{suelo}, W, Z)$
- para indicar que el mono se desplaza de la posición X a la posición Y .
- El estado se puede representar por la estructura $\text{estado}(X, Y, W, Z)$, que simplemente refleja la definición del estado.

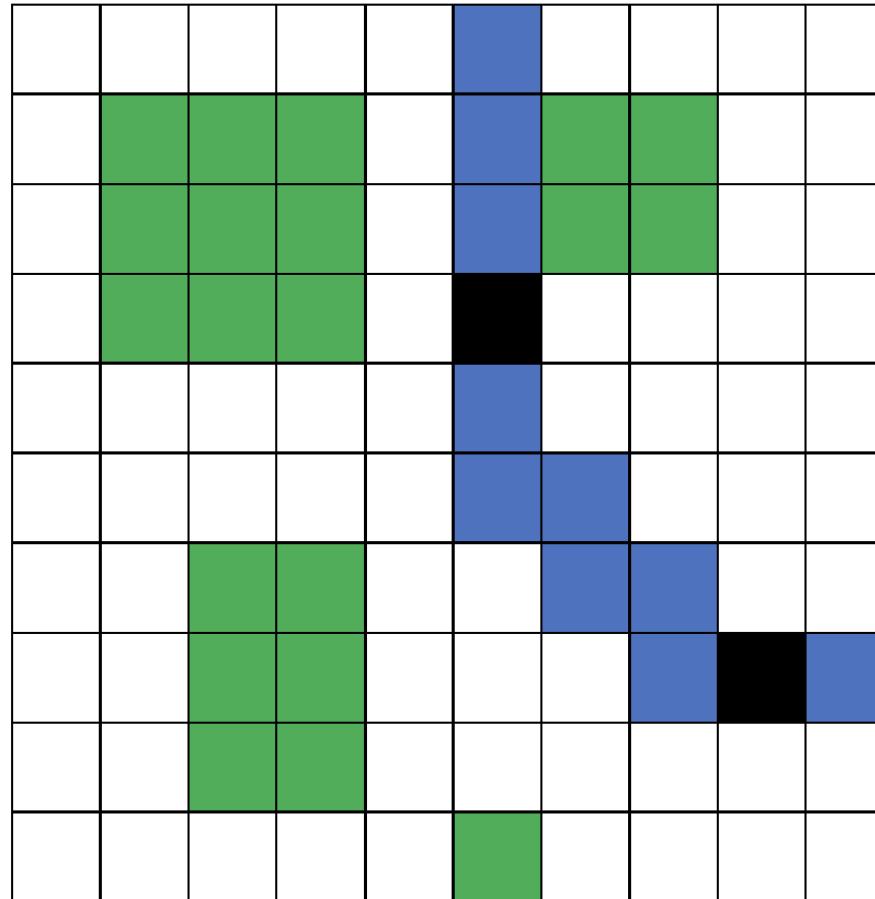
Ejemplo de agente deliberativo: Problema del viajante de comercio



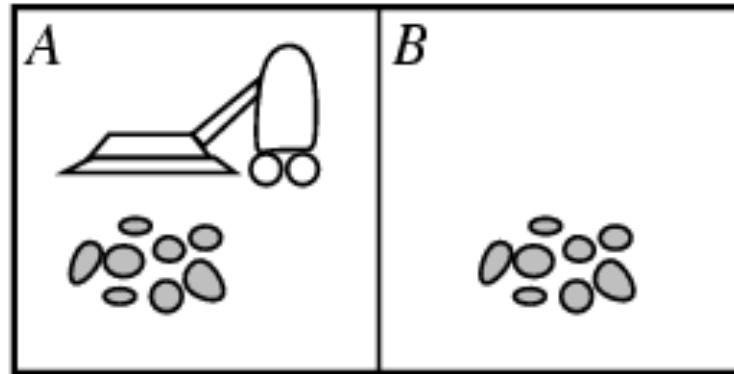
Ejemplo de agente deliberativo: Mapa de Carreteras



Ejemplo de agente deliberativo: Mapa de Carreteras

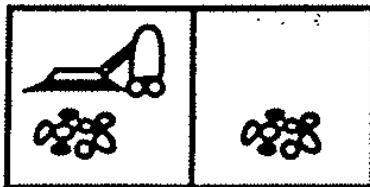


Ejemplo de un agente deliberativo: Problema de la aspiradora

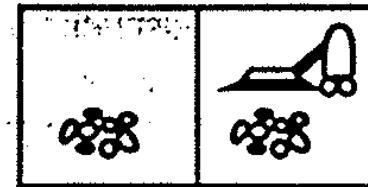


Problema de la aspiradora

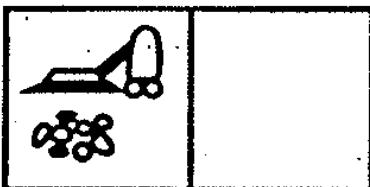
1



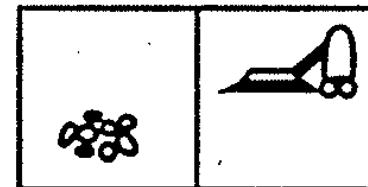
2



3



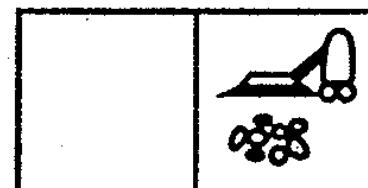
4



5



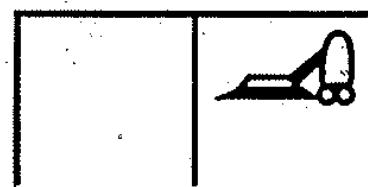
6



7



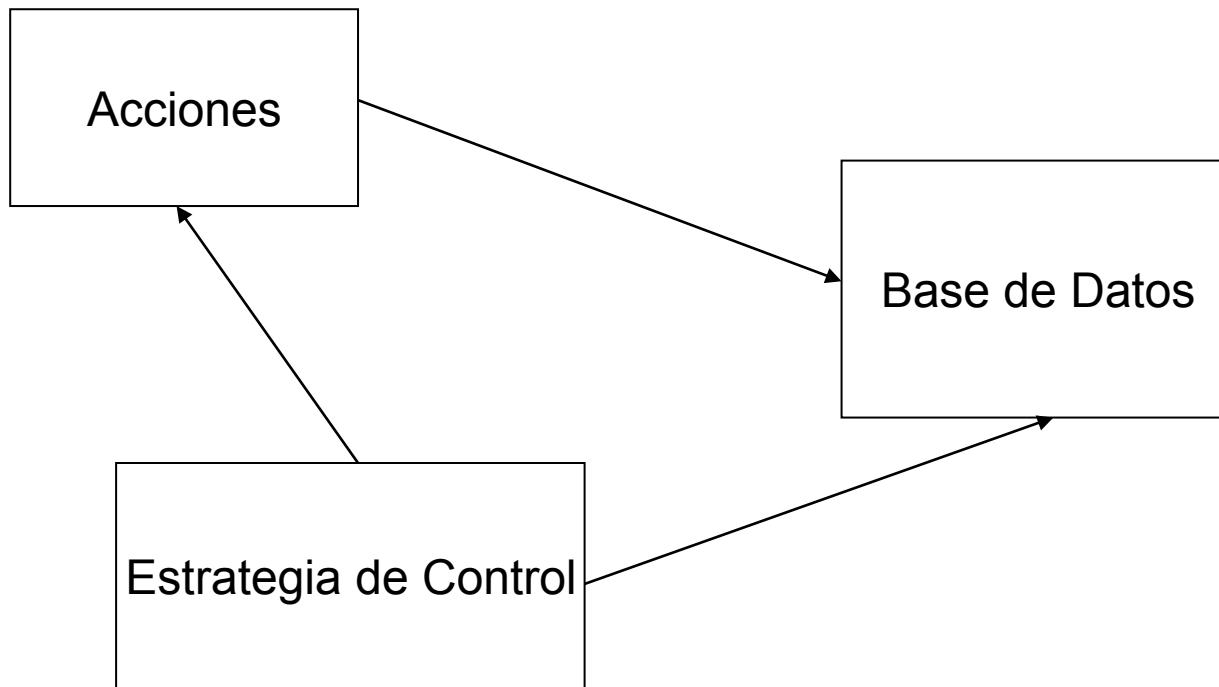
8



Contenido

- Diseño de un agente deliberativo: búsqueda
- **Sistemas de búsqueda y estrategias**
- Búsqueda sin información
- Búsqueda con información
- Problemas descomponibles y búsqueda

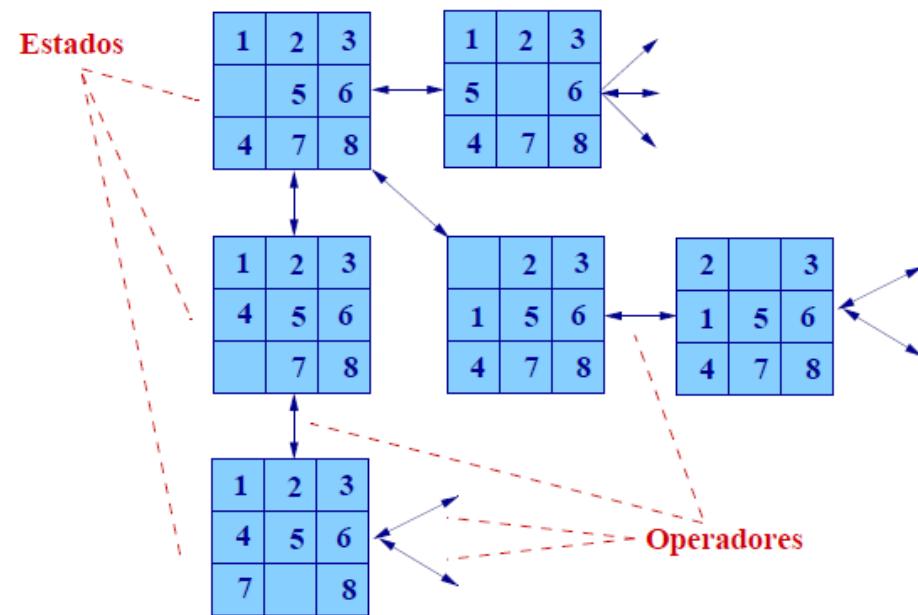
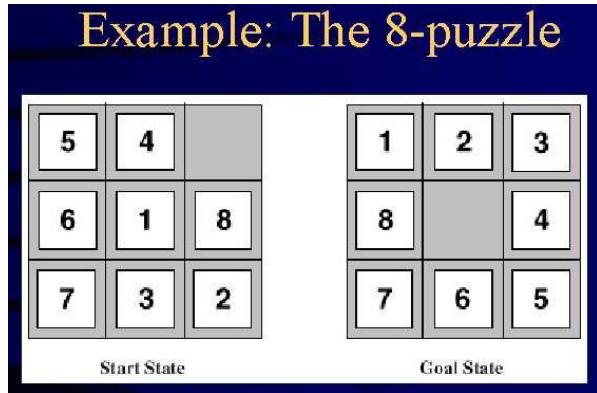
Sistemas de búsqueda y estrategias



Procedimiento Búsqueda

1. DATOS \leftarrow base de datos inicial
2. **until** DATOS satisface la condición de terminación
 do
3. **begin**
4. **select** alguna acción A en el conjunto de acciones que
 pueda ser aplicada a DATOS
5. DATOS \leftarrow resultado de aplicar A a DATOS
6. **end**

Un ejemplo



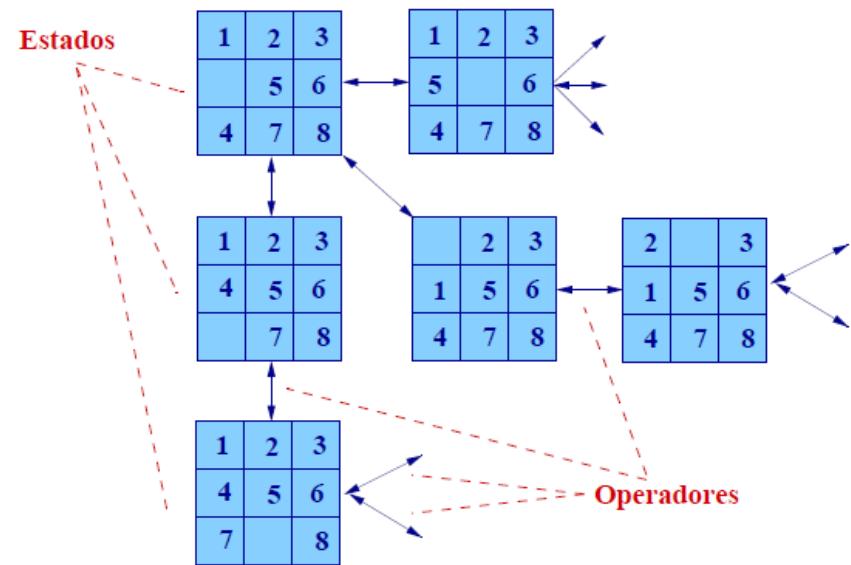
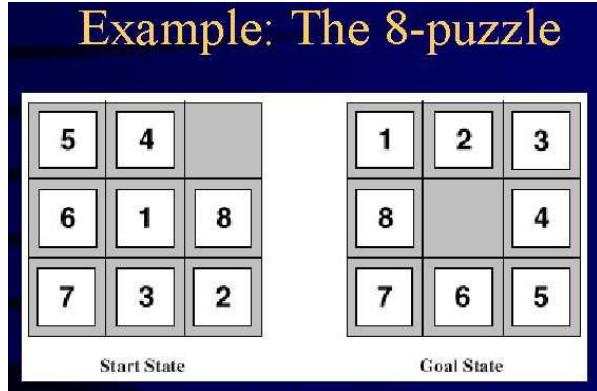
Estrategias de control

- Estrategias irrevocables
- Estrategias tentativas
 - Retroactivas
 - Búsqueda en grafos

Estrategias irrevocables

- En cada momento, el grafo explícito lo constituye un único nodo, que incluye la descripción completa del sistema en ese momento:
 - Se selecciona una acción A.
 - Se aplica sobre el estado del sistema E, para obtener el nuevo estado $E' = A(E)$.
 - Se borra de memoria E y se sustituye por E' .

Un ejemplo



Estrategia retroactiva (Backtracking)

- En memoria sólo guardamos un hijo de cada estado; esto es, se mantiene el camino desde el estado inicial hasta el actual.
- El grafo explícito, por tanto, es realmente una lista.
- ¿Cuando para el proceso? Cuando hemos llegado al objetivo y no deseamos encontrar más soluciones, o bien no hay más operadores aplicables al nodo raíz.

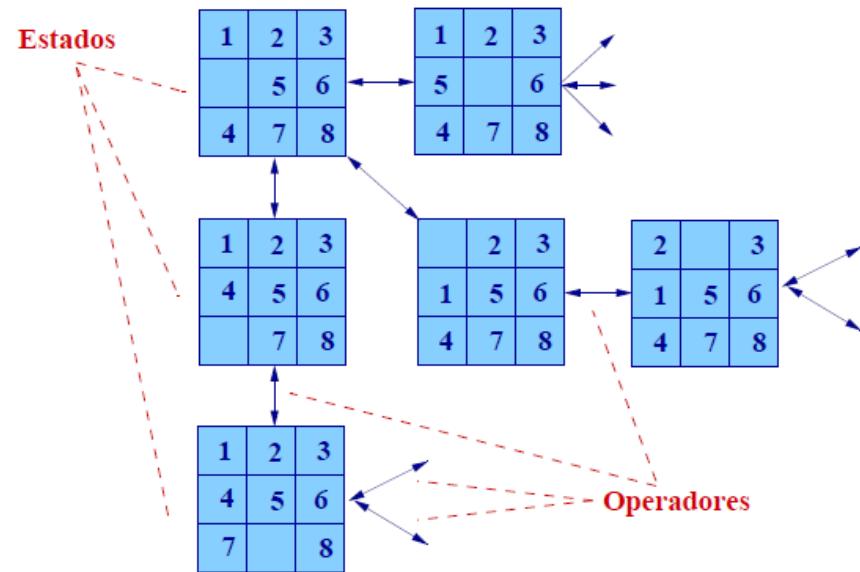
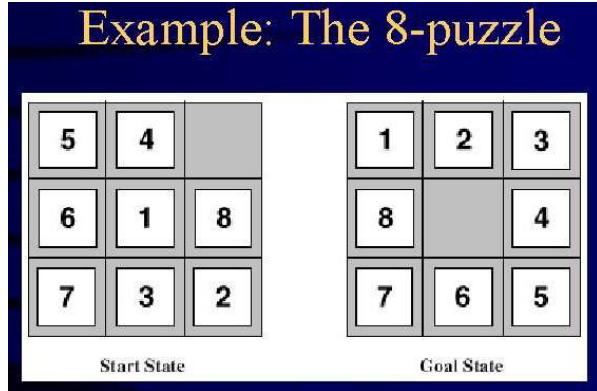
Estrategia retroactiva (Backtracking)

- ¿Cuando se produce una vuelta atrás (o retroceso)?
 - Cuando se ha encontrado una solución, pero deseamos encontrar otra solución alternativa.
 - Cuando se ha llegado a un límite en el nivel de profundidad explorado o el tiempo de exploración en una misma rama.
 - Cuando se ha generado un estado que ya existía en el camino.
 - Cuando no existen reglas aplicables al último nodo de la lista (último nodo del grafo explícito).

Búsqueda en grafos

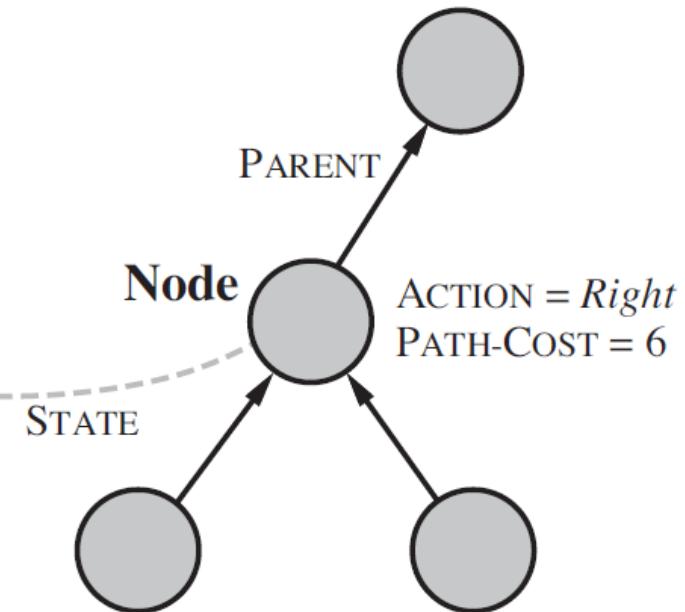
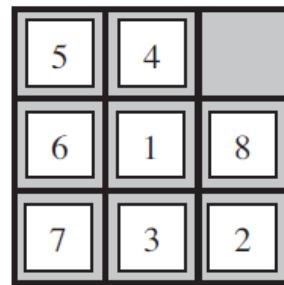
- En memoria se guardan todos los estados (o nodos generados hasta el momento), de forma que la búsqueda puede proseguir por cualquiera de ellos:
 1. Seleccionar un estado E del grafo.
 2. Seleccionar un operador A aplicable sobre E .
 3. Aplicar A , para obtener un nuevo nodo $A(E)$.
 4. Añadir el arco $E \rightarrow A(E)$ al grafo
 5. Repetir el proceso.

Un ejemplo



Infraestructura para los algoritmos de búsqueda

n.STATE
n.PARENT
n.ACTION
n.PATH-COST



Infraestructura para los algoritmos de búsqueda

```
function CHILD-NODE(problem, parent, action) returns a node  
    return a node with  
        STATE = problem.RESULT(parent.STATE, action),  
        PARENT = parent, ACTION = action,  
        PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

Medidas del comportamiento de un sistema de búsqueda

- **Completitud:** hay garantía de encontrar la solución si esta 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?

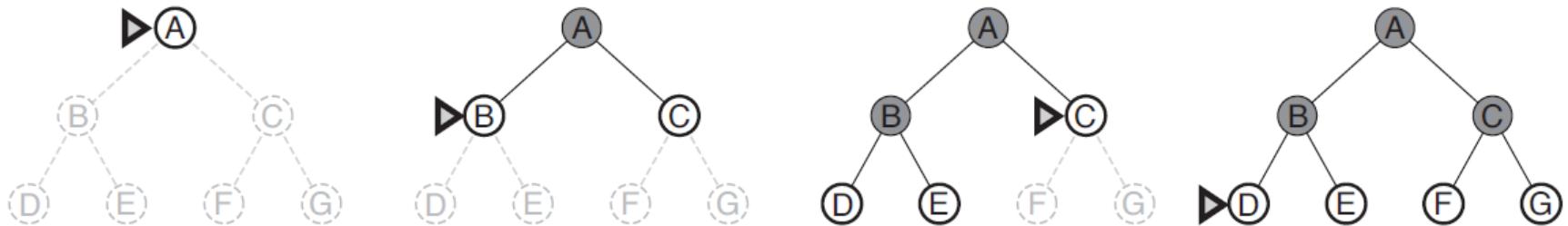
Contenido

- Diseño de un agente deliberativo: búsqueda
- Sistemas de búsqueda y estrategias
- **Búsqueda sin información**
- Búsqueda con información
- Problemas descomponibles y búsqueda

Búsqueda sin información

- Búsqueda en anchura
- Búsqueda en profundidad
- Búsqueda con costo
- Descenso Iterativo

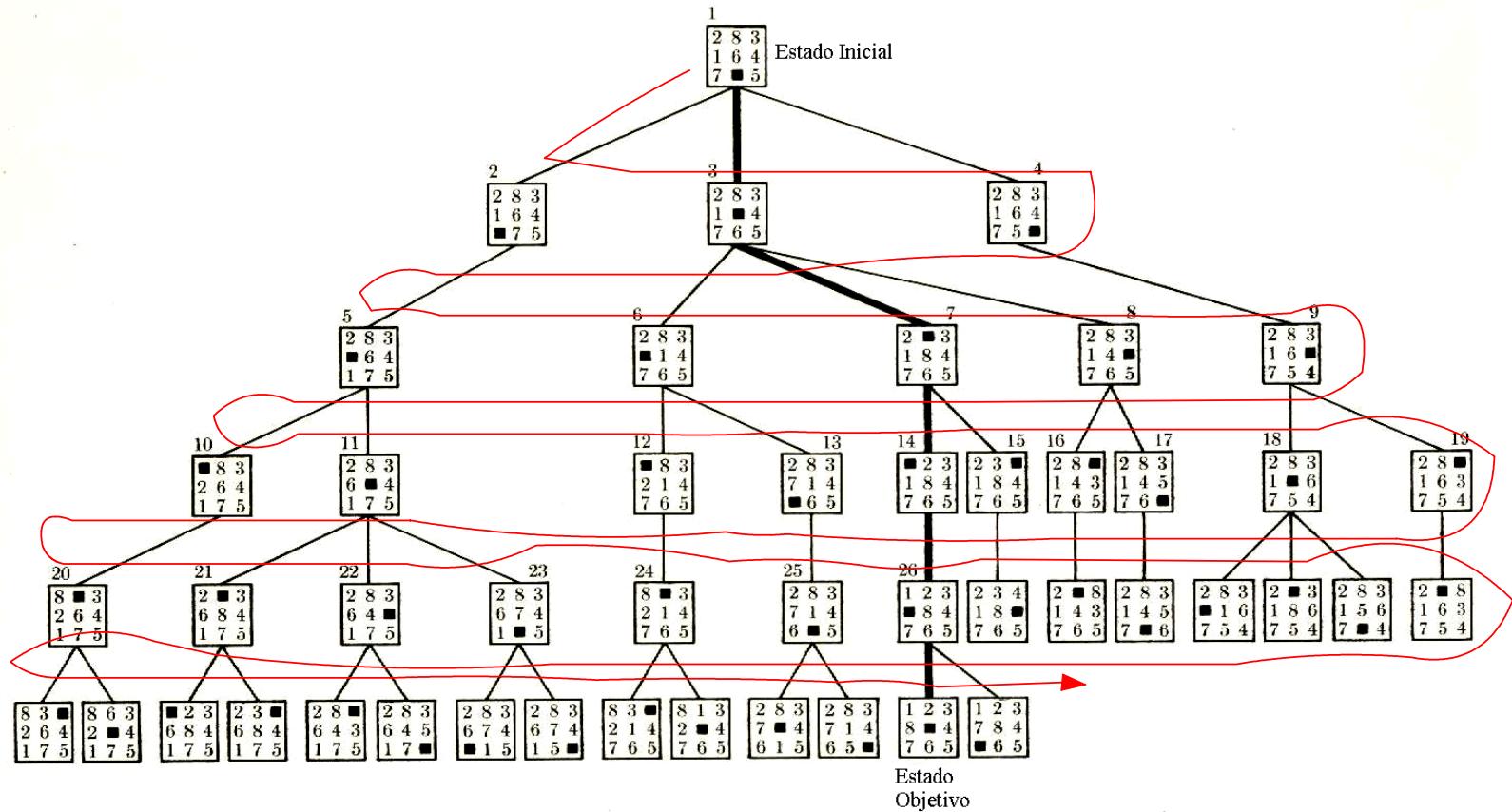
Búsqueda en anchura



Búsqueda en anchura

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier  $\leftarrow$  a FIFO queue with node as the only element
  explored  $\leftarrow$  an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node  $\leftarrow$  POP(frontier) /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child  $\leftarrow$  CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier  $\leftarrow$  INSERT(child, frontier)
```

Búsqueda en anchura



Características

- **Completo:** encuentra la solución si existe y el factor de ramificación es finito en cada nodo
- **Optimalidad:** si todos los operadores tienen el mismo coste, encontrara la solución óptima
- **Eficiencia:** buena si las metas están cercanas
- **Problema:** consume memoria exponencial

Búsqueda con costo uniforme

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier  $\leftarrow$  a priority queue ordered by PATH-COST, with node as the only element
  explored  $\leftarrow$  an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node  $\leftarrow$  POP(frontier) /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child  $\leftarrow$  CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier  $\leftarrow$  INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

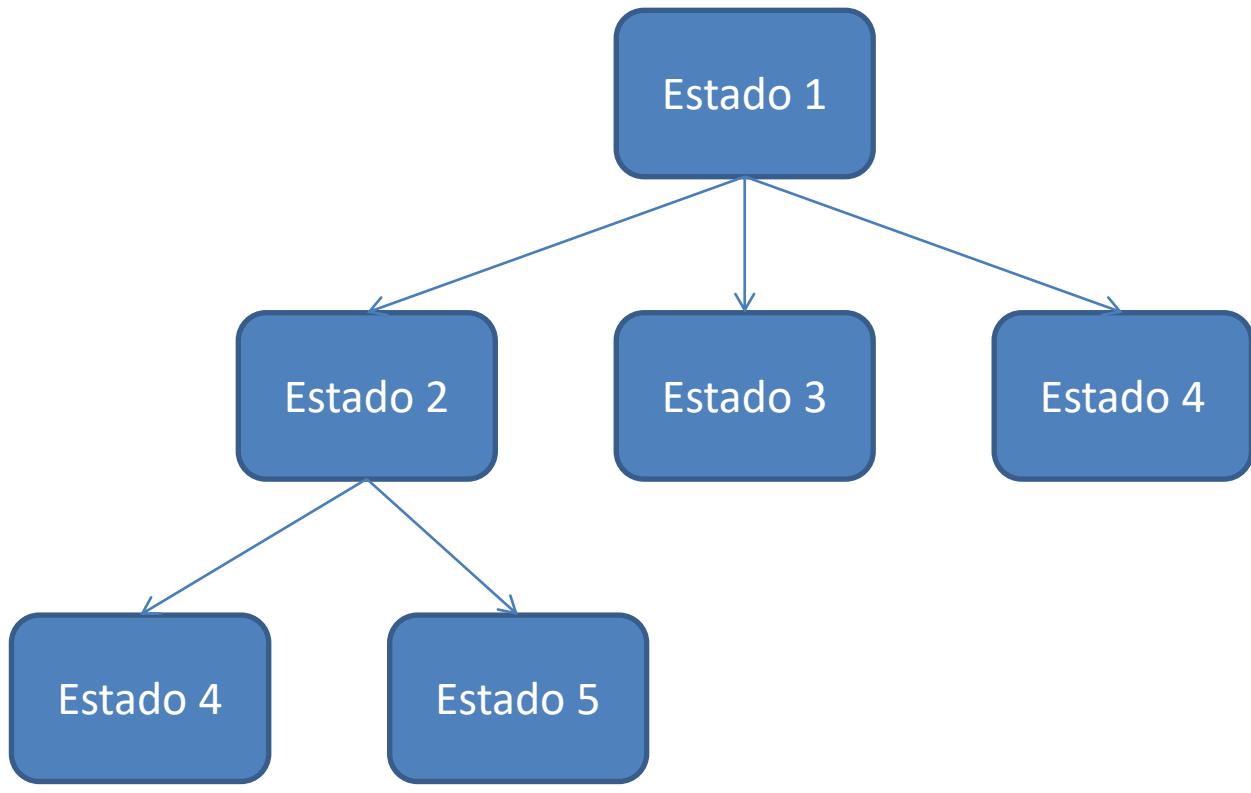
Búsqueda en profundidad

- Igual que la búsqueda en anchura cambiando FIFO por LIFO

Búsqueda en profundidad retroactiva

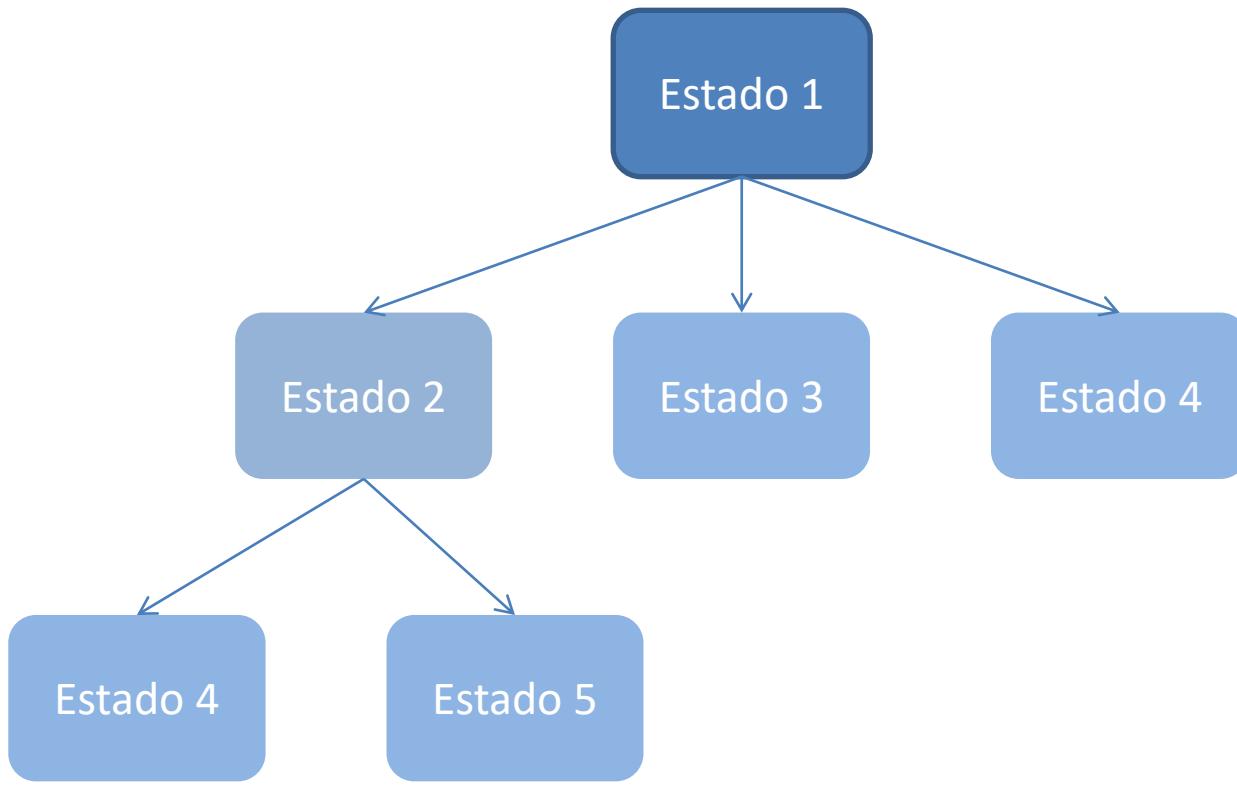
```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
    return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    else if limit = 0 then return cutoff
    else
        cutoff_occurred? ← false
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            result ← RECURSIVE-DLS(child, problem, limit - 1)
            if result = cutoff then cutoff_occurred? ← true
            else if result ≠ failure then return result
        if cutoff_occurred? then return cutoff else return failure
```



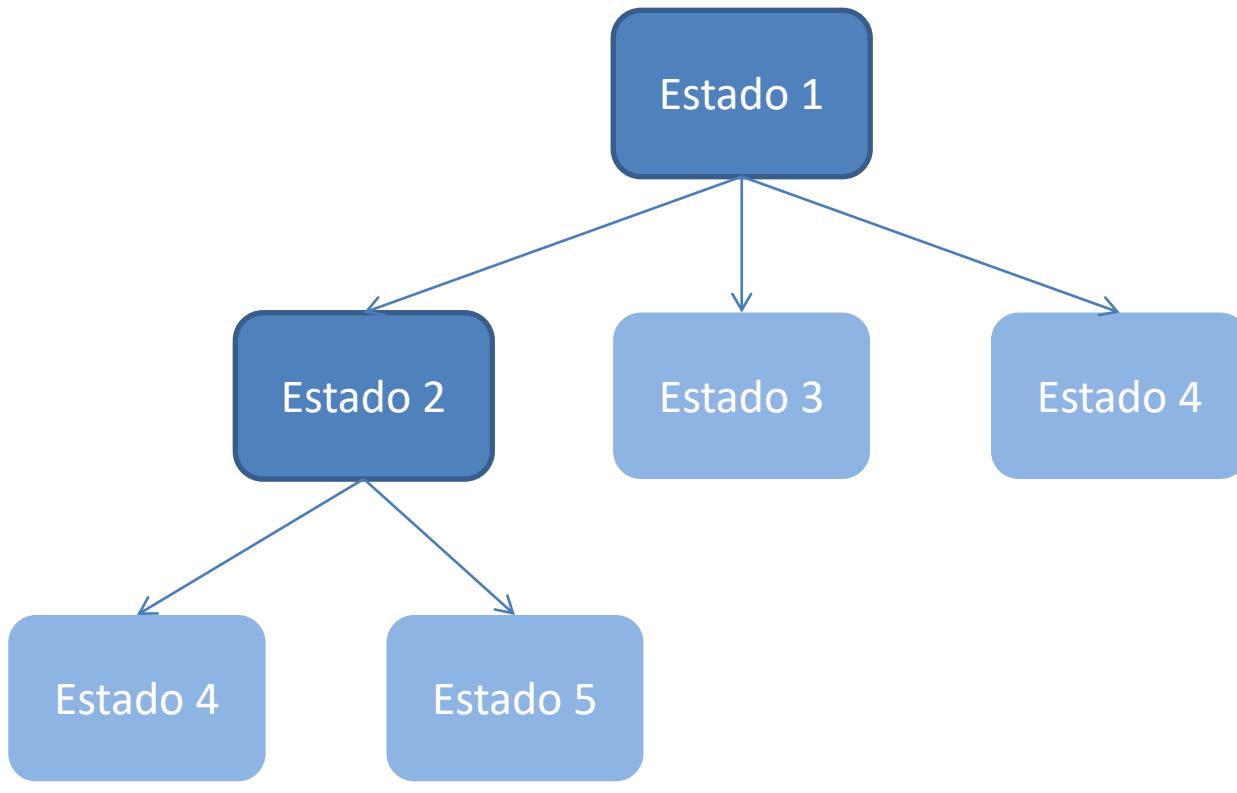
Grafo
Explícito

Grafo
Implícito



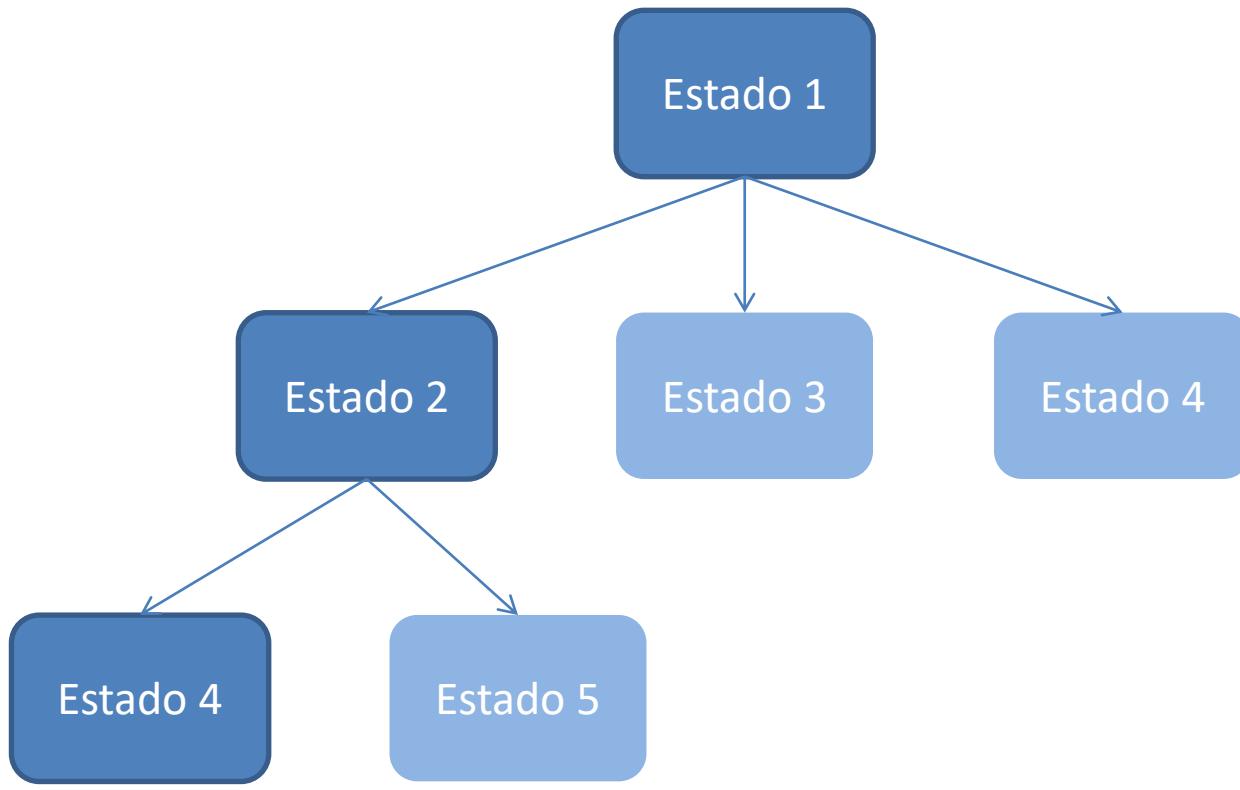
Grafo
Explícito

Grafo
Implícito



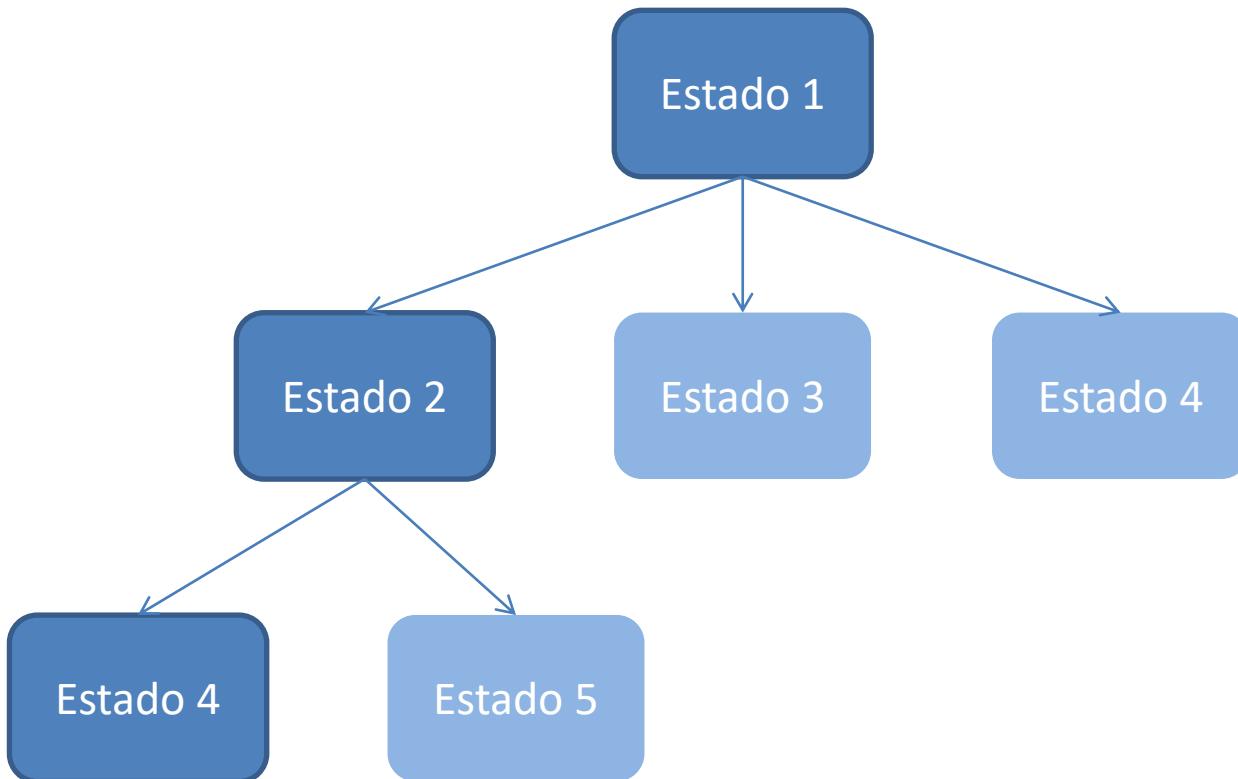
Grafo
Explícito

Grafo
Implícito



Grafo
Explícito

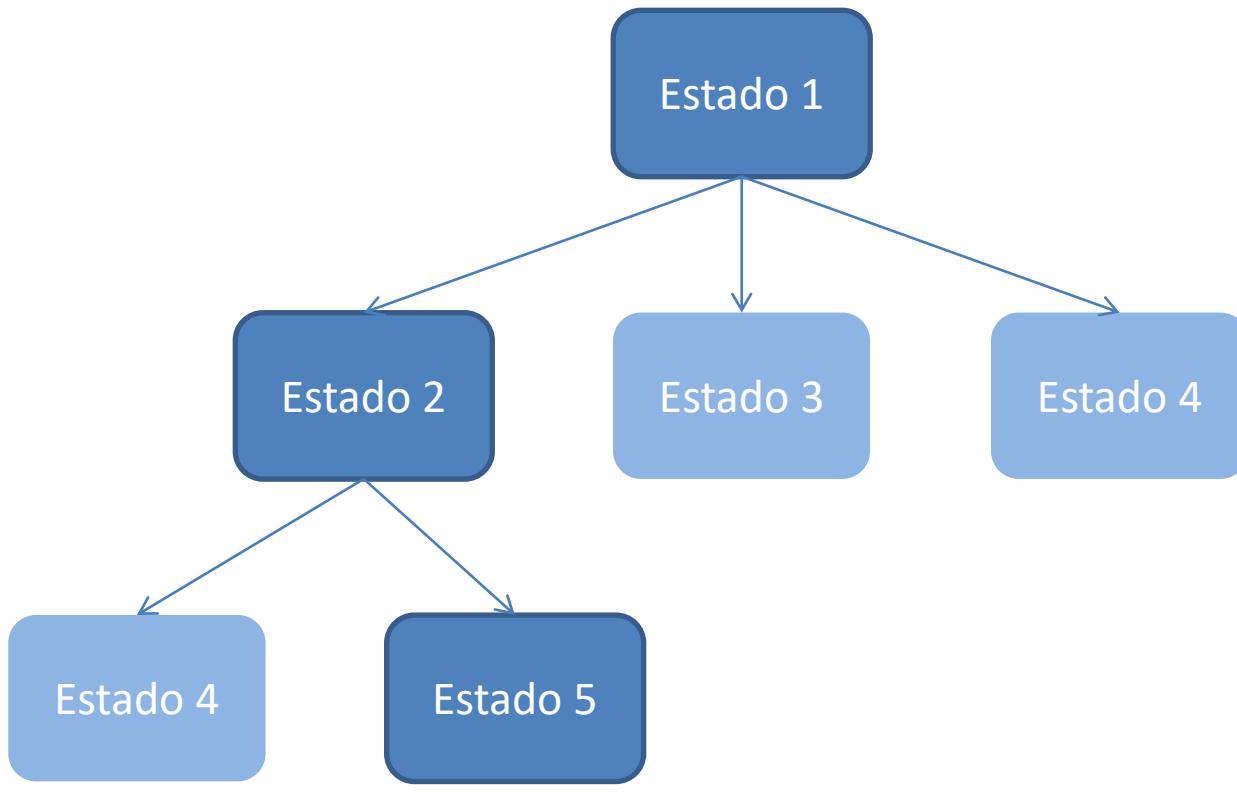
Grafo
Implícito



Vuelta atrás
cutoff

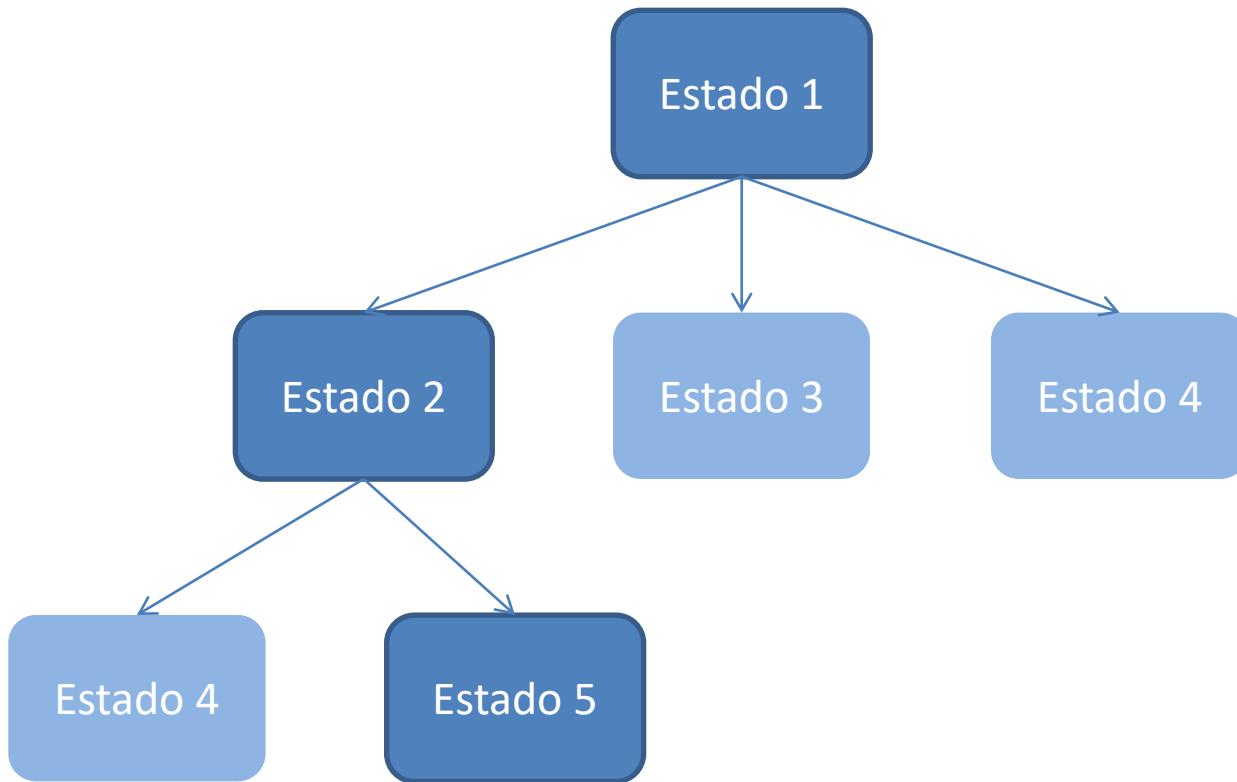
Grafo
Explícito

Grafo
Implícito



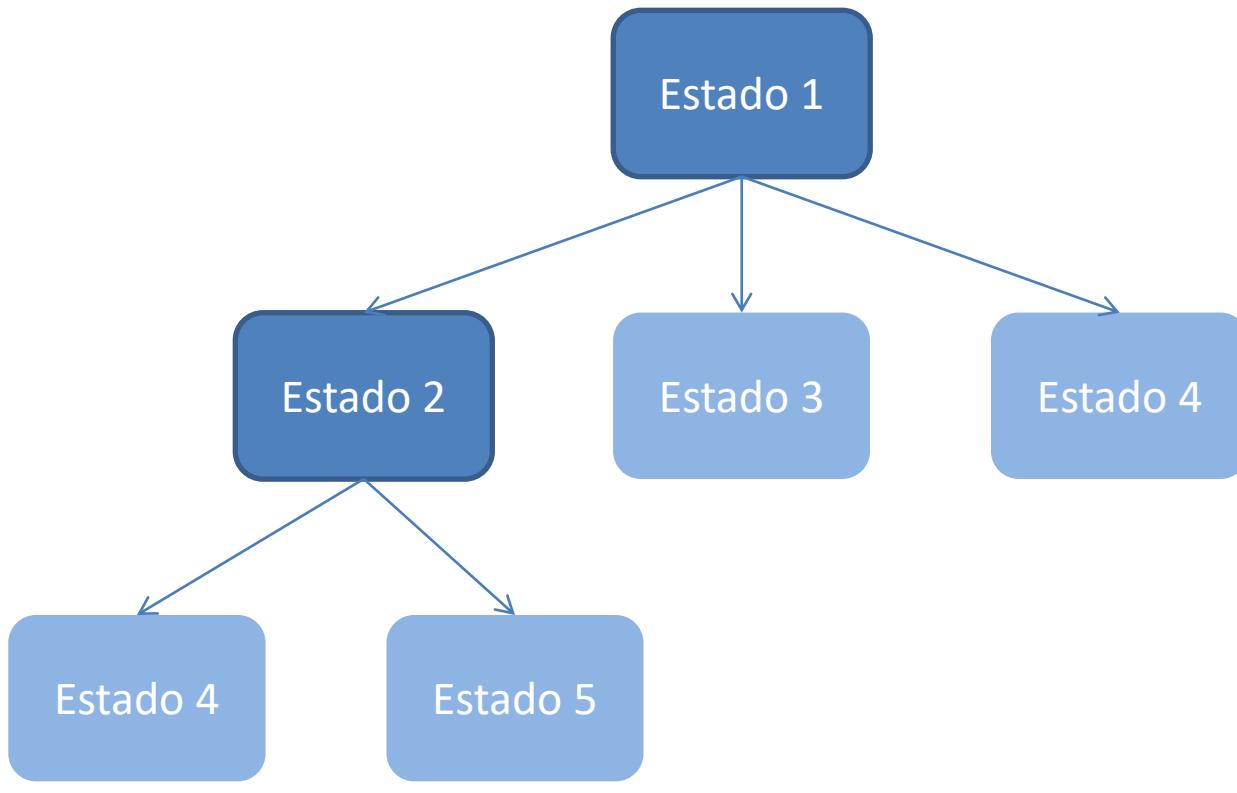
Grafo
Explícito

Grafo
Implícito



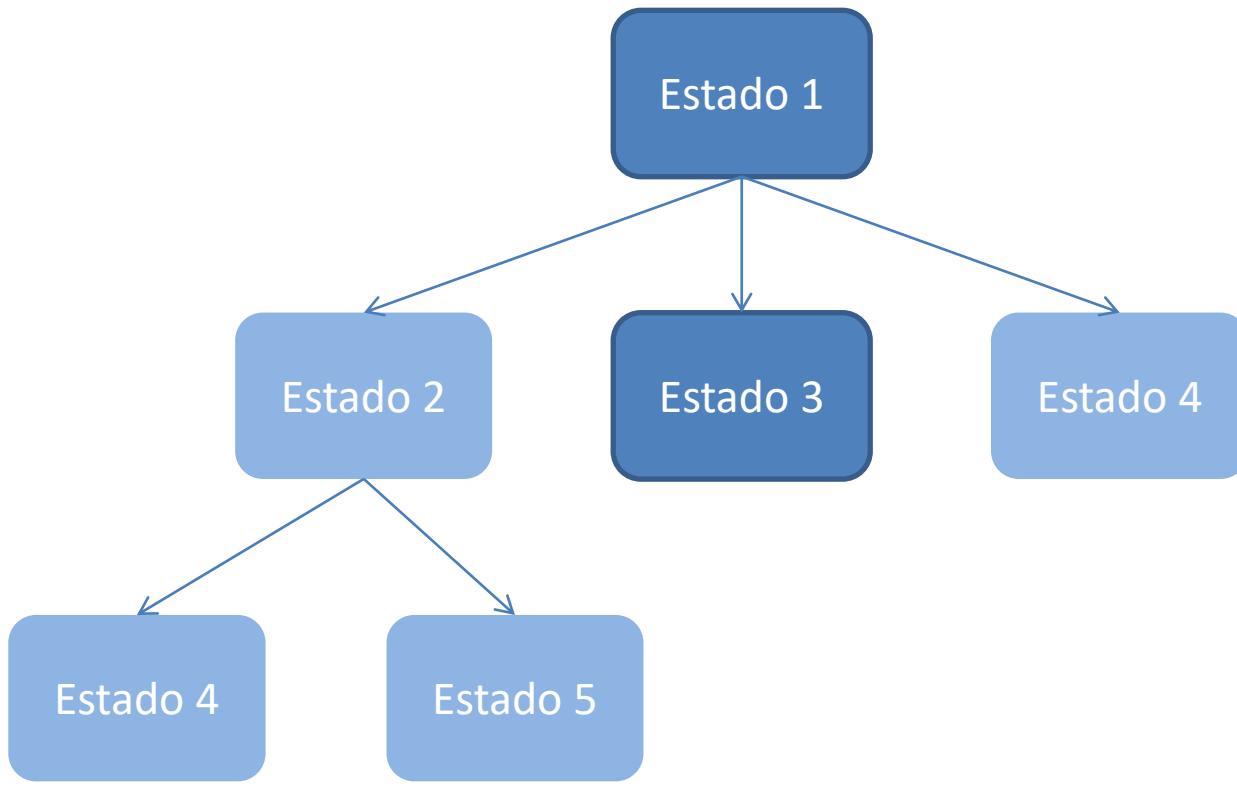
Grafo
Explícito

Grafo
Implícito



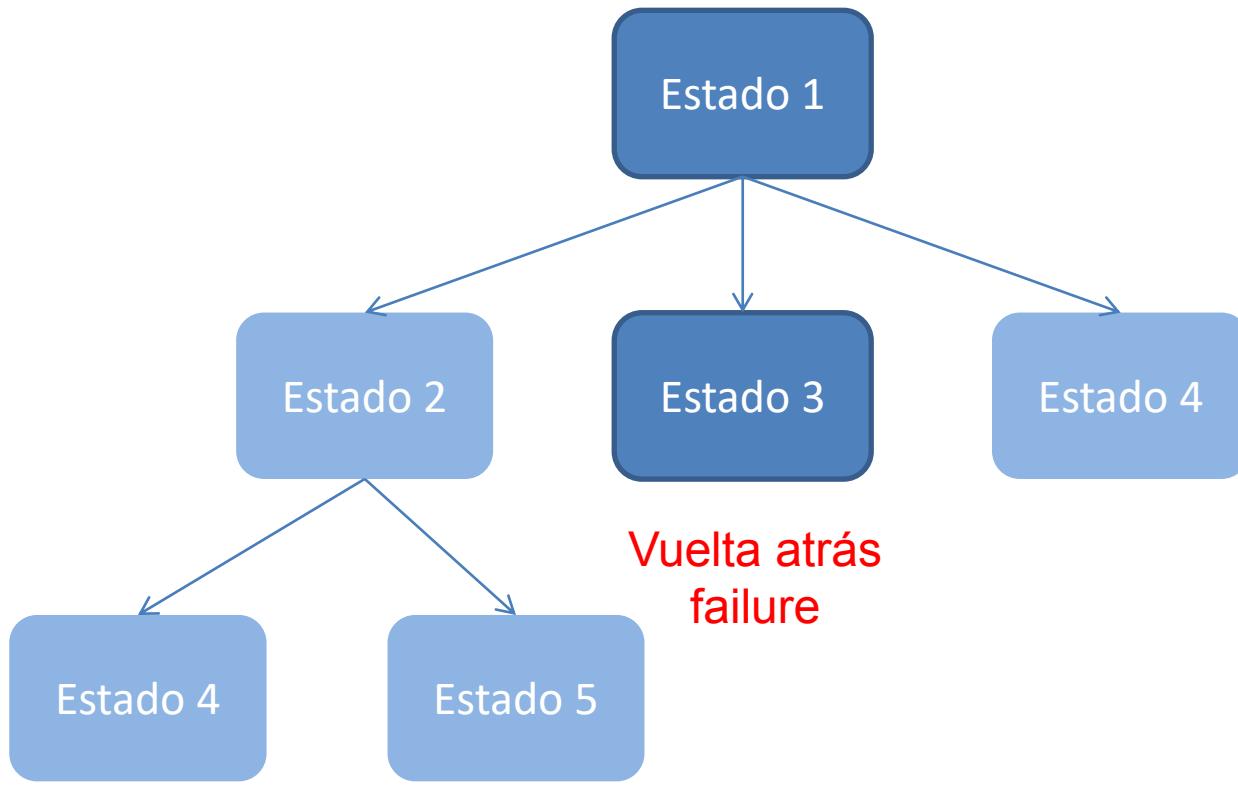
Grafo
Explícito

Grafo
Implícito



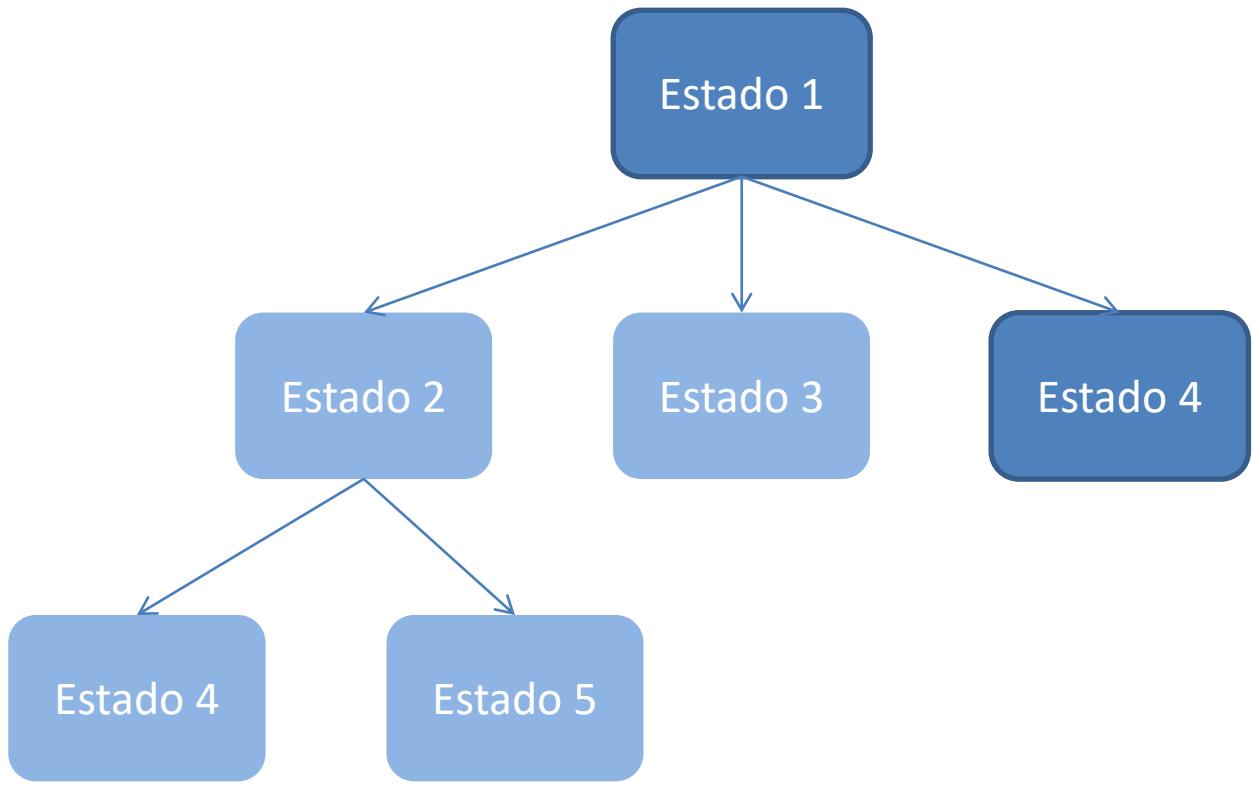
Grafo
Explícito

Grafo
Implícito



Grafo
Explícito

Grafo
Implícito



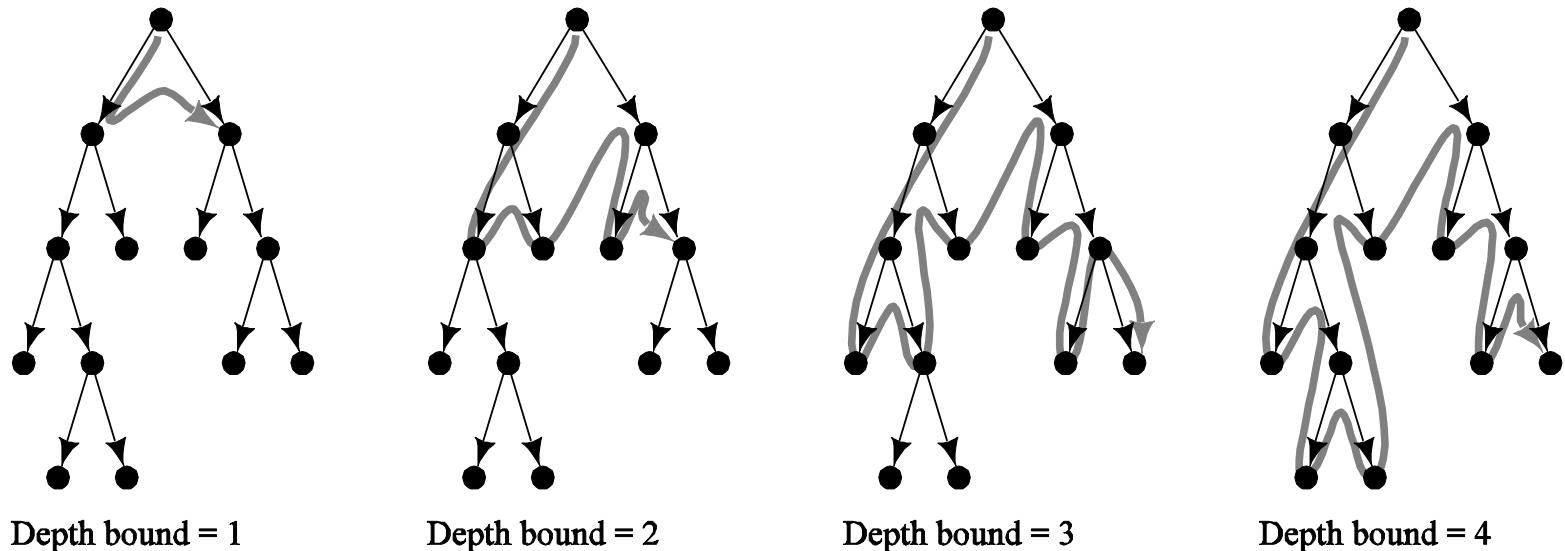
Grafo
Explícito

Grafo
Implícito

Características

- **Compleitud:** no asegura encontrar la solución
- **Optimalidad:** no asegura encontrar la solución óptima
- **Eficiencia:** bueno cuando las metas están alejadas del estado inicial, o hay problemas de memoria
- No es bueno cuando hay ciclos

Descenso iterativo

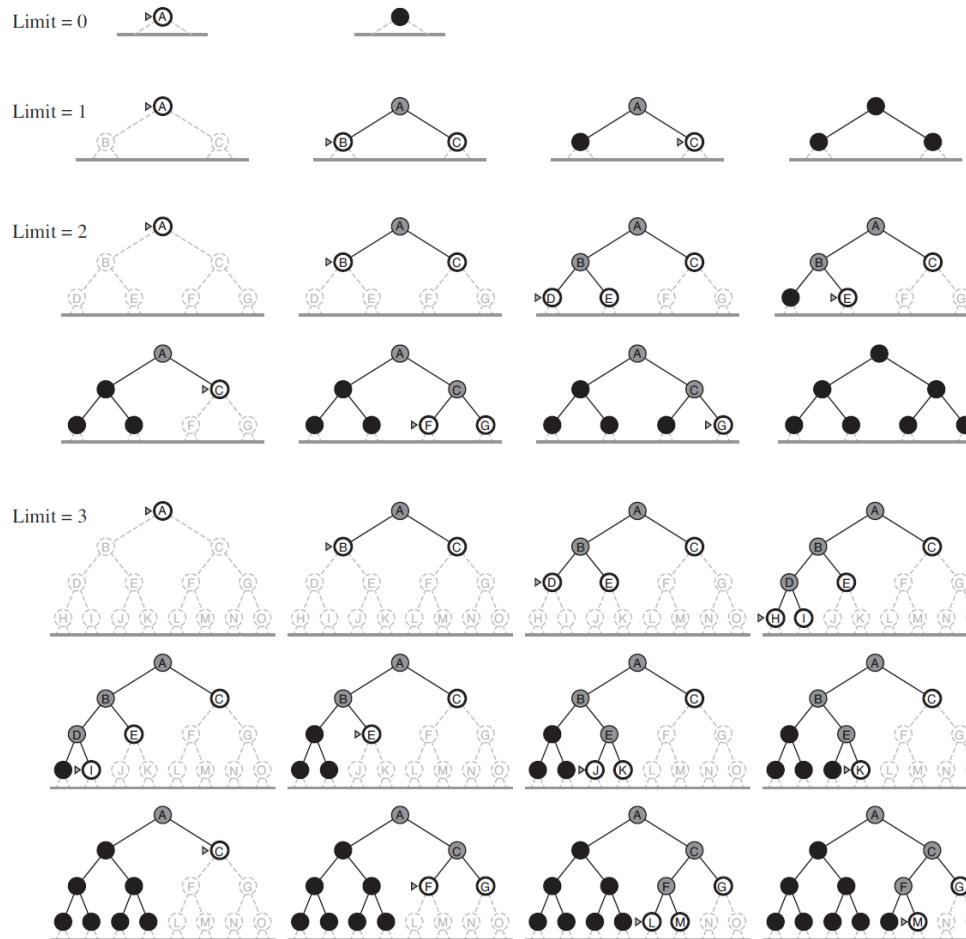


© 1998 Morgan Kaufman Publishers

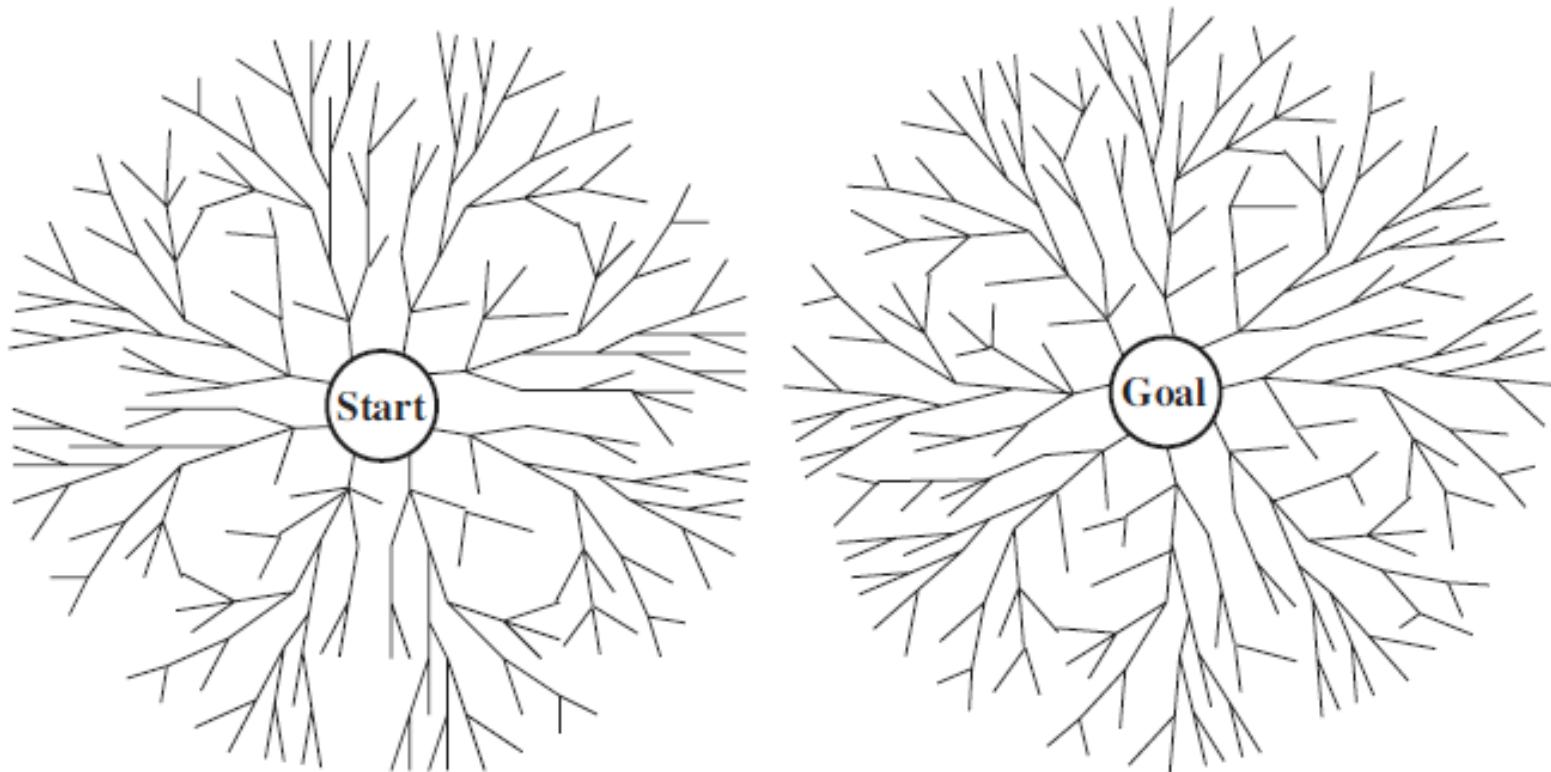
Descenso iterativo

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
```

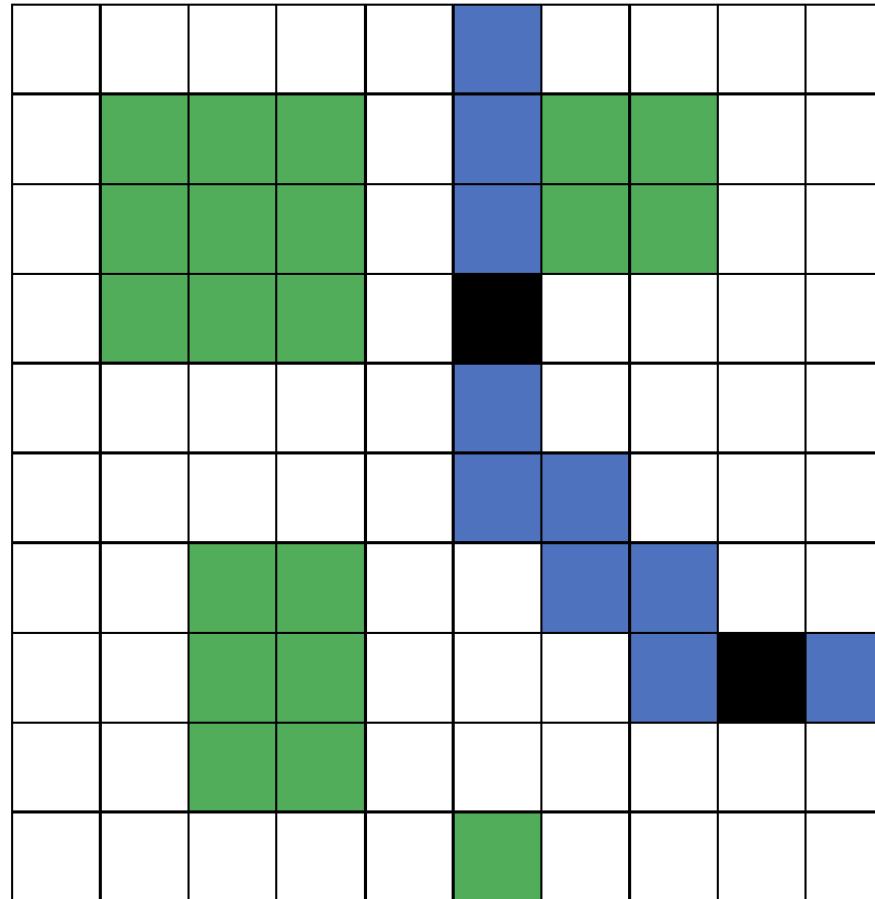
Descenso iterativo



Búsqueda bidireccional



¿Cómo funcionan los algoritmos?



Contenido

- Diseño de un agente deliberativo: búsqueda
- Sistemas de búsqueda y estrategias
- Búsqueda sin información
- **Búsqueda con información**
- Problemas descomponibles y búsqueda

Búsqueda con información

SIMPLE
HEURISTICS
THAT MAKE US
SMART

- Heurísticas
- Métodos de escalada
- Métodos poblacionales:
 - Algoritmos genéticos
- Búsqueda primero el mejor



Heurísticas

- Si se tiene conocimiento perfecto : algoritmo exacto
- Si no se tiene conocimiento : búsqueda sin información
- En la mayor parte de los problemas que resuelven los humanos, se está en posiciones intermedias
- Heurística: (del griego “heurisko" yo encuentro) conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio

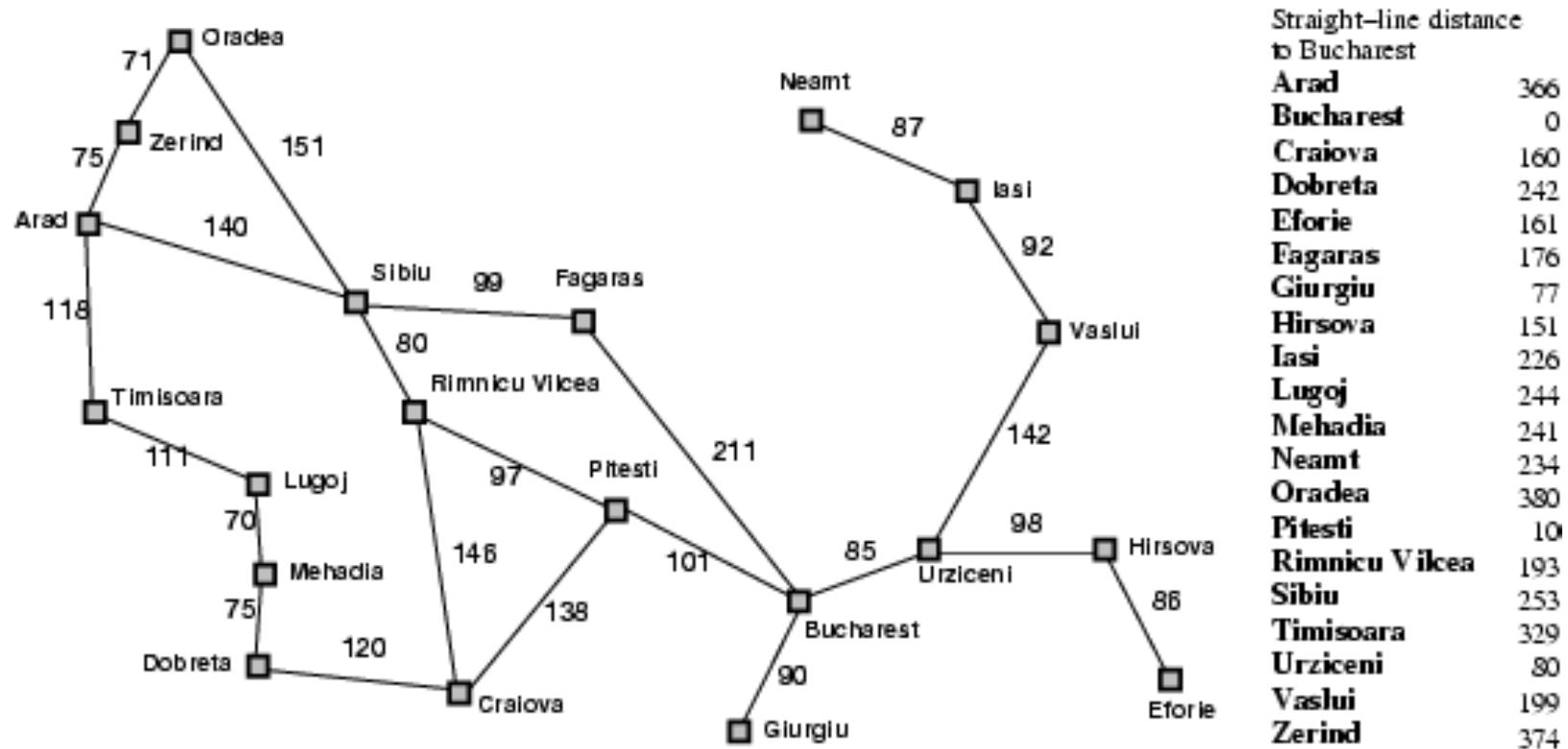
Heurísticas

- Las **heurísticas** son criterios, métodos o principios para decidir cuál de entre varias acciones promete ser la mejor para alcanzar una determinada meta
- En IA, entendemos por heurística un **método para resolver problemas** que en general **no garantiza la solución óptima**, pero que en media **produce resultados satisfactorios** en la resolución de un problema.
- Una heurística encapsula el conocimiento específico/experto que se tiene sobre un problema, y sirve de guía para que un algoritmo de búsqueda pueda encontrar una solución válida aceptable.
- Eventualmente, una heurística puede devolver siempre soluciones óptimas bajo ciertas condiciones (requiere demostración).

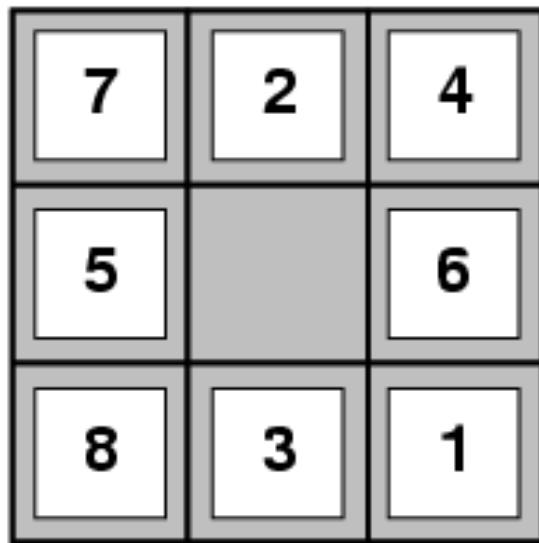
Ajedrez



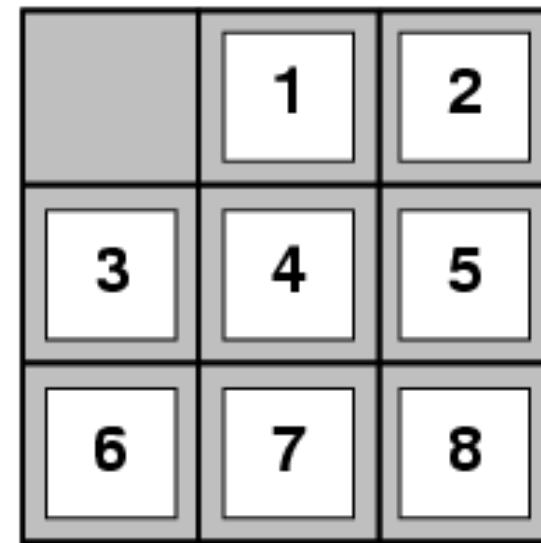
Mapa de carreteras



8-puzzle



Start State

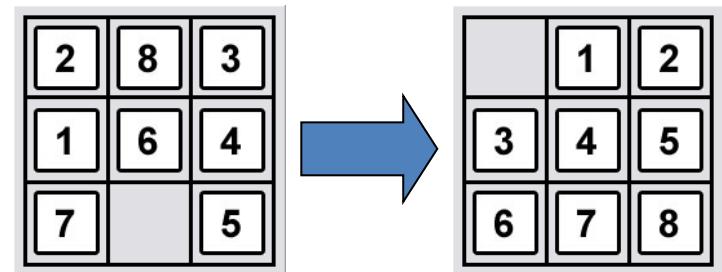


Goal State

8-puzzle

- En IA, implementaremos heurísticas como funciones que devuelven un valor numérico, cuya maximización o minimización guiará al proceso de búsqueda a la solución.
- Ejemplo de heurística en el problema del 8-puzzle:
 - $f(n) = n^o$ de fichas descolocadas en comparación con la posición objetivo a alcanzar.
 - **Objetivo:** Minimizar f .
 - n es un estado (posición de las piezas) del problema.
 - $f(n)$ es la *función heurística*.

$$f((2, 8, 3, 1, 6, 4, 7, 0, 5)) = 9$$



Búsqueda con información

SIMPLE
HEURISTICS
THAT MAKE US
SMART

- Heurísticas
- **Métodos de escalada**
- Métodos poblacionales:
 Algoritmos genéticos
- Búsqueda primero el mejor

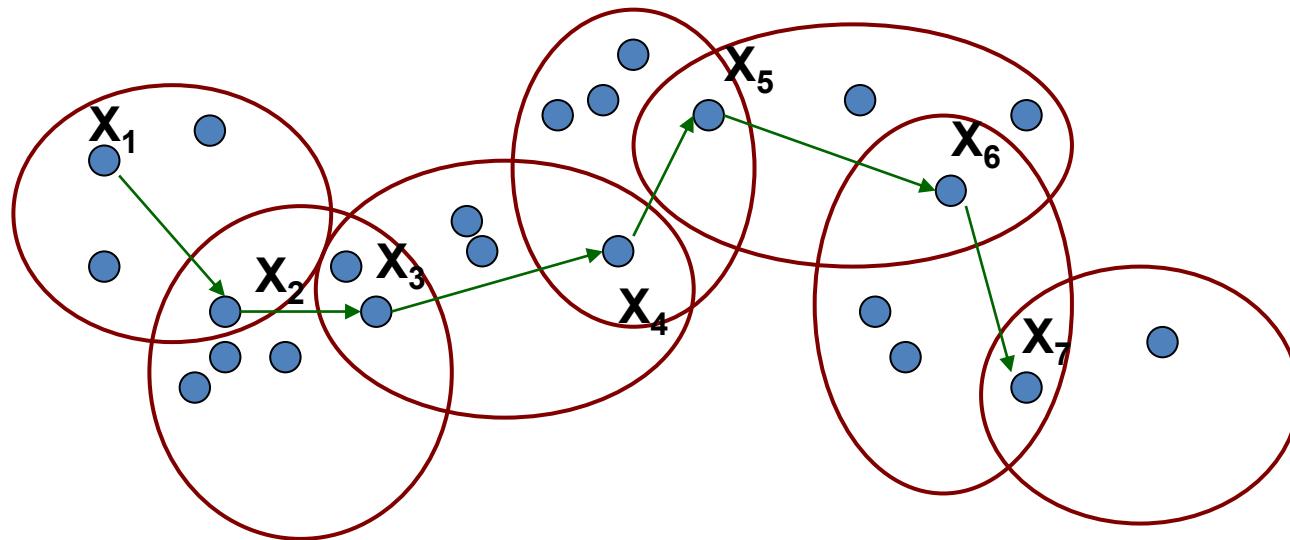


Métodos de escalada

- Algoritmo de escalada simple
- Algoritmo de escalada por la máxima pendiente
- Algunas variaciones estocásticas

Métodos de escalada

- Si dibujamos las soluciones como puntos en el espacio, una **búsqueda local** consiste en seleccionar la solución mejor en el vecindario de una solución inicial, e ir viajando por las soluciones del espacio hasta encontrar un óptimo (local o global).



Algoritmo de escalada simple (ascensión de colinas del primer vecino mejor)

E: Estado activo

```
while (E no sea el objetivo
      y queden nodos por explorar a partir de E) {
    Seleccionar operador A para aplicarlo a E
    Evaluar  $f(A(E))$ 
    if ( $f(A(E)) > f(E)$ ) {
         $E = R(E)$ 
    }
}
```

Algoritmo de escalada por la máxima pendiente (ascensión de colinas del mejor vecino)

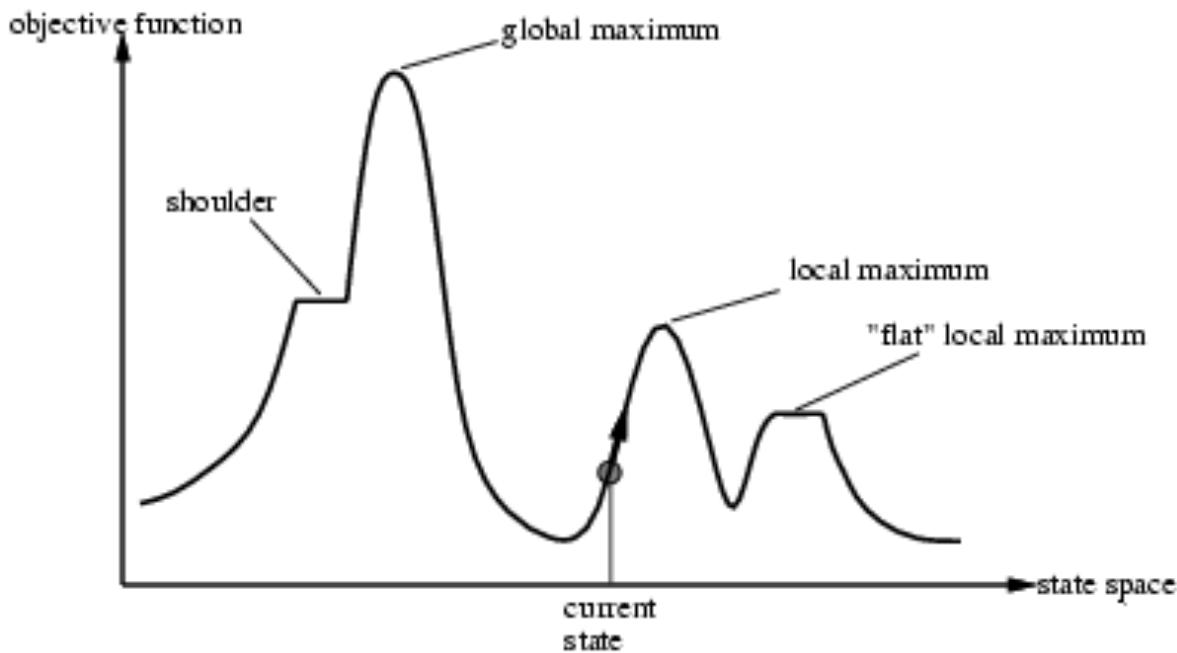
E: Estado activo

```
while (E no sea el objetivo  
y queden nodos por explorar a partir de E) {  
    Para todos los operadores  $A_i$ , obtener  $E_i = A_i(E)$   
    Evaluar  $f(E_i)$  para todos los estados  $E_i = A_i(E)$   
    Seleccionar  $E_{max}$  tal que  $f(E_{max}) = \max\{f(E_i)\}$   
    if ( $f(E_{max}) > f(E)$ ) {  
         $E = E_{max}$   
    } else return E  
}
```

Características

- **Compleitud:** no tiene porque encontrar la solución
- **Admisibilidad:** no siendo completo, aun menos será admisible
- **Eficiencia:** rápido y útil si la función es monótona (de)creciente

Métodos de escalada



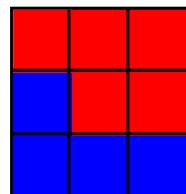
Ejemplo

- **Ejemplo:** Colorear una matriz de **filas*columnas** con **n** colores, de modo que cada celda tenga el mínimo número de celdas adyacentes del mismo color. Asumimos que las celdas adyacentes son las que se encuentran una casilla hacia arriba, abajo, izquierda o derecha de la casilla considerada.
- **Función objetivo:** Minimizar la suma del número de pares de casillas adyacentes del mismo color.
- **Definición del entorno:** El vecindario de una solución estará formado por aquellas soluciones cuyos colores varíen en una única posición de la solución dada.

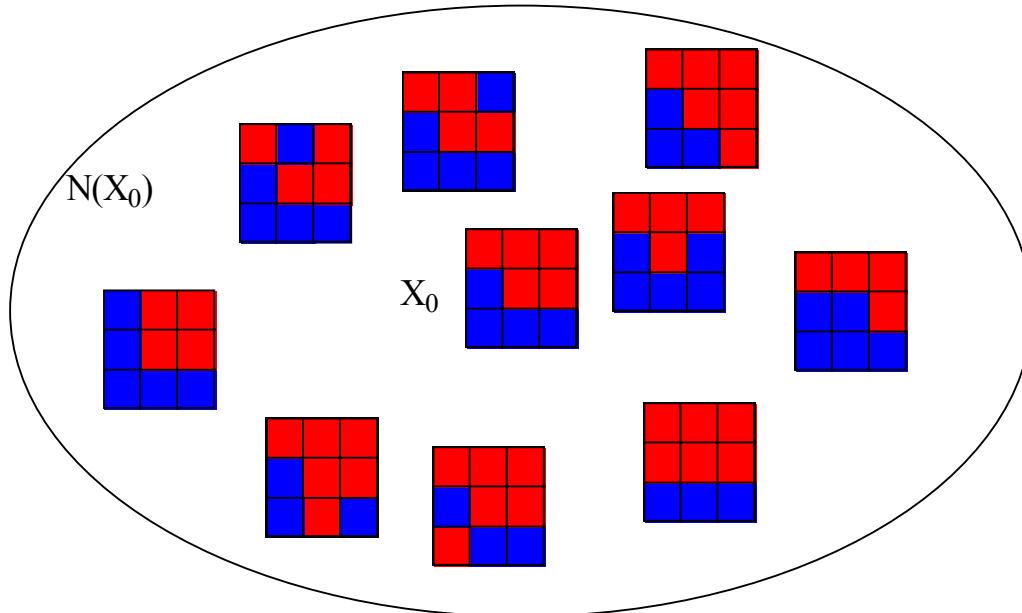
Ejemplo

- **Ejemplo:**

- **Solución inicial:** Generada de forma aleatoria. Supongamos que se ha generado la siguiente para una matriz de **3*3**, a llenar con **2 colores rojo y azul**, con valor de función objetivo **v=8**:

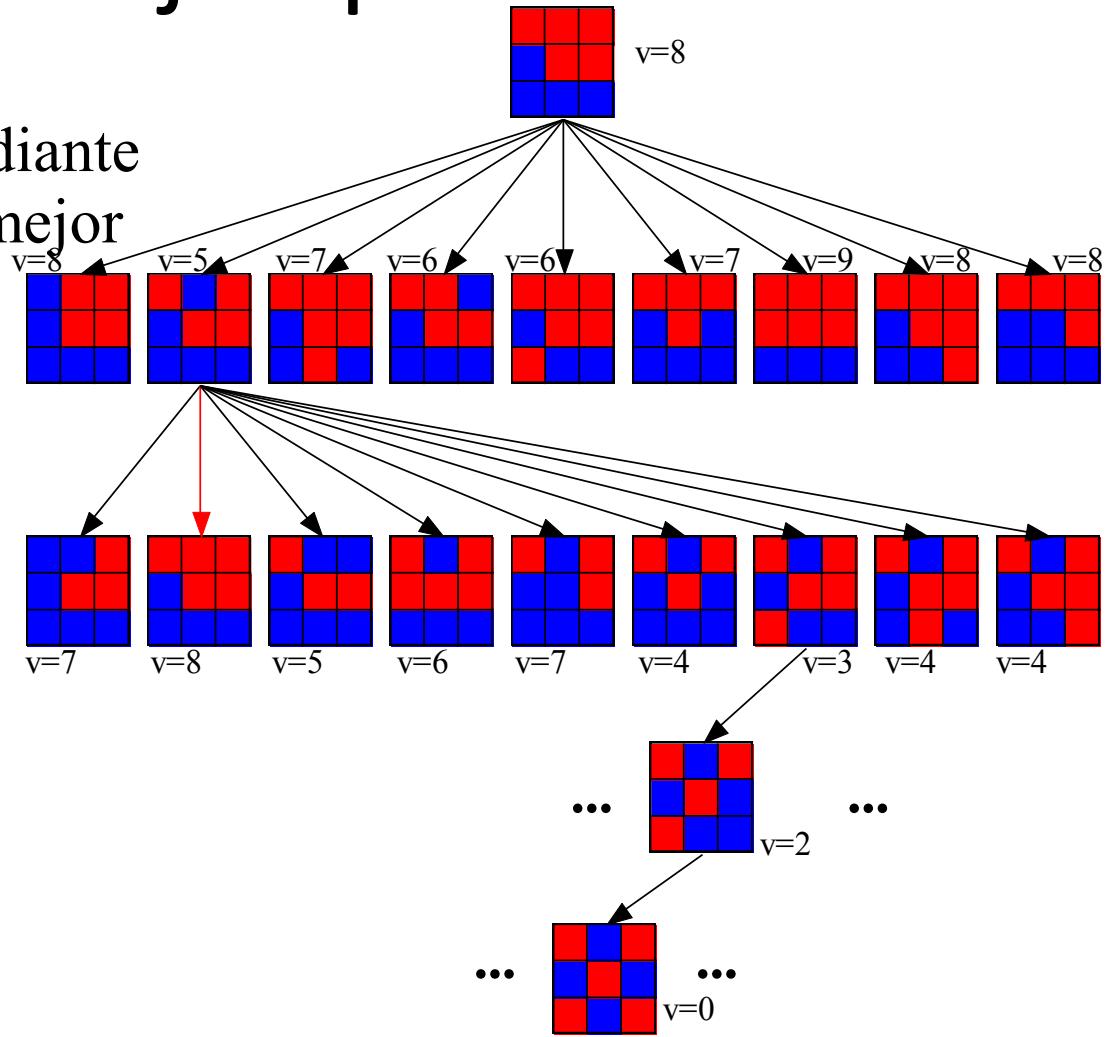


$v=8$



Ejemplo

- **Ejemplo:** Solución al problema anterior mediante ascensión de colinas/mejor opción

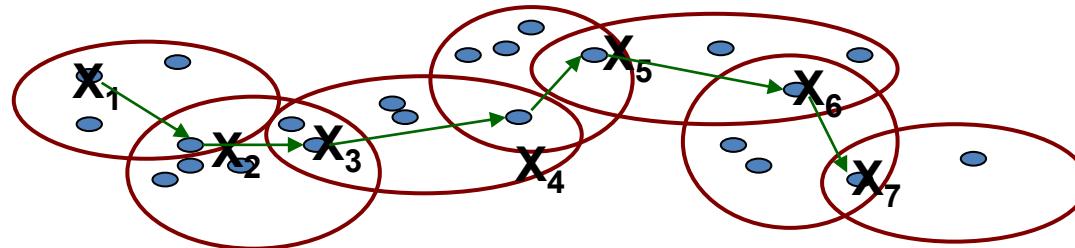


Algunas variaciones estocásticas

- Algoritmo de escalada estocástico
- Algoritmo de escalada de primera opción
- Algoritmo de escalada de reinicio aleatorio
- Enfriamiento simulado

Algoritmo de enfriamiento simulado

- Es un método de búsqueda local.
- Se basa en principios de Termodinámica.
- Al contrario que otros métodos de ascensión de colinas, permite visitar soluciones peores que la actual para evitar óptimos locales.



Algoritmo de enfriamiento simulado

• Un poco de historia:

- En el campo de la Termodinámica, en los años 50 se simuló el proceso de enfriamiento en sistemas de partículas hasta que se llegaba a un estado estable.
- El proceso simulaba la diferencia de energía del sistema, δE , y se quería verificar que la probabilidad de que el sistema tuviese el cambio δE seguía la siguiente fórmula (t es la temperatura actual del sistema; k es una constante física):

$$P[\delta E] = e^{-\frac{\delta E}{k \cdot T}}$$

Algoritmo de enfriamiento simulado

- **Analogía entre el proceso de enfriamiento y el algoritmo de enfriamiento simulado:**
 - Los **estados** por los que pasa el sistema físico de partículas equivalen a las **soluciones factibles** del algoritmo.
 - La **energía E** del **estado actual** del sistema es el valor de la **función objetivo de la solución actual**. Ambos tienen que minimizarse.
 - Un **cambio de estado** en el sistema equivale a **explorar el entorno de una solución y viajar a una solución vecina**.
 - El **estado final estable** (congelado) es la **solución final** del algoritmo.

Algoritmo de enfriamiento simulado

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
inputs: *problem*, a problem
 schedule, a mapping from time to “temperature”

```
current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
for  $t = 1$  to  $\infty$  do
     $T \leftarrow schedule(t)$ 
    if  $T = 0$  then return current
    next  $\leftarrow$  a randomly selected successor of current
     $\Delta E \leftarrow next.VALUE - current.VALUE$ 
    if  $\Delta E > 0$  then current  $\leftarrow next$ 
    else current  $\leftarrow next$  only with probability  $e^{\Delta E/T}$ 
```

Algoritmo de enfriamiento simulado

- La **solución inicial** se puede generar de forma aleatoria, por conocimiento experto, o por medio de otras técnicas algorítmicas como *greedy*.
- La **actualización de temperatura** también es heurística, y hay varios métodos:
 - $T \leftarrow \alpha \cdot T$, con α en $(0,1)$
 - $T \leftarrow 1/(1+k)$, con $k =$ número de iteraciones del algoritmo hasta el momento, etc.
- **Número de vecinos a generar:** Fijo $N(T) = \text{cte}$, dependiente de la temperatura $N(T) = f(T)$, etc.

Algoritmo de enfriamiento simulado

- Tanto la temperatura inicial como la temperatura final T_i y T_f son parámetros de entrada al algoritmo.
- Es difícil asignar un valor concreto a T_f , por lo que la condición de parada se suele sustituir por un número específico de iteraciones a realizar.

• Ventajas:

- Al ser un método probabilístico, tiene capacidad para salir de óptimos locales.
- Es eficiente.
- Es fácil de implementar.

Algoritmo de enfriamiento simulado

- **Inconvenientes:**

- Encontrar la temperatura inicial T_i , el método de actualización de temperatura α , el número de vecinos a generar en cada estado y el número de iteraciones óptimo es una tarea que requiere de **muchas pruebas de ensayo y error** hasta que ajustamos los parámetros óptimos.
- Pese a todo, el algoritmo puede proporcionar soluciones mucho mejores que utilizando algoritmos no probabilísticos.

Búsqueda con información

SIMPLE
HEURISTICS
THAT MAKE US
SMART

- Heurísticas
- Métodos de escalada
- **Métodos poblacionales:**
Algoritmos genéticos
- Búsqueda primero el mejor



Algoritmos genéticos

- La simulación de procesos naturales es un campo de investigación muy amplio en Inteligencia Artificial.
- Ejemplos son la **computación evolutiva, biocomputación, algoritmos bioinspirados**, etc.
- Si ha funcionado bien en la naturaleza, ¿porqué una simulación de estos procesos no iba a proporcionar buenos resultados en un computador?
- Ejemplos:
 - Algoritmos genéticos.
 - Algoritmos basados en Colonias de Hormigas.
 - Algoritmos basados en inteligencia de enjambres.

Algoritmos genéticos

- Son algoritmos de optimización basados en el proceso de la evolución natural de Darwin.
- En un proceso de evolución, existe una **población de individuos**. Los más adecuados a su entorno **se reproducen y tienen descendencia** (a veces **con mutaciones** que mejoran su idoneidad al entorno). Los más adecuados sobreviven para la siguiente generación.
- No necesitan partir de un nodo/estado inicial: ¡Hay toda una población!



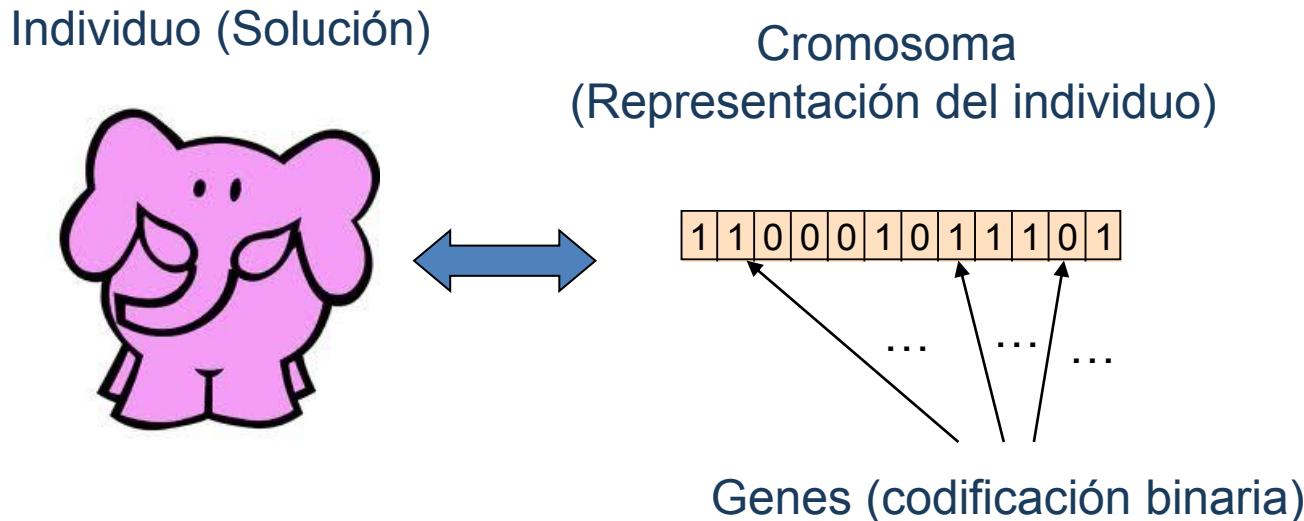
- Su objetivo es encontrar una solución cuyo valor de función objetivo sea óptimo.

Algoritmos genéticos

- **Cromosoma** \leftrightarrow Vector representación de una solución al problema.
- **Gen** \leftrightarrow Característica/Variable/Atributo concreto del vector de representación de una solución
- **Población** \leftrightarrow Conjunto de soluciones al problema.
- **Adecuación al entorno** \leftrightarrow Valor de función objetivo (fitness).
- **Selección natural** \leftrightarrow Operador de selección.
- **Reproducción sexual** \leftrightarrow Operador de cruce.
- **Mutación** \leftrightarrow Operador de mutación.
- **Cambio generacional** \leftrightarrow Operador de reemplazamiento.

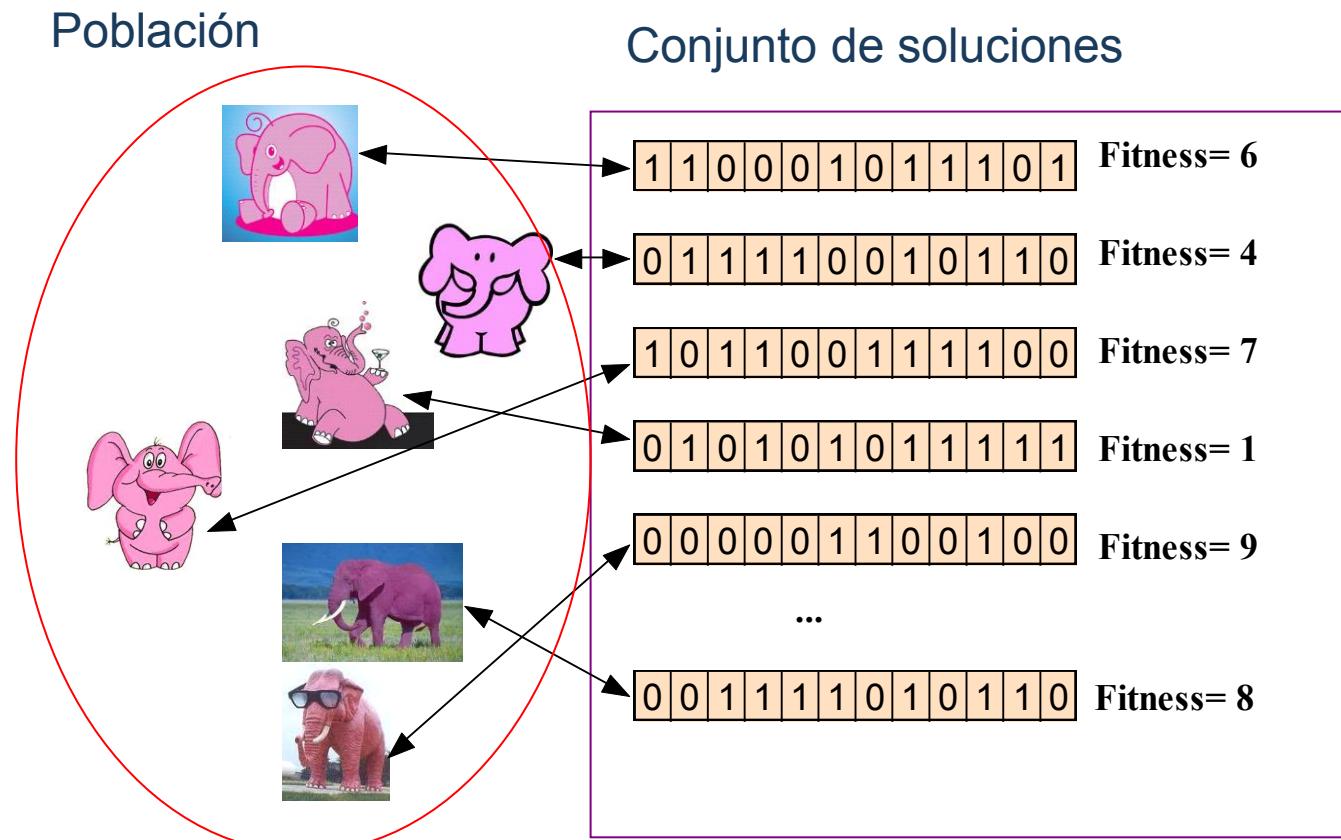
Algoritmos genéticos

- **Ejemplo:** Cromosoma que codifica una solución a un problema.
Cada característica del problema es un valor 0/1.



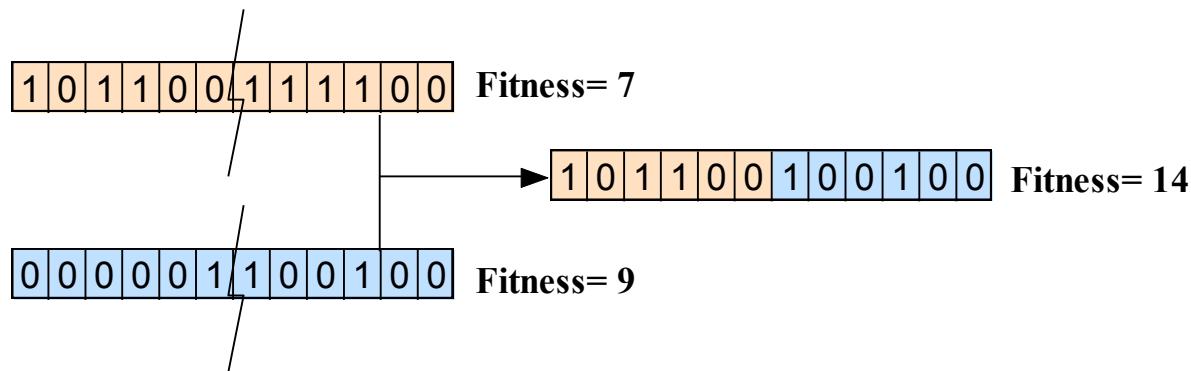
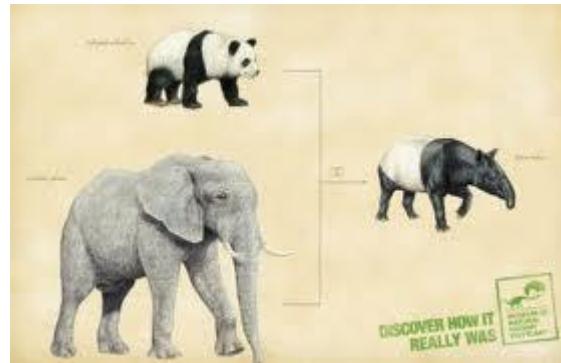
Algoritmos genéticos

- **Ejemplo:** Población. Conjunto de individuos (cada uno con su fitness).



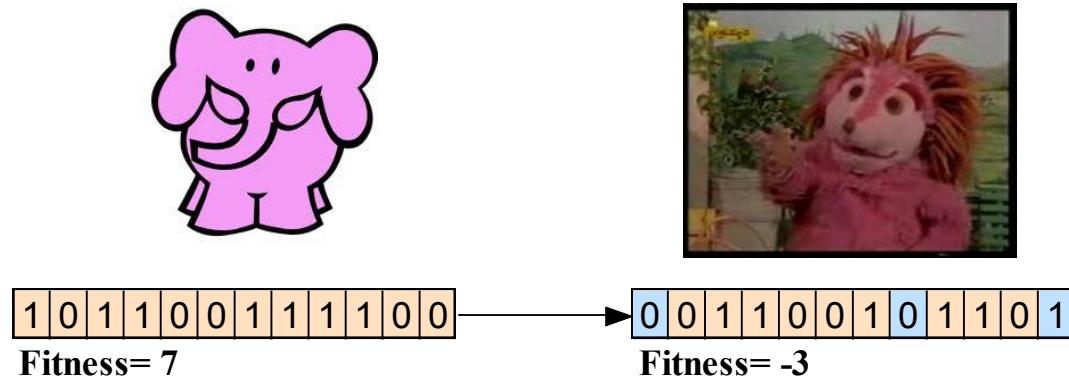
Algoritmos genéticos

- **Ejemplo:** Cruce. Combinación de soluciones de la población para generar descendientes.



Algoritmos genéticos

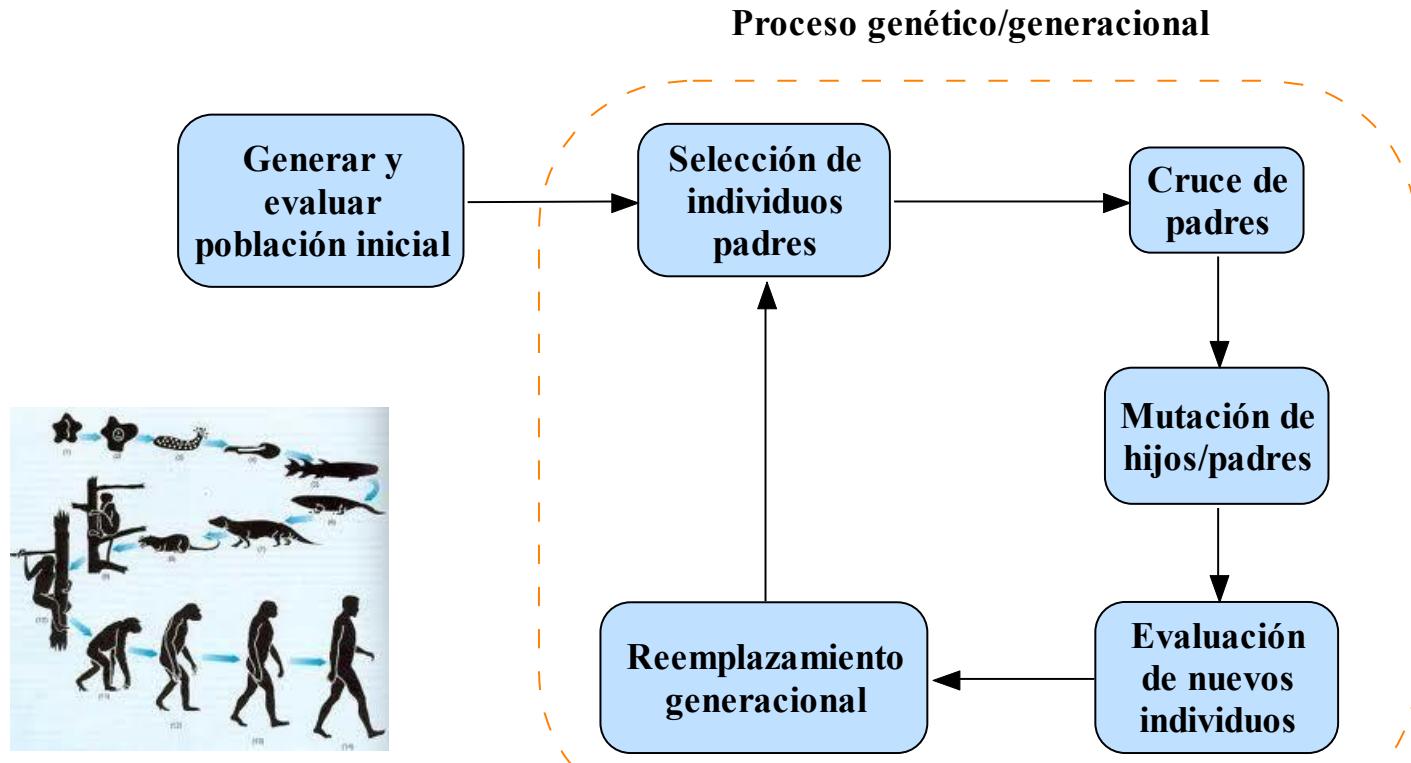
- **Ejemplo:** Mutación. Uno o más genes de un individuo pueden mutar para generar una nueva solución.



- En la población, hay una probabilidad dada a priori de que un individuo pueda mutar. A su vez, cuando un individuo muta, existe otra probabilidad de que cada gen mute o no.

Algoritmos genéticos

• Proceso de un algoritmo genético:



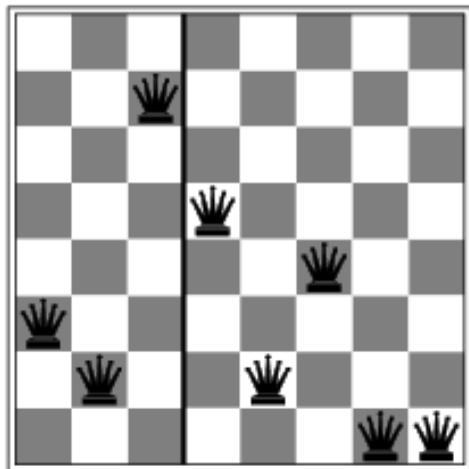
Ejemplo



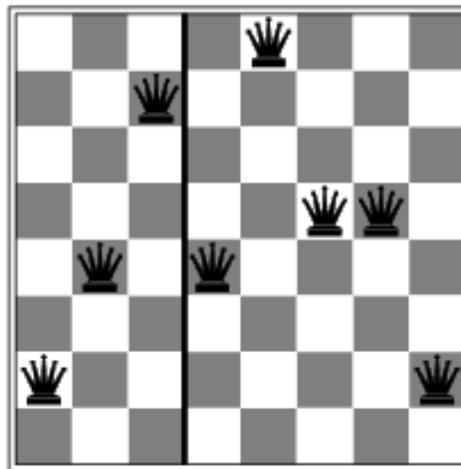
Función de evaluación (8 reinas) = número de pares de reinas no atacadas

28 para una solución

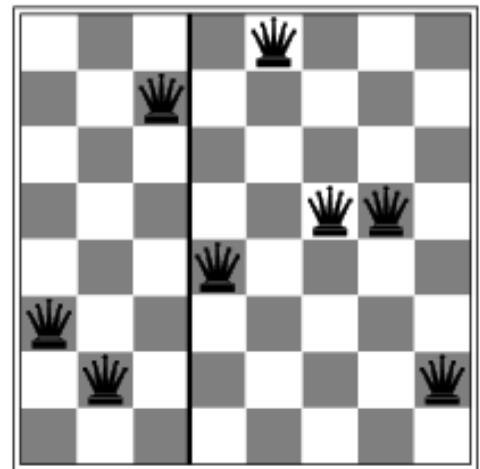
Ejemplo



+

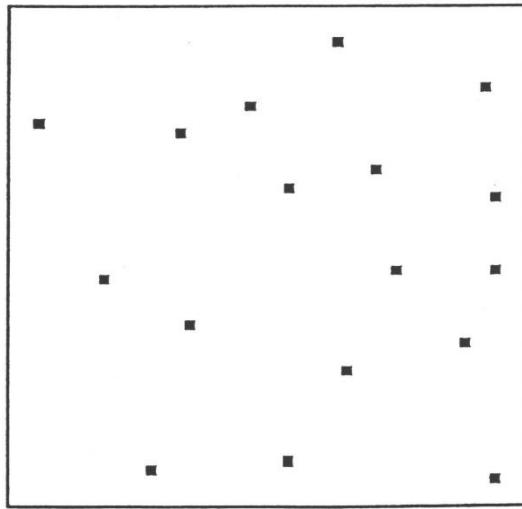


=



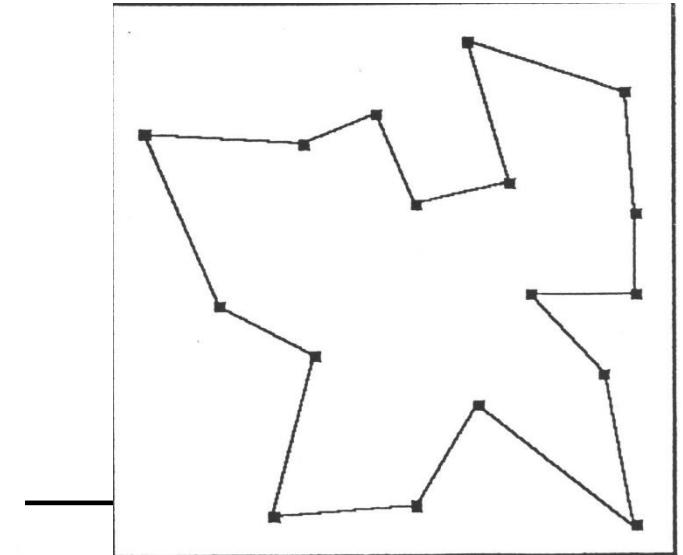
Ejemplo: El problema del viajante de comercio

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	12.87	19.71	31.56	22.70	17.26	23.33	12.16	24.71	34.51	12.58	21.38	42.37	27.43	36.51	19.10	1.18
2	0	15.80	37.51	21.52	28.57	35.43	22.70	16.78	28.57	11.13	25.26	50.62	38.16	35.97	9.04	34.56	
3	0	50.18	36.56	35.86	35.51	21.60	31.50	43.51	25.58	38.78	61.57	46.15	51.10	23.50	48.52		
4	0	20.90	21.52	37.62	38.14	33.26	31.90	27.13	13.03	15.53	18.39	19.37	35.84	8.12			
5	0	26.00	40.72	33.74	12.87	14.71	11.68	9.72	35.86	30.96	15.06	16.78	15.27				
6	0	16.99	18.53	34.51	40.20	22.34	18.53	27.70	10.80	34.94	32.08	25.24					
7	0	14.54	46.60	54.54	33.80	34.52	40.35	22.09	51.20	41.84	41.73						
8	0	36.31	46.12	24.21	30.50	45.72	28.09	46.77	30.20	39.71							
9	0	12.54	13.31	21.52	48.18	41.50	23.85	8.50	27.43								
10	0	22.43	23.33	46.67	44.80	16.31	20.53	24.58									
11	0	14.71	40.81	30.52	26.21	10.50	23.93										
12	0	27.43	21.97	17.20	23.35	10.35											
13	0	18.89	32.78	50.15	22.59												
14	0	35.88	40.51	24.71													
15	0	30.18	11.90														
16	0	31.31															
17	0																



17! (3.5568734e14)
soluciones posibles

Solución óptima:
Coste=226.64



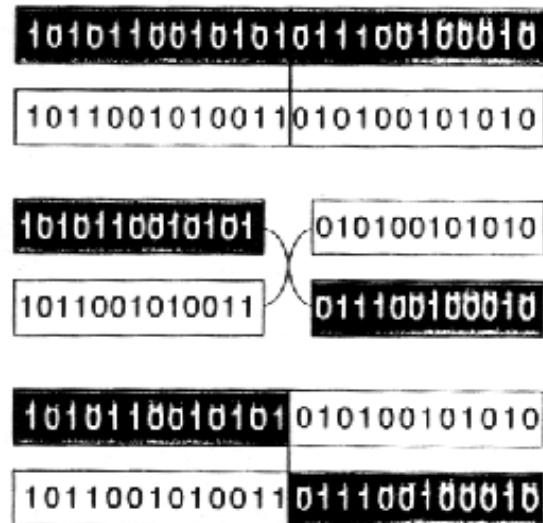
¿Cómo aplicar los algoritmos genéticos a la Representación de Orden? Viajante de Comercio

Padre 1

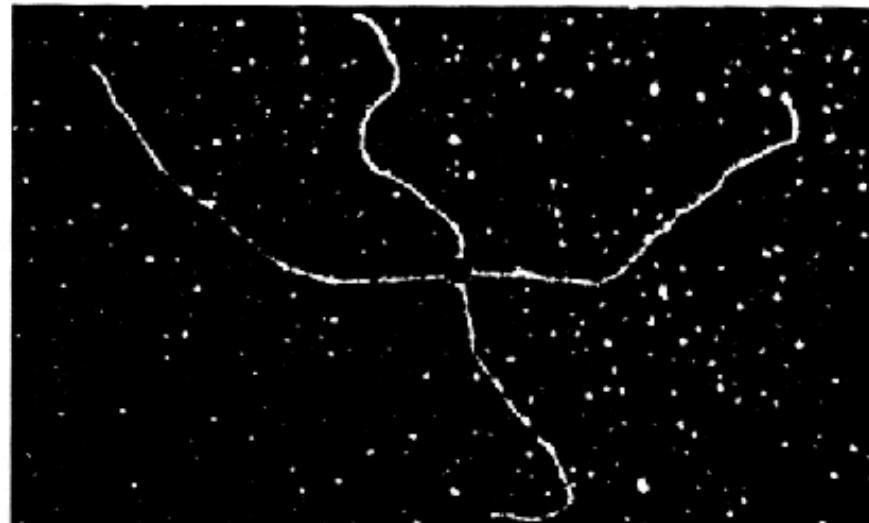
7	3	1	8	2	4	6	5
---	---	---	---	---	---	---	---

Padre 2

4	3	2	8	6	7	1	5
---	---	---	---	---	---	---	---



CROSSOVER is the fundamental mechanism of genetic rearrangement for both real organisms and genetic algorithms.



Chromosomes line up and then swap the portions of their genetic code beyond the crossover point.

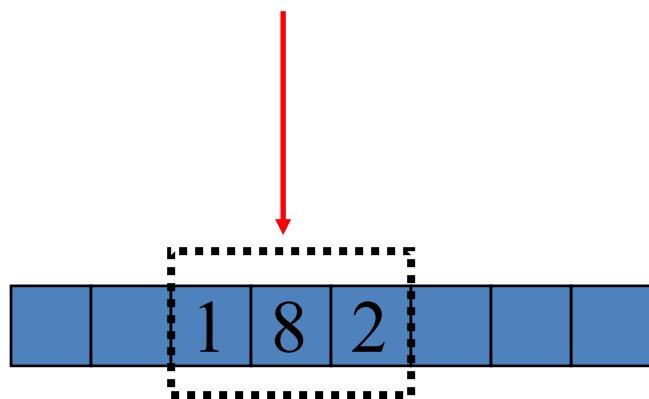
Representación de Orden y Operador de cruce OX

Padre 1

7	3	1	8	2	4	6	5
---	---	---	---	---	---	---	---

Padre 2

4	3	2	8	6	7	1	5
---	---	---	---	---	---	---	---



¿Qué hacemos con las ciudades restantes?

Representación de Orden y Operador de cruce OX

Padre 1

7	3	1	8	2	4	6	5
---	---	---	---	---	---	---	---

Padre 2

4	3	2	8	6	7	1	5
---	---	---	---	---	---	---	---

7, 3, 4, 6, 5

4 3 6 7 5

		1	8	2			
--	--	---	---	---	--	--	--

¿Qué hacemos con las ciudades restantes?

731 86715

Fuera: 4 y 2

Repetidos: 7 y 1

Representación de Orden y Operador de cruce OX

Padre 1

7	3	1	8	2	4	6	5
---	---	---	---	---	---	---	---

Padre 2

4	3	2	8	6	7	1	5
---	---	---	---	---	---	---	---

7, 3, 4, 6, 5

		1	8	2			
--	--	---	---	---	--	--	--

Orden

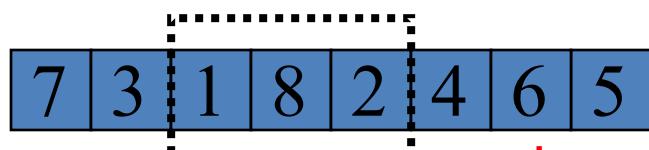
4, 3, 6, 7, 5

Hijo 1

7	5	1	8	2	4	3	6
---	---	---	---	---	---	---	---

Representación de Orden y Operador de cruce OX

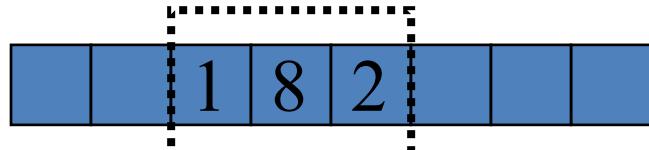
Padre 1



Padre 2



7, 3, 4, 6, 5



Orden

4, 3, 6, 7, 5

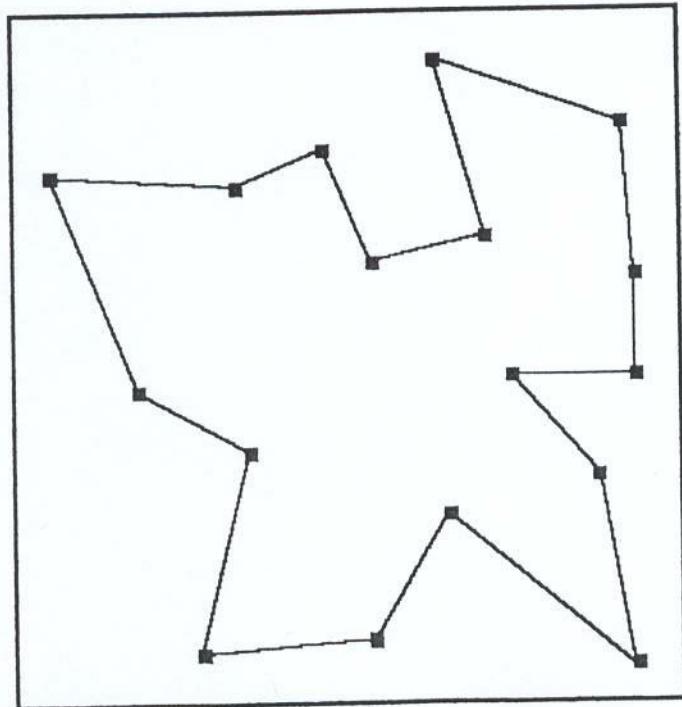
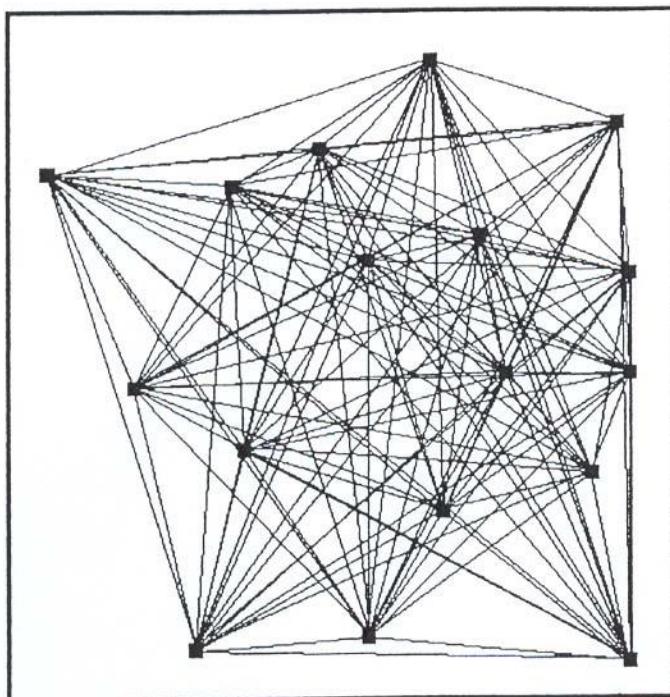
Hijo 1



4 3 6 7 5

1 8 2 4 3 6 7 5

Viajante de Comercio

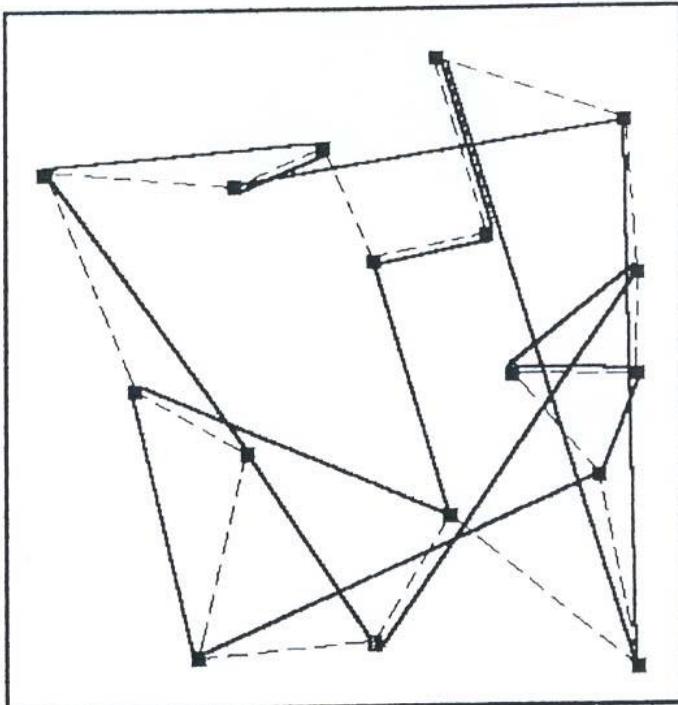


$17! = 3.5568743 \text{ e}14$ recorridos posibles

Solución óptima: 226.64

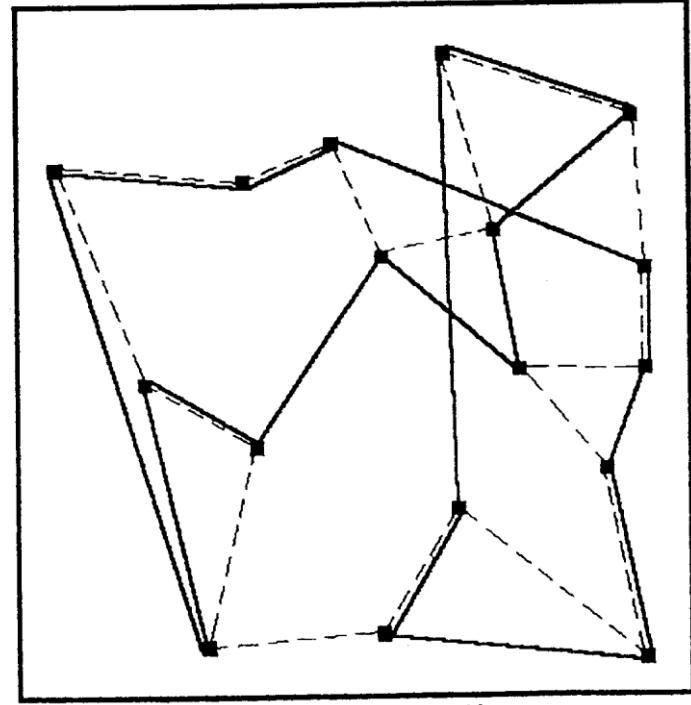
Viajante de Comercio

36 descendientes nuevos por iteración $\times 25 = 900$ descendientes, más 15 cromosomas mutados, aprox. 906 soluciones evaluadas



— Mejor solución
- - - Solución óptimal

Iteración: 0 Costo: 403.7



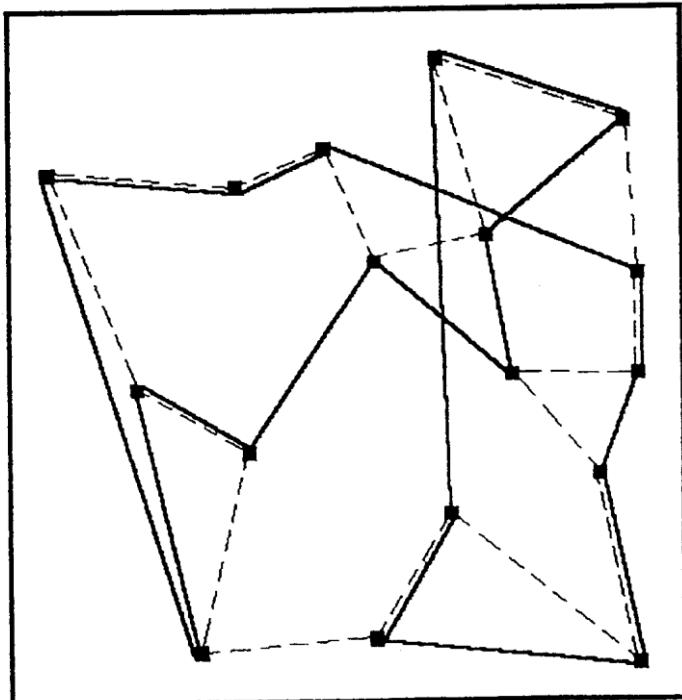
— Mejor solución
- - - Solución óptimal

Iteración: 25 Costo: 303.86

Solución óptima: 226.64

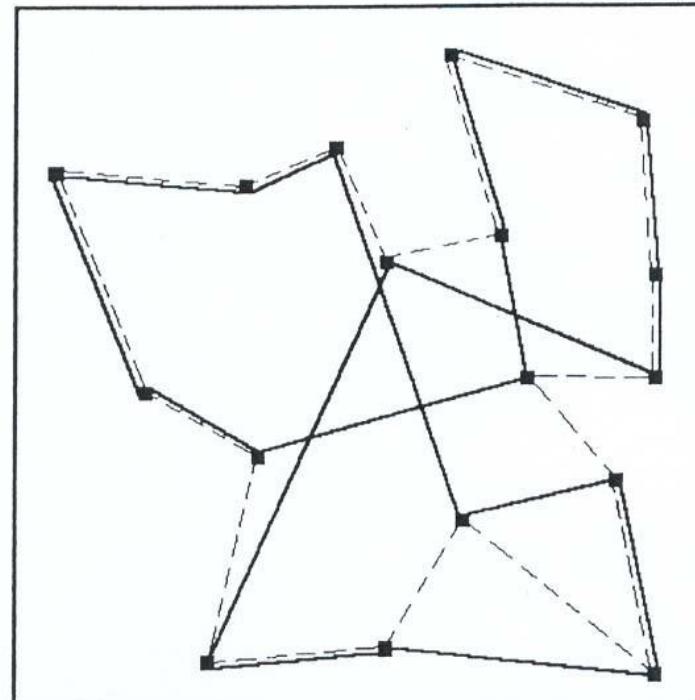
Viajante de Comercio

36 descendientes nuevos por iteración $\times 50 = 1800$ descendientes, más 30 cromosomas mutados, aprox. 1812 soluciones evaluadas



— Mejor solución
- - - Solución óptimal

Iteración: 25 Costo: 303.86



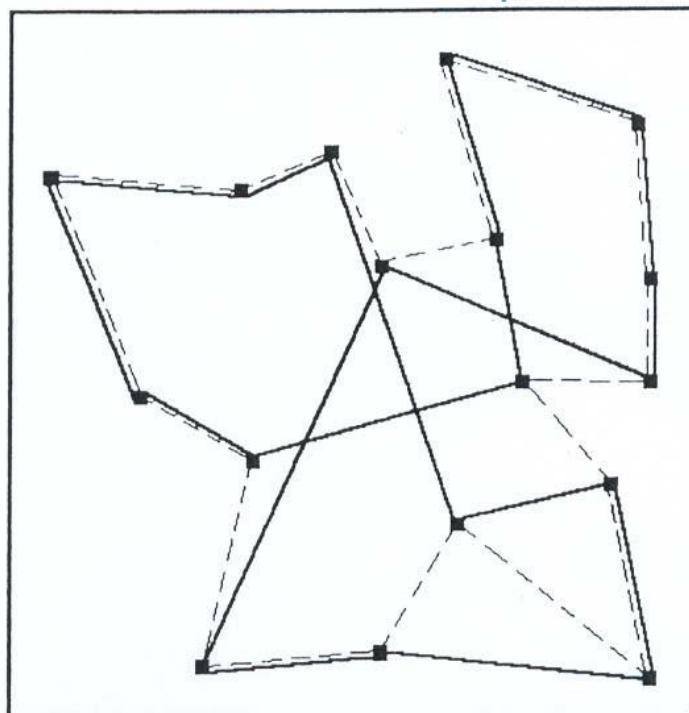
— Mejor solución
- - - Solución óptimal

Iteración: 50 Costo: 293.6

Solución óptima: 226.64

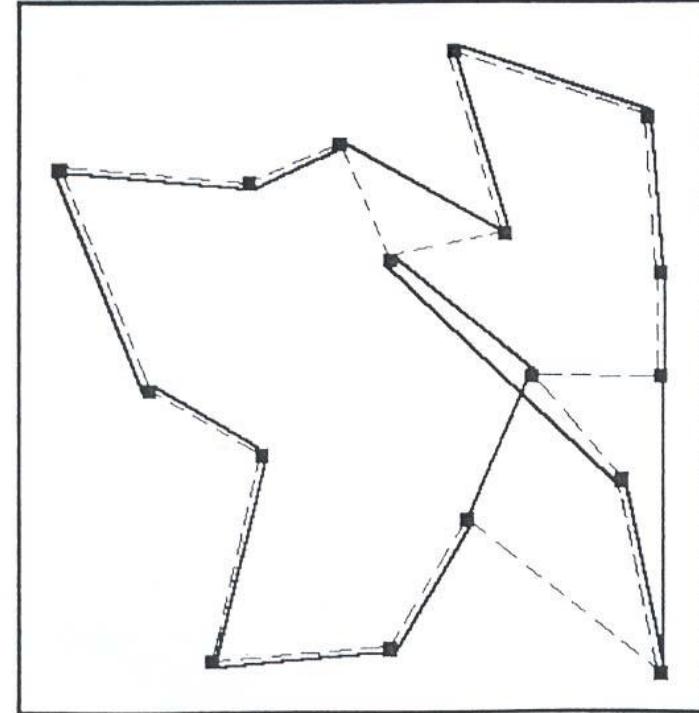
Viajante de Comercio

36 descendientes nuevos por iteración $\times 100 = 3600$ descendientes, más 60 cromosomas mutados, aprox. 3624 soluciones evaluadas



— Mejor solución
- - - Solución óptimal

Iteración: 50 Costo: 293.6



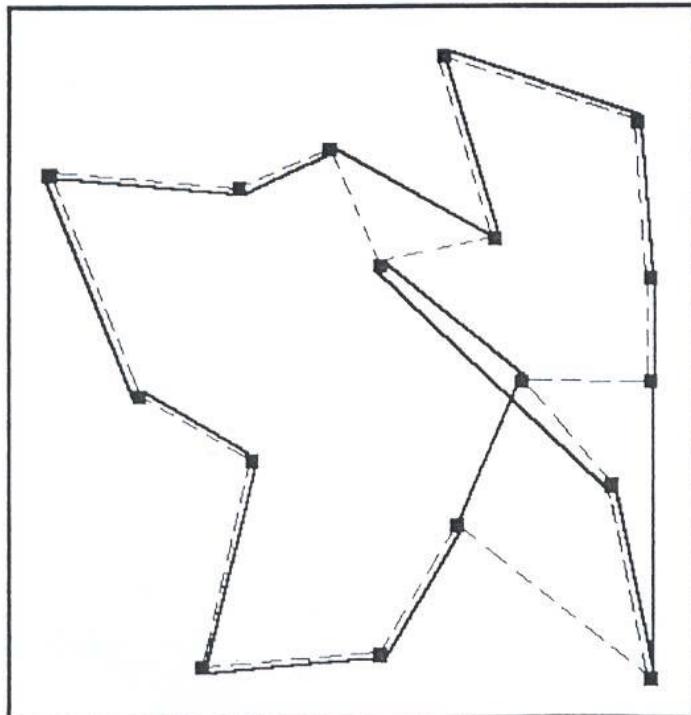
— Mejor solución
- - - Solución óptimal

Iteración: 100 Costo: 256.55

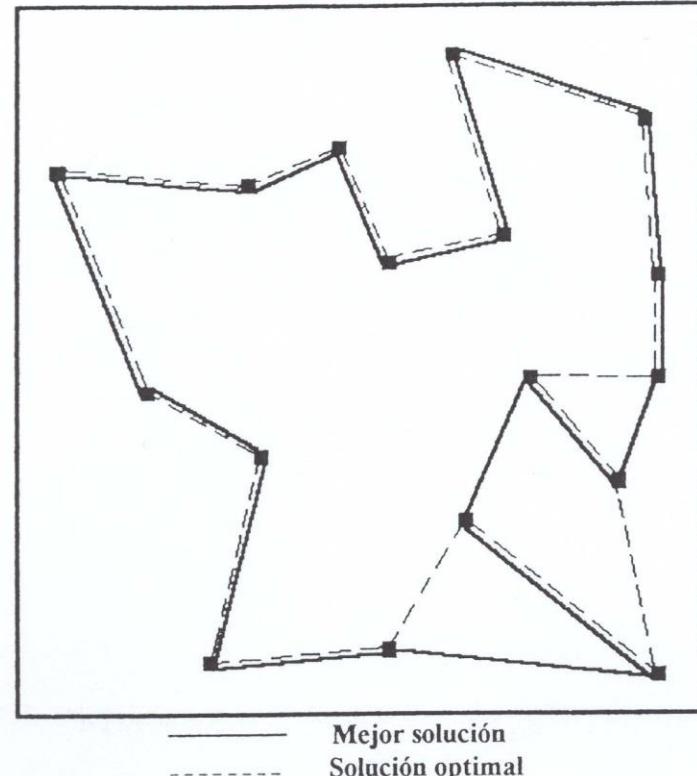
Solución óptima: 226.64

Viajante de Comercio

36 descendientes nuevos por iteración $\times 200 = 7200$ descendientes, más 120 cromosomas mutados, aprox. 7248 soluciones evaluadas



Iteración: 100 Costo: 256.55

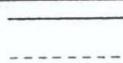
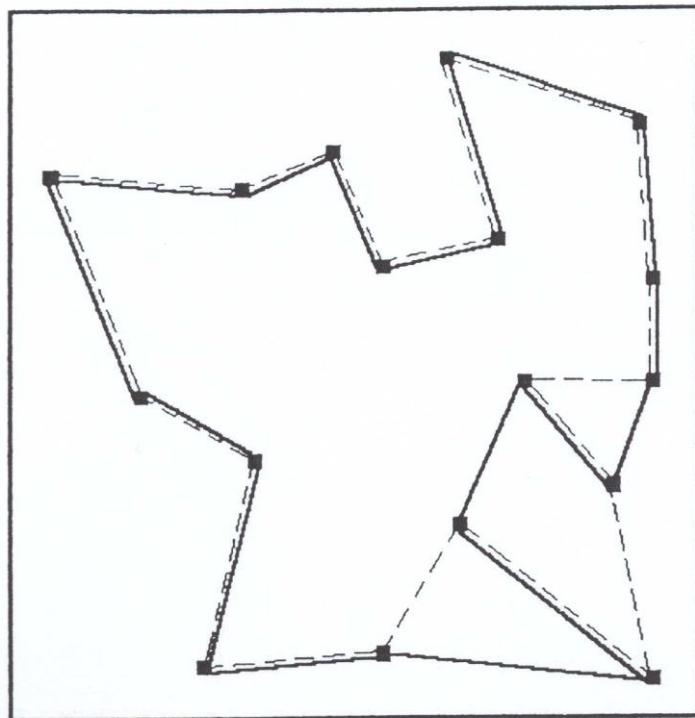


Iteración: 200 Costo: 231.4

Solución óptima: 226.64

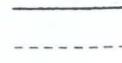
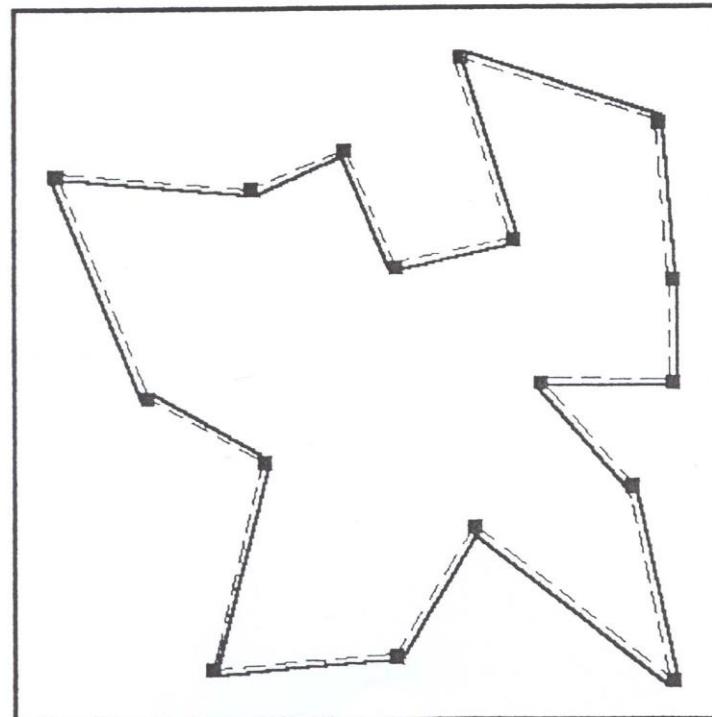
Viajante de Comercio

36 descendientes nuevos por iteración $\times 250 = 9000$ descendientes, más 60 cromosomas mutados, aprox. 9.060 soluciones evaluadas



Mejor solución
Solución óptimal

Iteración: 200 Costo: 231.4

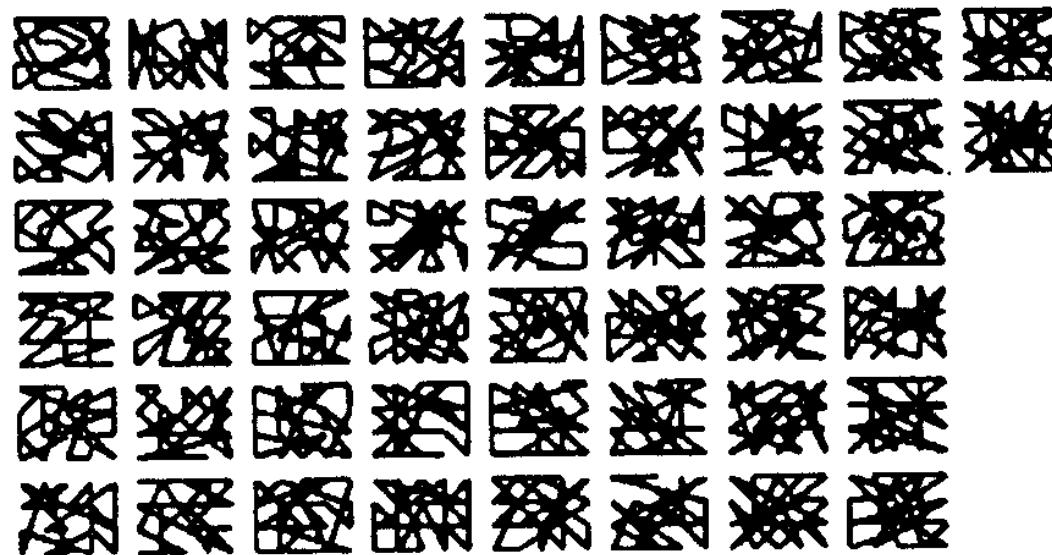


Mejor solución
Solución óptimal

Iteración: 250 Solución
óptima: 226.64

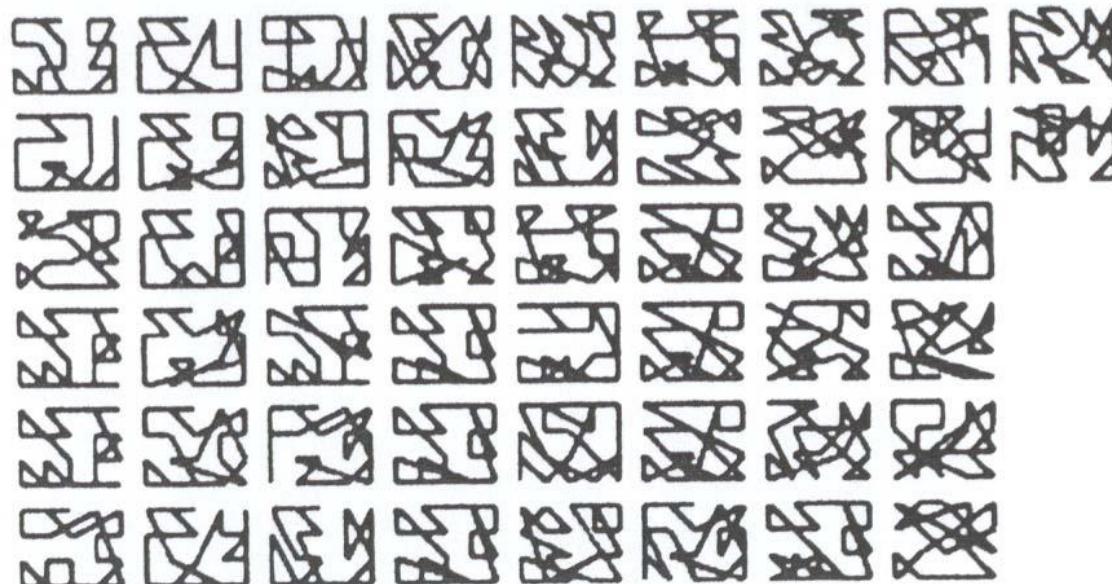
17! = 3.5568743 e14 recorridos posibles

Viajante de Comercio



(0)

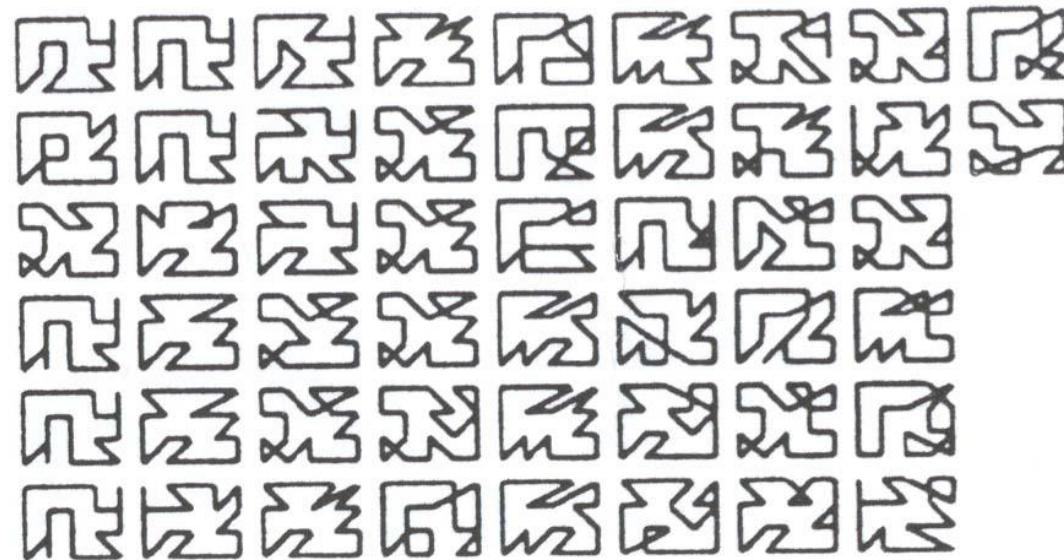
Viajante de Comercio



(10)

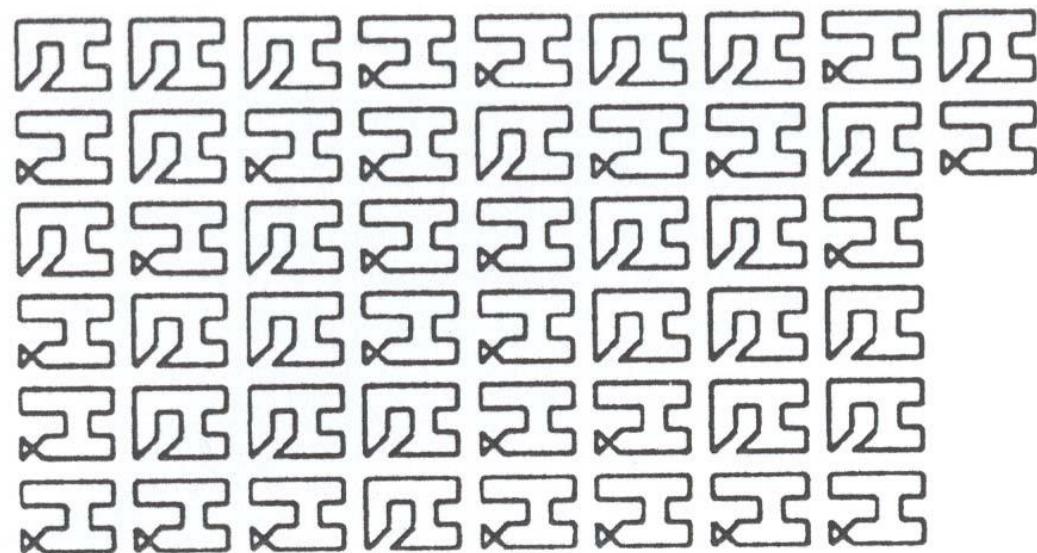
Visualización de la evolución de una población de 50 cromosomas
y 70 iteraciones

Viajante de Comercio



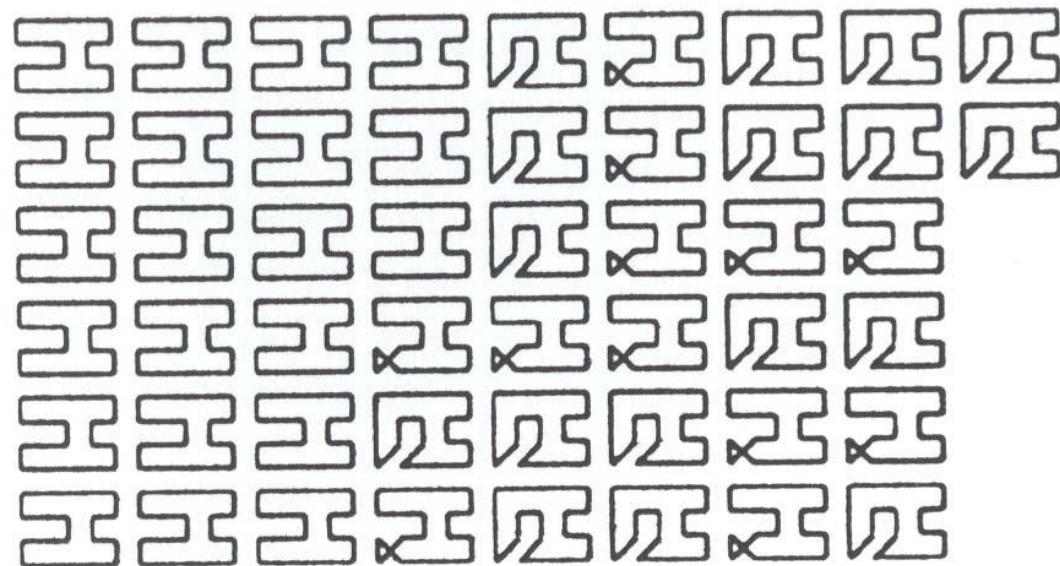
(30)

Viajante de Comercio



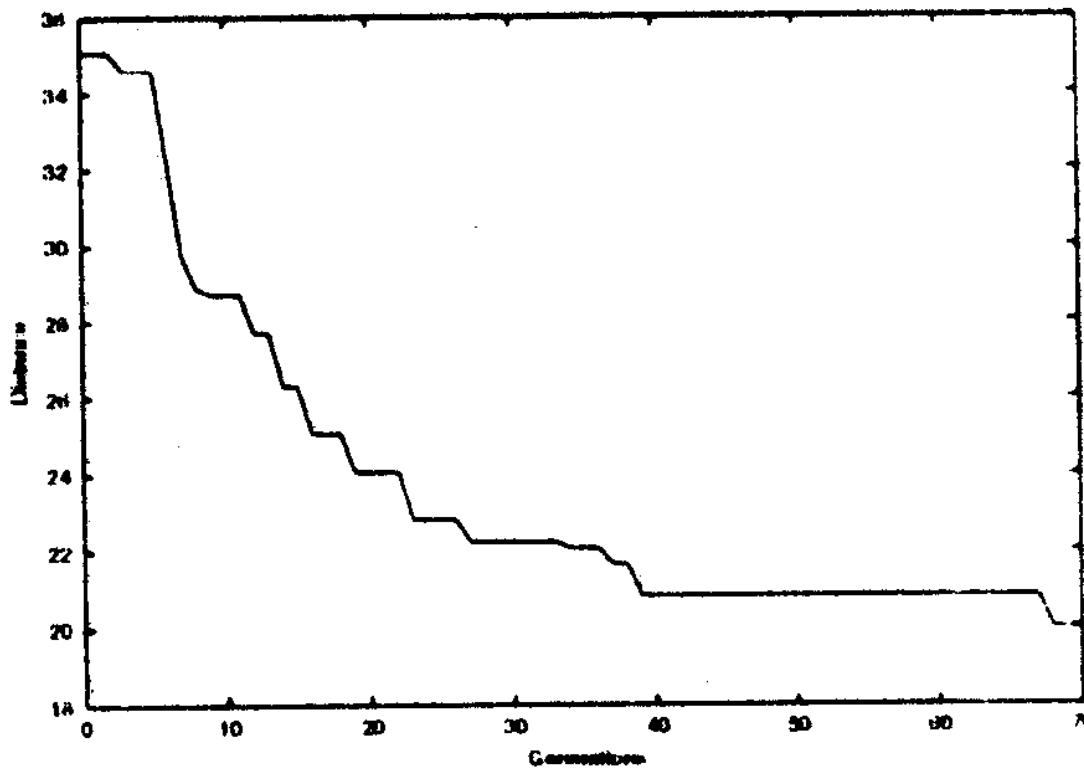
(50)

Viajante de Comercio



(70)

Viajante de Comercio



Visualización de la evolución de una población de 50 cromosomas
y 70 iteraciones

Búsqueda con información

SIMPLE
HEURISTICS
THAT MAKE US
SMART

- Heurísticas
- Métodos de escalada
- Métodos poblacionales:
 - Algoritmos genéticos
- **Búsqueda primero el mejor**



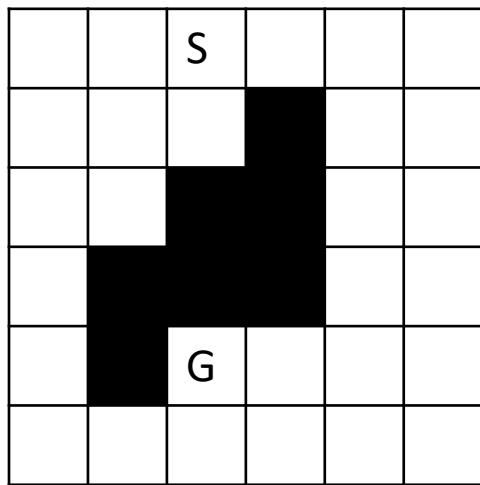
Búsqueda primero el mejor

- Búsqueda primero mejor greedy (BFS)
- Algoritmo A*
- Búsqueda dirigida

Búsqueda primero el mejor

- Modelo de búsqueda en árboles o grafos con una cola con prioridad para almacenar los nodos a visitar/expandir.
- Una función de evaluación $f(n)$ para seleccionar el nodo a expandir, y los prioriza de acuerdo a esta función de evaluación

Heurísticas para este problema



Solo movimientos en horizontal y vertical.

Búsqueda primero mejor greedy (BFS)

Trata de expandir el nodo más cercano al objetivo, alegando que probablemente conduzca rápidamente a una solución.

Se aplica a problemas donde se tiene una evaluación de distancia a la meta/objetivo.

Se introduce una función de evaluación $f(n) = h(n)$ que mide la distancia a la meta a alcanzar (con una cota optimista de distancia)

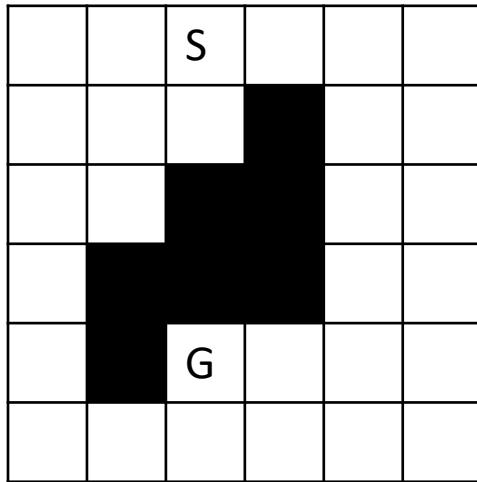
Si no hubiera
obstáculos

Ejemplos:

Calcular el recorrido en un mapa con obstáculos

Calcular el recorrido hacia Budapest.

Ejemplo



- Solo movimientos en horizontal y vertical.
- Heurística: distancia manhattan.
- Suma de la distancia vertical y horizontal medida en celdas

Ejemplo

	5	S	5	6	
6	4	3		5	
4	3			4	
3				3	
2		G	1	2	
3	2	1			

- Calculemos el valor heurístico de algunos nodos
- ¿Cuál es el mejor camino?

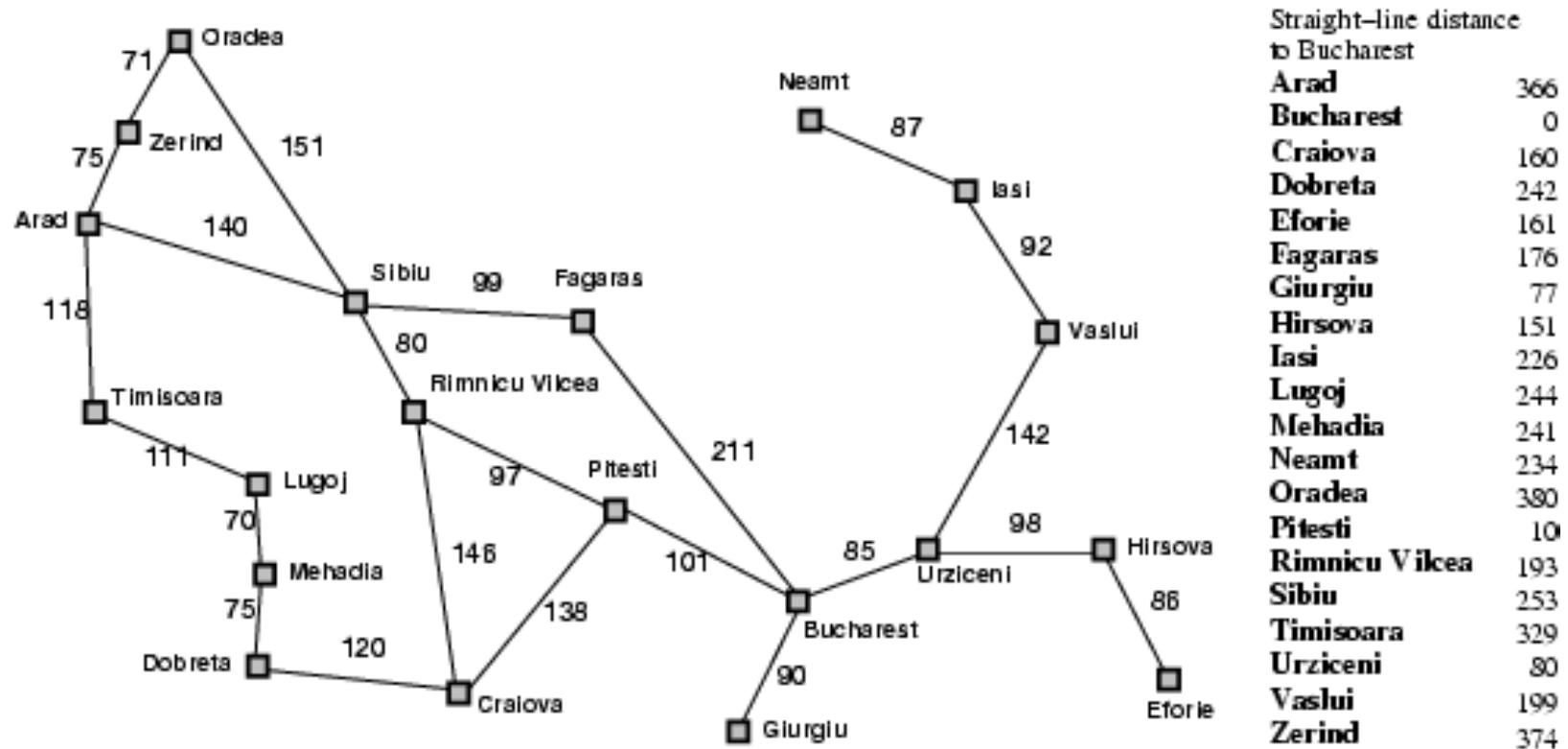
Se toma la casilla adyacente
de menor valor

Ejemplo

	5	S	5	6	
6	4	3		5	
4	3			4	
3				3	
2		G	1	2	
3	2	1			

- ¿Cómo funcionaría BFS greedy? ¿Y un método de escalada?
- ¿Es el camino óptimo?

Mapa de carreteras



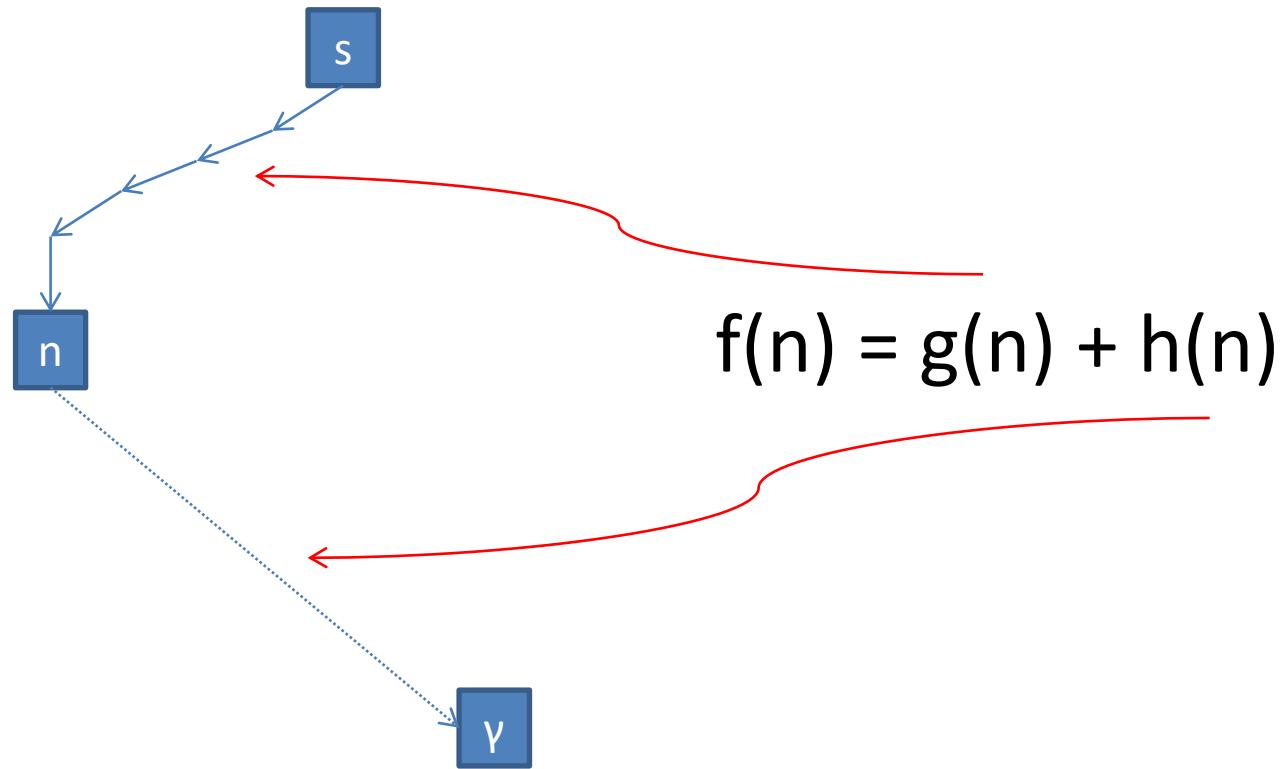
Algoritmo A*

- Estrategia de búsqueda sobre grafos de la familia búsqueda primero el mejor, evalúa los nodos combinando información heurística e información de coste del nodo
- Heurística: $f(n) = g(n) + h(n)$

$g(n)$ da el coste del camino desde el nodo inicial al nodo n
 $h(n)$ el coste estimado del camino más barato desde n al objetivo

$f(n)$ = coste más barato estimado de la solución a través de n

Heurística para el algoritmo A*



Algoritmo A*

- Estructura de un nodo:

estado	hijos	mejor_padre	g
			h

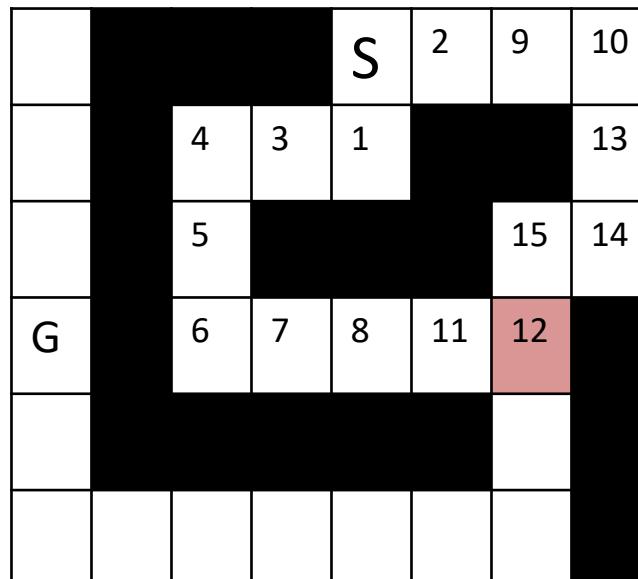
- Búsqueda en grafos donde abiertos es una cola con prioridad ordenada de acuerdo a $f(n)$.

Algoritmo A*

- ABIERTOS contiene el nodo inicial, CERRADOS (nodos visitados) esta vacío
- Comienza un ciclo que se repite hasta que se encuentra solución o hasta que ABIERTOS queda vacío
 - Seleccionar el mejor nodo de ABIERTOS
 - Si es un nodo objetivo terminar
 - En otro caso se expande dicho nodo
 - Para cada uno de los nodos sucesores
 - Si está en ABIERTOS insertarlo manteniendo la información del mejor parent
 - Si está en CERRADOS insertarlo manteniendo la información del mejor parent y actualizar la información de los descendientes
 - En otro caso, insertarlo como un nodo nuevo

Nodos repetidos en abiertos

- Si un nodo n ya existe en abiertos hay que comprobar si el nuevo camino es mejor que el anterior.



El número indica el orden de generación de los nodos

Nodos repetidos en cerrados

- n = un nodo en cerrados.
- caso 1: el nuevo padre de n no es mejor que el padre anterior. FIN
- caso 2: el nuevo padre de n es mejor que el padre anterior.
 - Actualizar enlace al padre, actualizar el nuevo valor de coste del camino. **Propagar la información a los hijos de n**

A* puede encontrar óptimo en el problema anterior

- A* es óptima si $h(n)$ es una heurística admisible, es decir, que $h(n)$ nunca sobreestime el coste de alcanzar el objetivo.
Ejemplo: Distancia en línea recta a Bucarest y el ejemplo del mapa

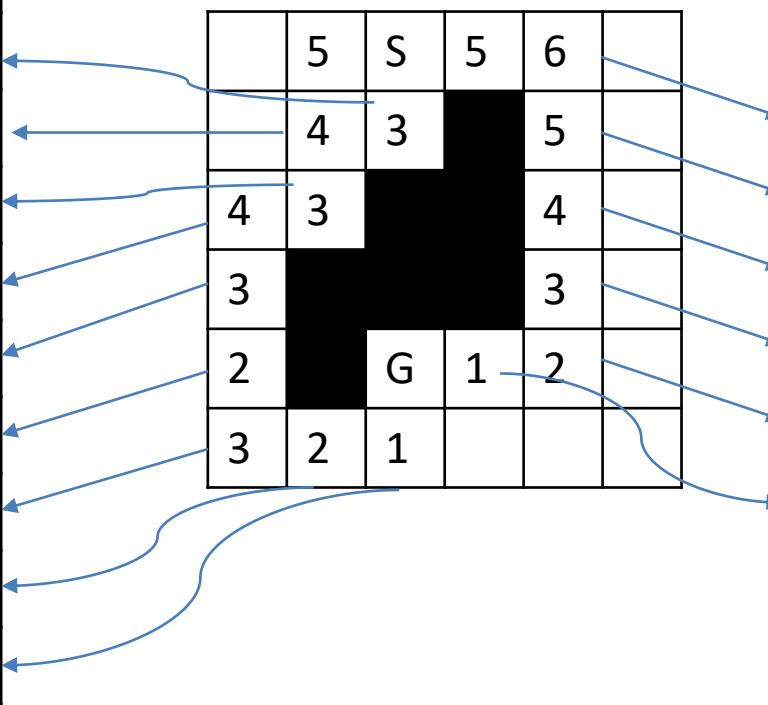
A* puede encontrar óptimo en el problema anterior

	5	S	5	6	
6	4	3		5	
4	3			4	
3				3	
2		G	1	2	
3	2	1			

- Para cada nodo n , A* tiene en cuenta la **estimación**, $h(n)$, al objetivo y el **coste real**, $g(n)$ desde el inicio
- $f(n) = g(n) + h(n)$
- A* es óptima si $h(n)$ es una heurística admisible, es decir, que $h(n)$ nunca sobreestime el coste de alcanzar el objetivo.
Ejemplo: Distancia en línea recta a Bucarest

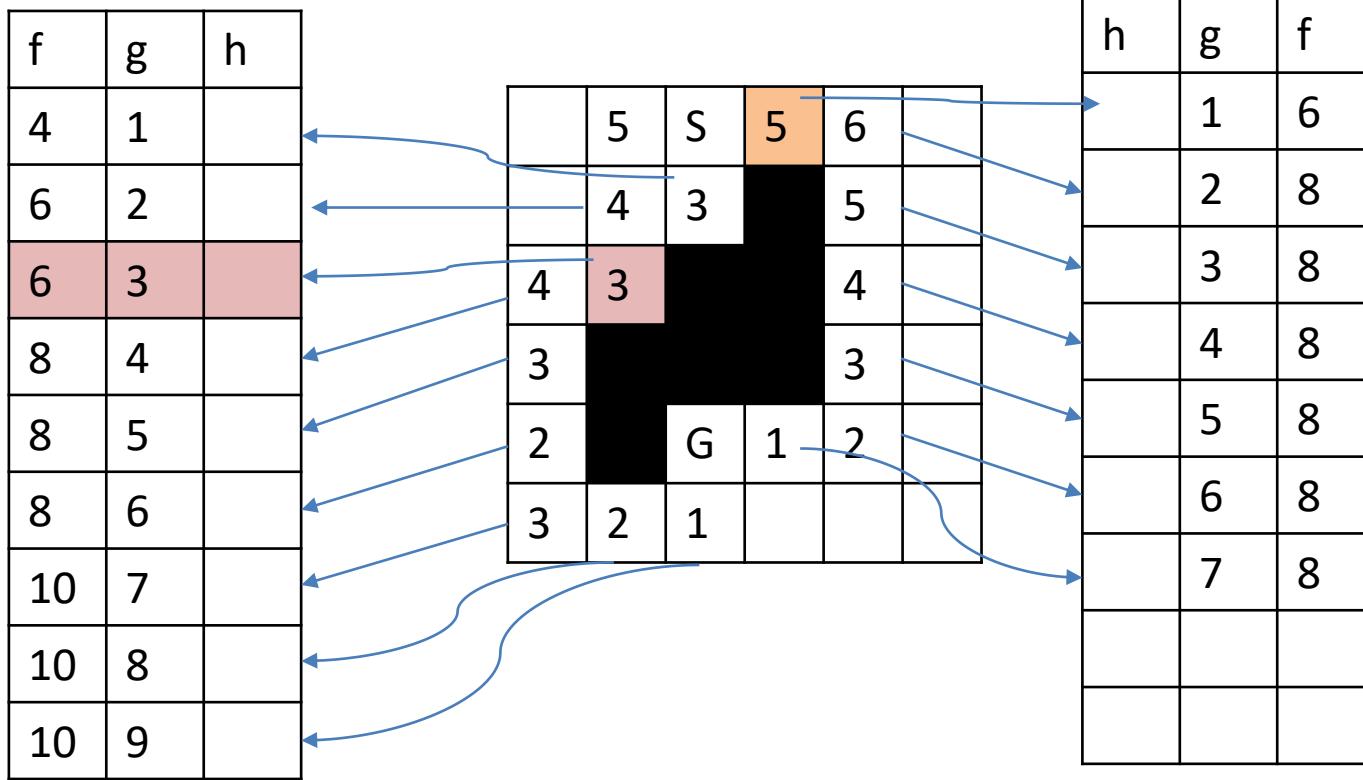
Sigamos la traza del algoritmo

f	g	h
4	1	
6	2	
6	3	
8	4	
8	5	
8	6	
10	7	
10	8	
10	9	

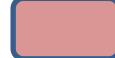


h	g	f
	1	6
	2	8
	3	8
	4	8
	5	8
	6	8
	7	8

Observar



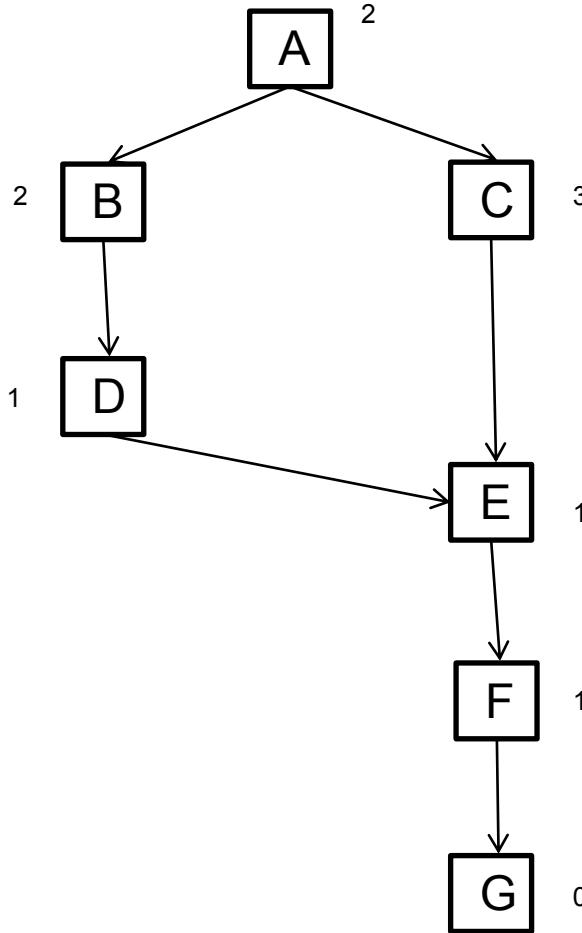
Observar

- A* detecta que el camino inicial no tiene por qué ser el mejor cuando el nodo  genera su hijo y éste es peor que el nodo 
- Toda la descendencia de  no empeora (empatan) respecto al hijo de 
- Usamos como criterio de desempate el nodo más joven.

Casos particulares del algoritmo A*

- Supongamos que usamos el algoritmo A* pero tomamos siempre $h(n)=0$, entonces
 - En el caso que el coste de cada arco sea siempre unidad, el algoritmo se comporta como la búsqueda en anchura.
 - En otro caso, el algoritmo se comporta como la búsqueda de coste uniforme.
- Supongamos que usamos el algoritmo A* pero tomamos siempre $g(n)=0$, entonces el algoritmo se comporta como el algoritmo de búsqueda primero el mejor greedy.

Ejemplo



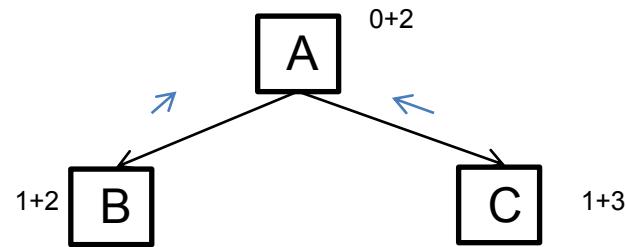
El coste es
unidad por arco,
la heurística h
esta reflejada en
cada nodo

Ejemplo



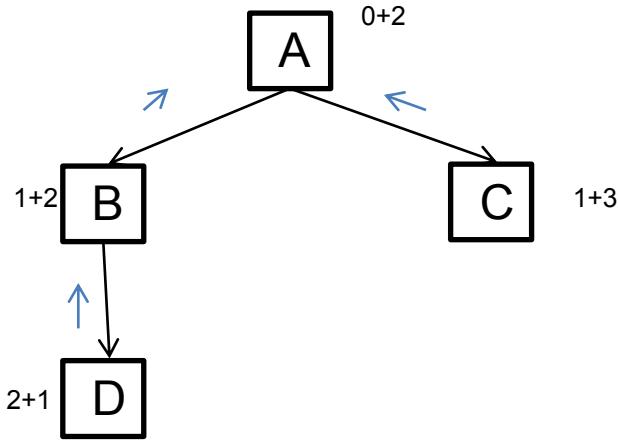
Sobre cada nodo
se refleja ahora
 $f=g+h$

Ejemplo



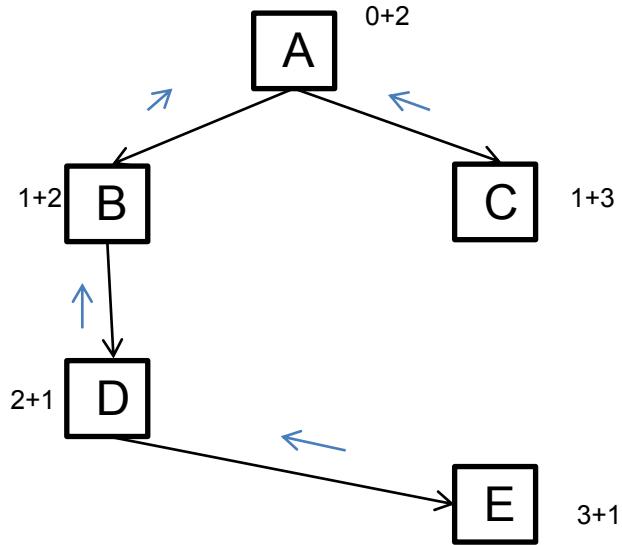
Sobre cada nodo
se refleja ahora
 $f=g+h$

Ejemplo



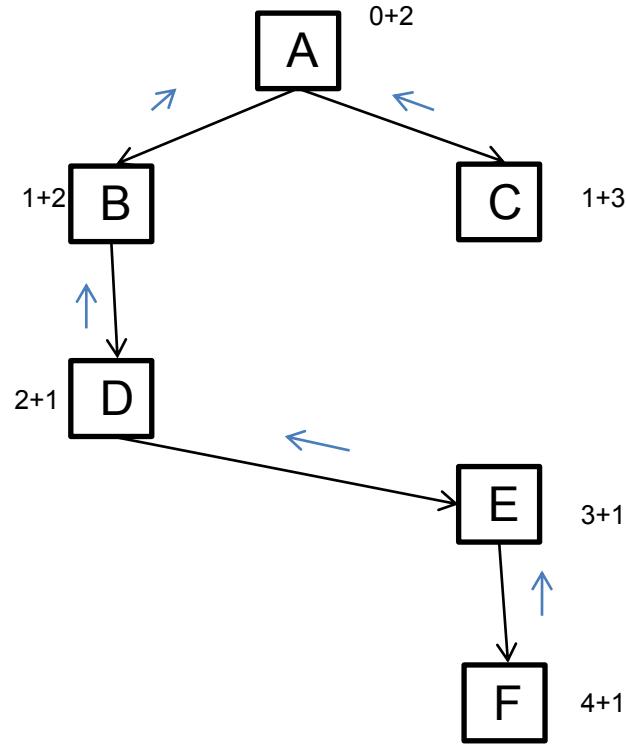
Sobre cada nodo se refleja ahora
 $f=g+h$

Ejemplo



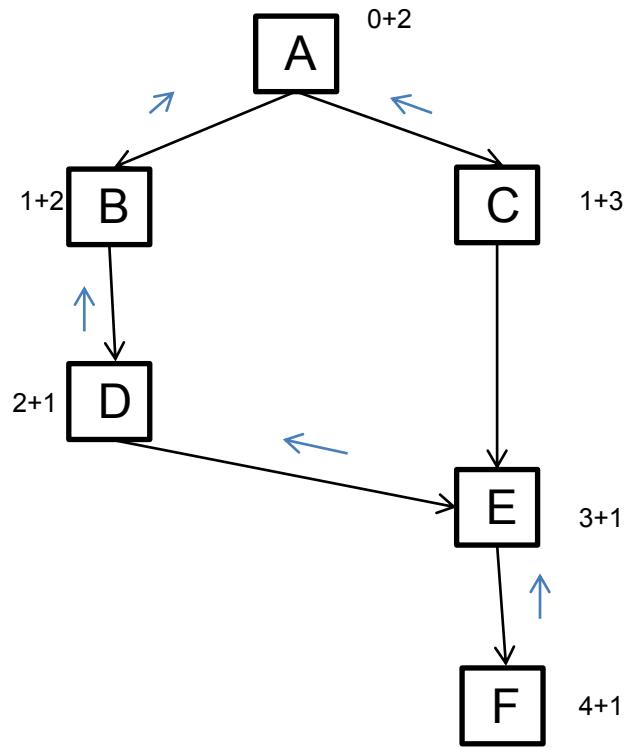
Sobre cada nodo se refleja ahora
 $f=g+h$

Ejemplo



Sobre cada nodo se refleja ahora
 $f=g+h$

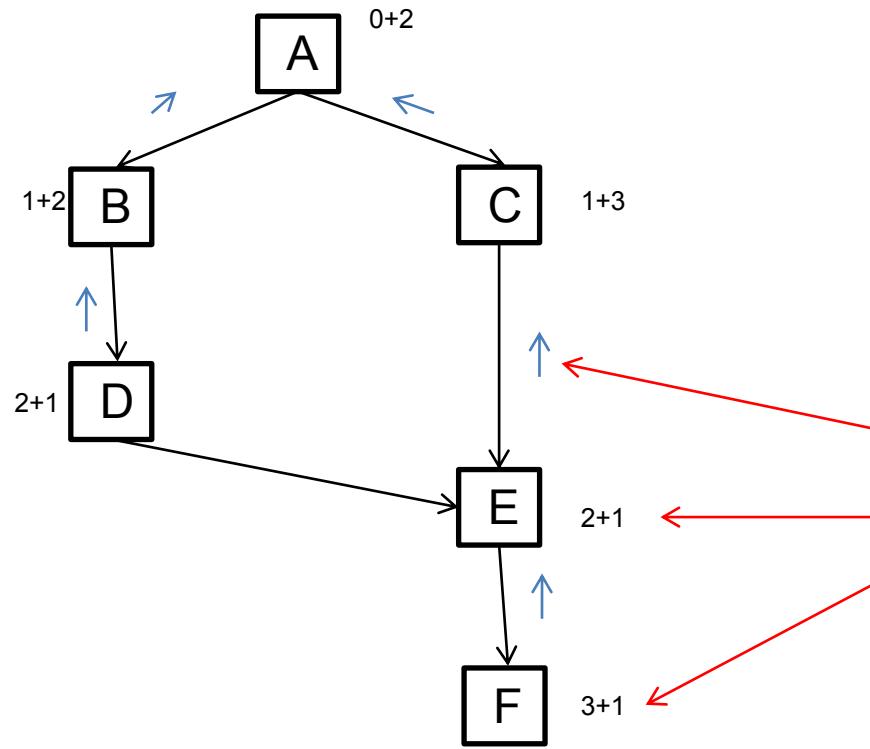
Ejemplo



Sobre cada nodo se refleja ahora $f=g+h$

E está en CERRADOS, hemos obtenido un mejor parent, hacemos cambios y propagamos

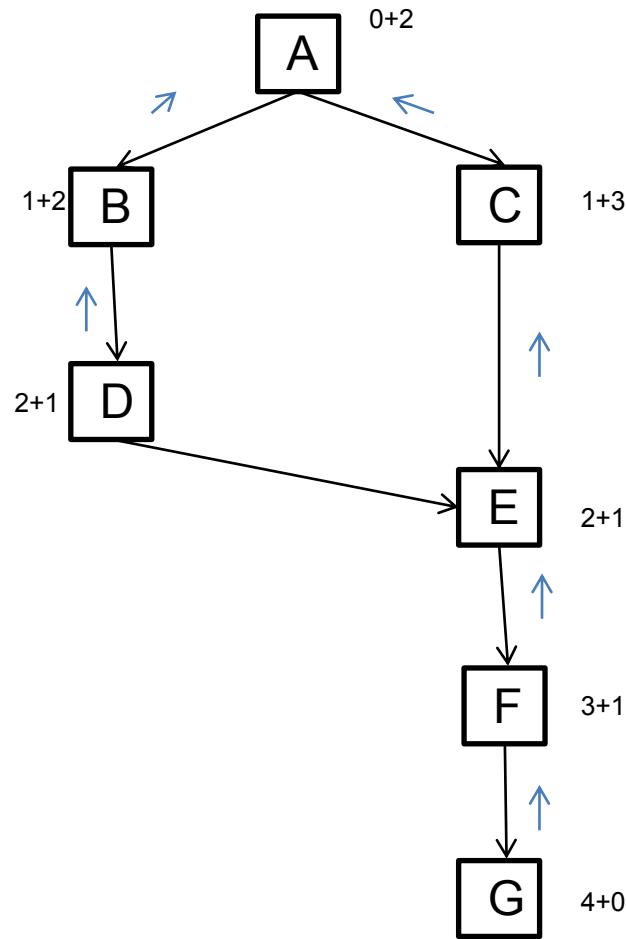
Ejemplo



Sobre cada nodo se refleja ahora $f=g+h$

Vemos como cambia enlace a mejor parente y coste de E y F

Ejemplo



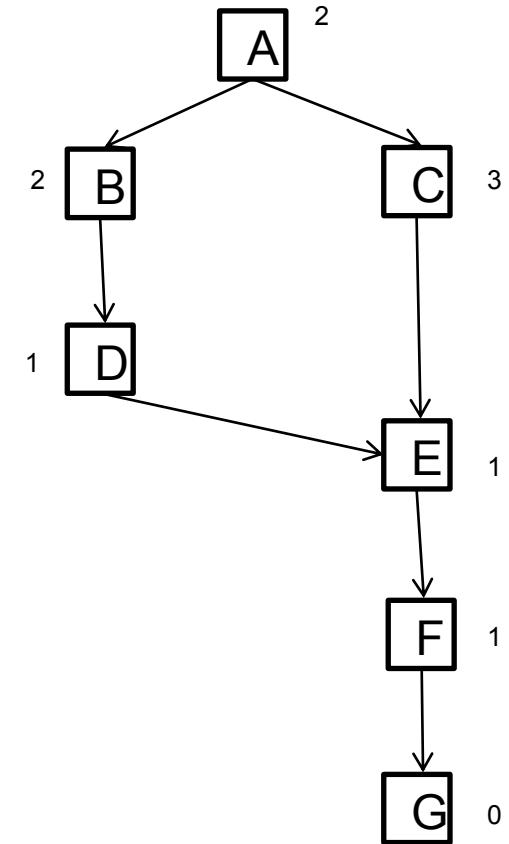
Sobre cada nodo se refleja ahora $f=g+h$

Genera hijo de F, en la siguiente iteración extrae el mejor nodo que es G y el algoritmo termina

Nueva resolución

ABIERTOS
(A,-,0,2)

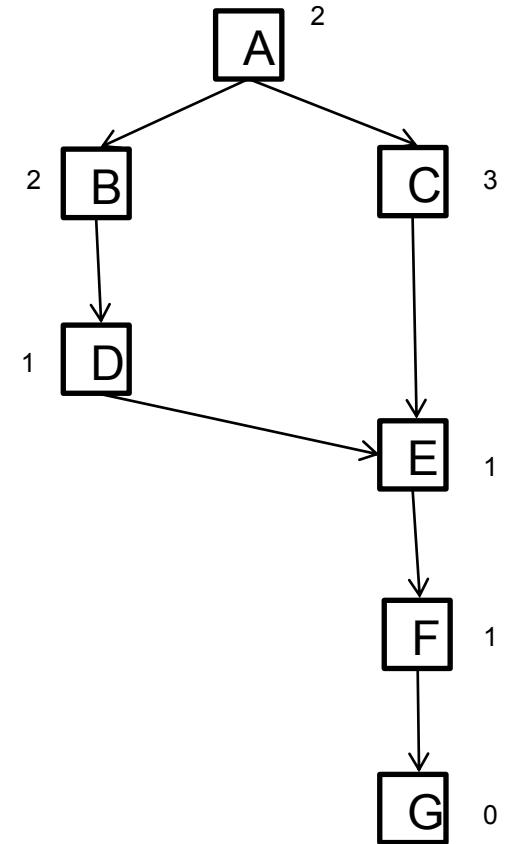
CERRADOS



Nueva resolución

ABIERTOS
 $(B, A, 1, 2)$
 $(C, A, 1, 3)$

CERRADOS
 $(A, -, 0, 2)$



Nueva resolución

ABIERTOS

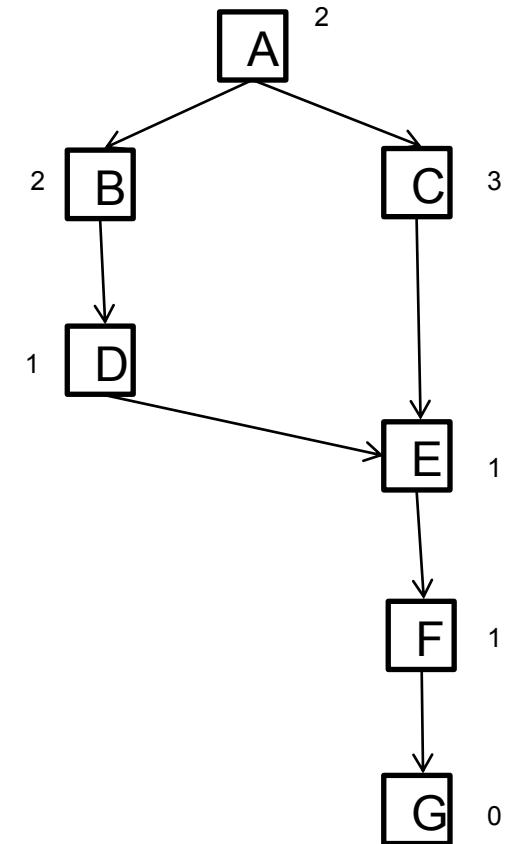
(C,A,1,3)

(D,B,2,1)

CERRADOS

(A,-,0,2)

(B,A,1,2)



Nueva resolución

ABIERTOS

(C,A,1,3)

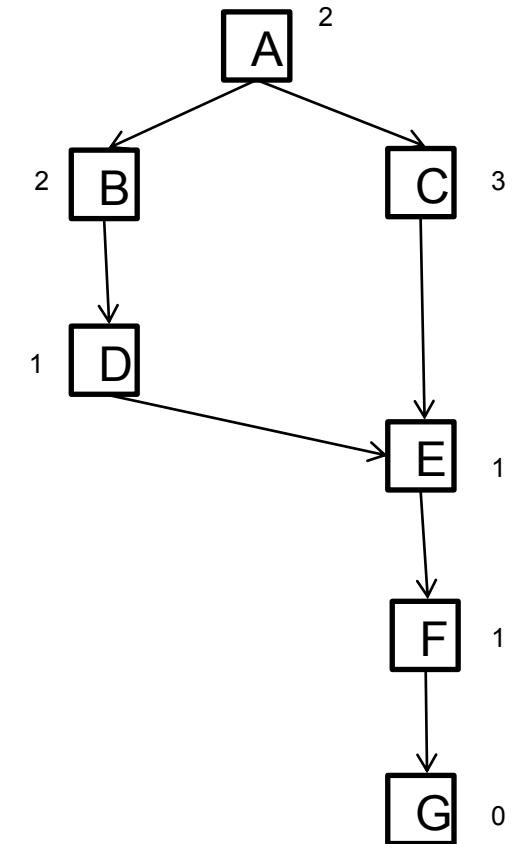
(E,D,3,1)

CERRADOS

(A,-,0,2)

(B,A,1,2)

(D,B,2,1)



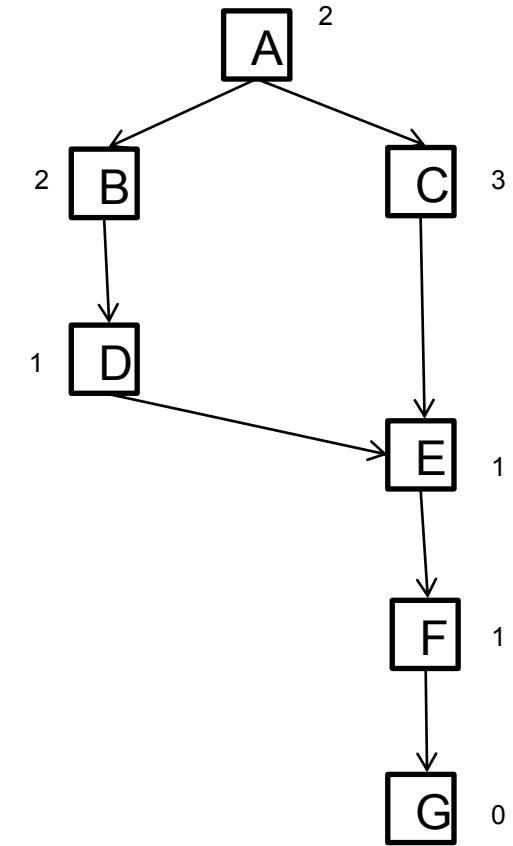
Nueva resolución

ABIERTOS

(C,A,1,3)
(F,E,4,1)

CERRADOS

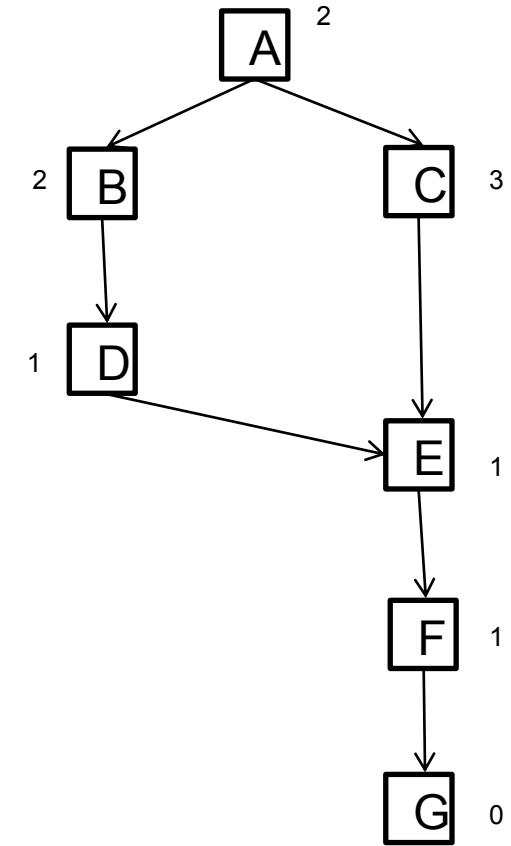
(A,-,0,2)
(B,A,1,2)
(D,B,2,1)
(E,D,3,1)



Nueva resolución

ABIERTOS
(F,E,4,1)

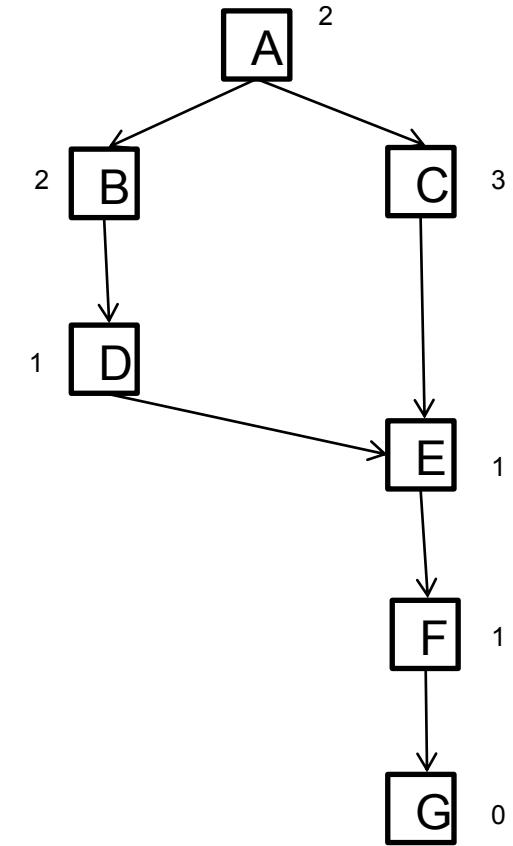
CERRADOS
(A,-,0,2)
(B,A,1,2)
(D,B,2,1)
(E,D,3,1)
(C,A,1,3)



Nueva resolución

ABIERTOS
(F,E,4,1)

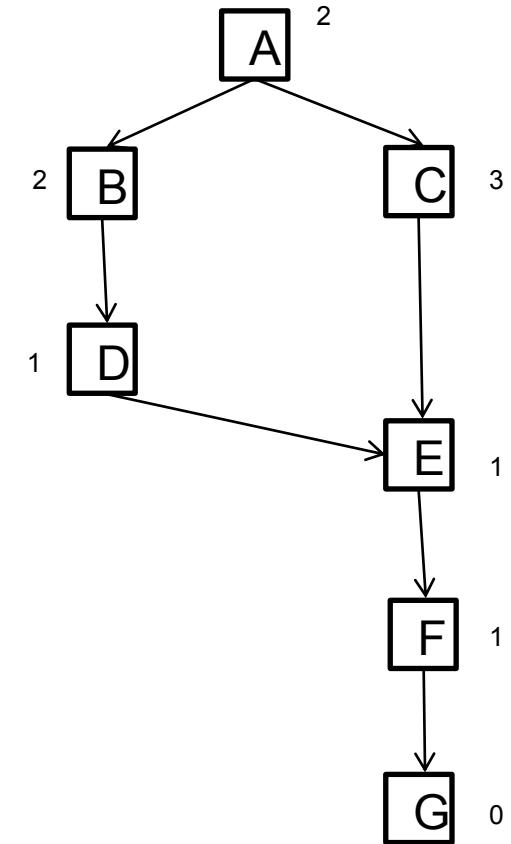
CERRADOS
(A,-,0,2)
(B,A,1,2)
(D,B,2,1)
(E,C,2,1)
(C,A,1,3)



Nueva resolución

ABIERTOS
(F,E,3,1)

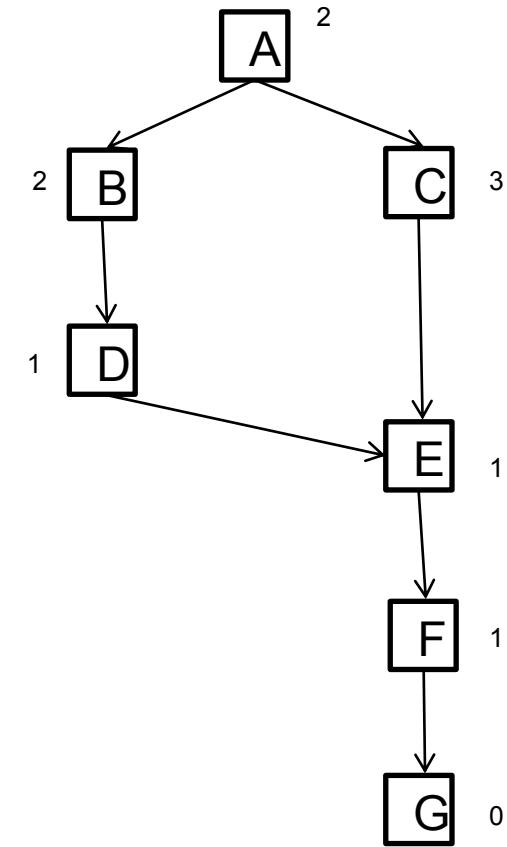
CERRADOS
(A,-,0,2)
(B,A,1,2)
(D,B,2,1)
(E,C,2,1)
(C,A,1,3)



Nueva resolución

ABIERTOS
(G,F,4,0)

CERRADOS
(A,-,0,2)
(B,A,1,2)
(D,B,2,1)
(E,C,2,1)
(C,A,1,3)
(F,E,3,1)



Nueva resolución

ABIERTOS

CERRADOS

(A,-,0,2)

(B,A,1,2)

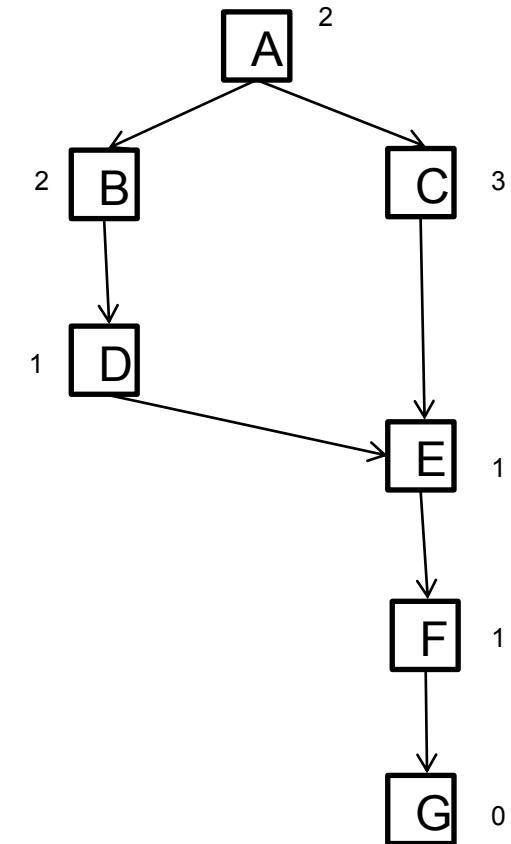
(D,B,2,1)

(E,C,2,1)

(C,A,1,3)

(F,E,3,1)

(G,F,4,0)



Nueva resolución

ABIERTOS

CERRADOS

(A,-,0,2)

(B,A,1,2)

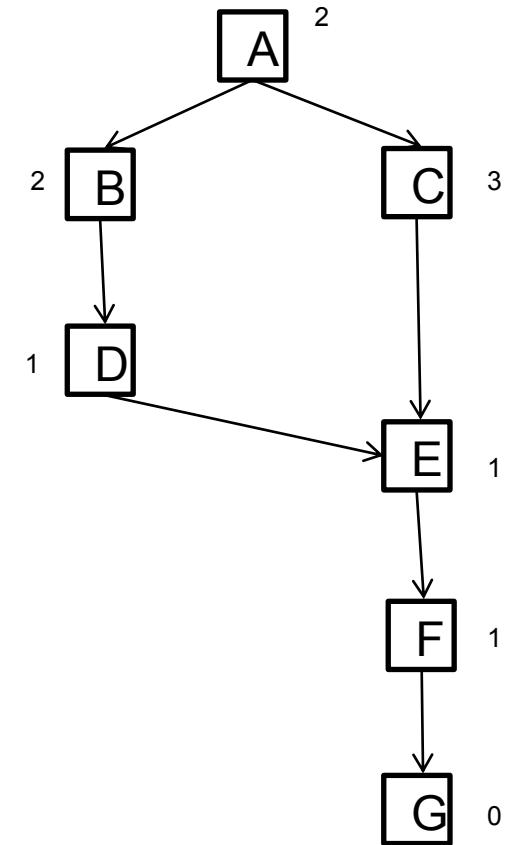
(D,B,2,1)

(E,C,2,1)

(C,A,1,3)

(F,E,3,1)

(G,F,4,0)



Características

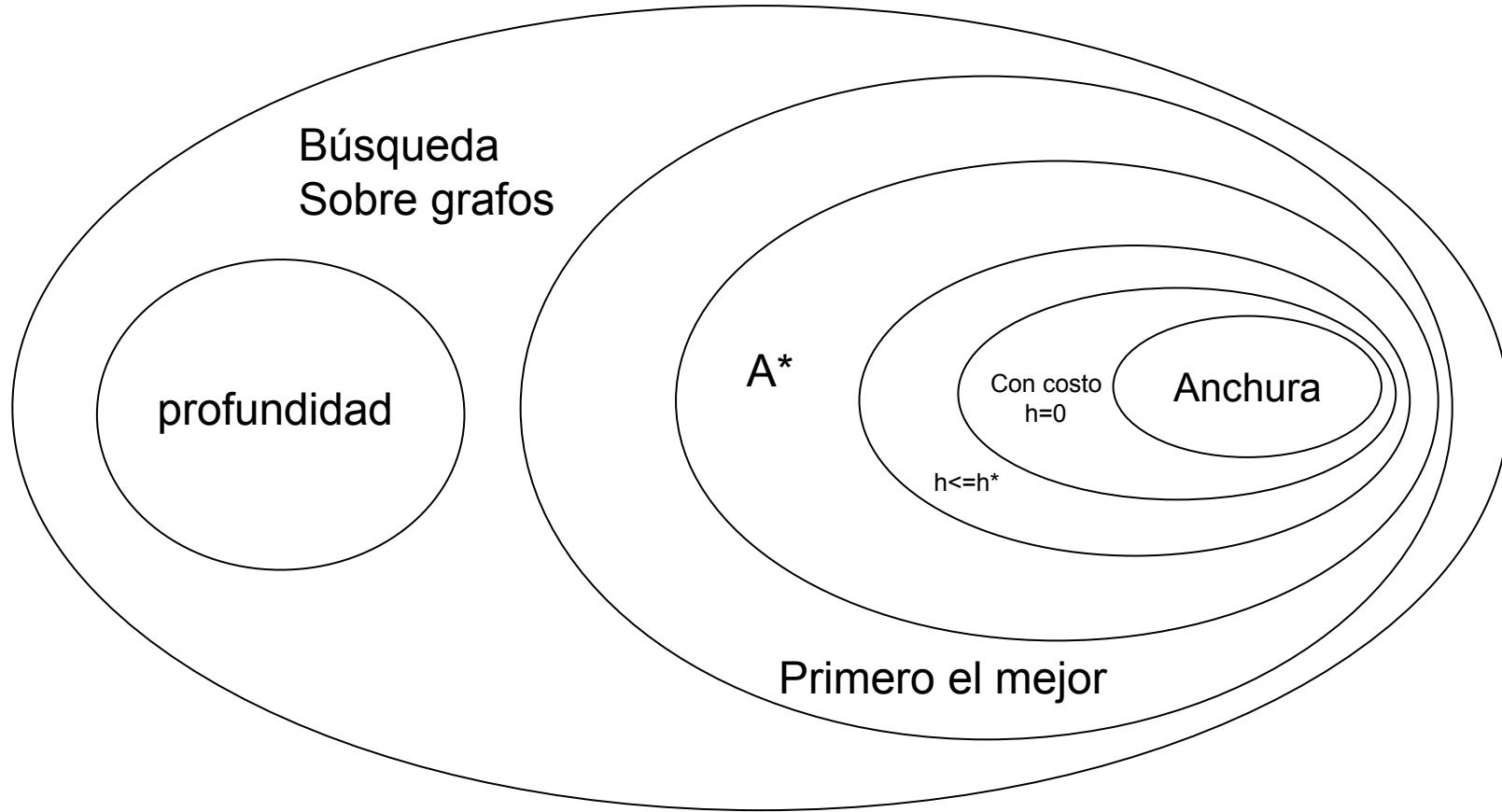
- **Compleitud:** si existe solución, la encuentra bajo condiciones muy generales.
- **Admisibilidad:** si hay una solución óptima, bajo unas condiciones muy generales y si

La función $h(n)$ es admisible: $h(n) \leq h^*(n)$

Búsqueda Dirigida (Beam Search)

- Una variación del algoritmo A* que limita el factor de ramificación en problemas complejos.
- Cada vez que se expande un nodo, se generan sus sucesores, se evalúan con la función heurística f , y se eliminan aquellos sucesores con peor valor de la f , quedándonos con un número fijo de sucesores.

Algoritmos de búsqueda



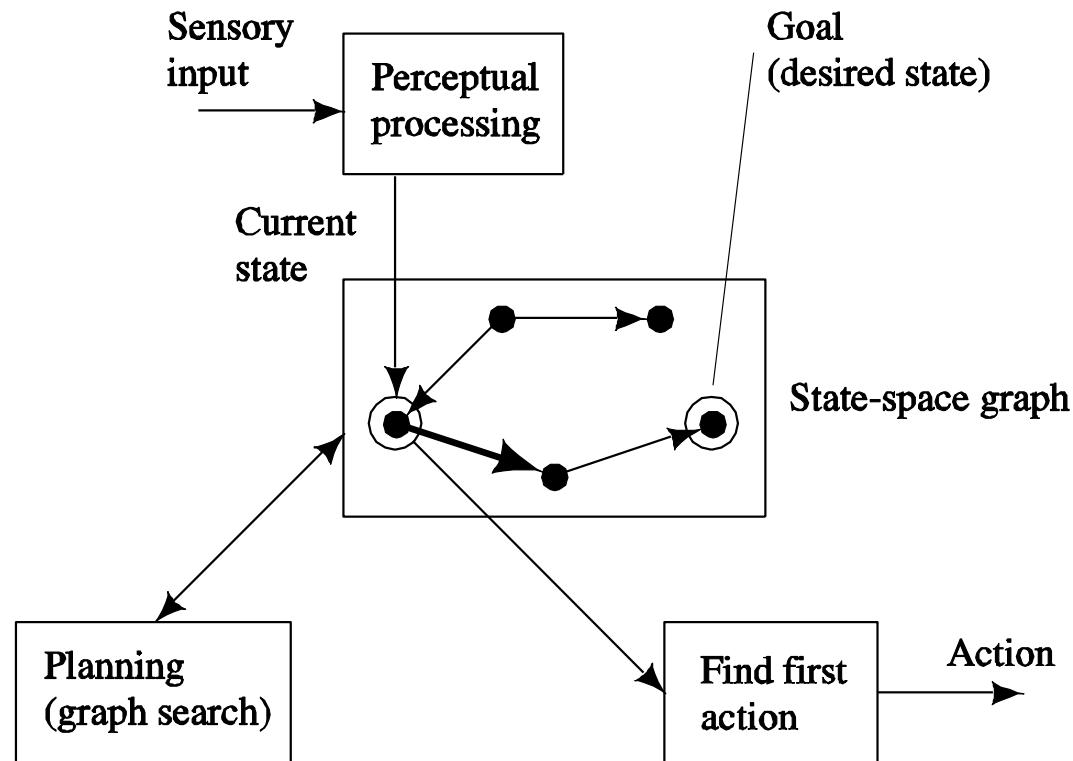
Dificultades del proceso

- Los procesos de percepción no siempre pueden obtener la información necesaria acerca del **estado** del entorno
- Las acciones pueden no disponer siempre de modelos de sus **efectos**
- Pueden haber otros procesos físicos, u **otros agentes**, en el mundo

Dificultades del proceso

- En el tiempo que transcurre desde la construcción de un plan, el mundo puede cambiar de tal manera que el plan ya no sea adecuado
- Podría suceder que se le requiriese al agente actuar antes de que pudiese completar una búsqueda de un estado objetivo
- Aunque el agente dispusiera de tiempo suficiente, sus recursos de memoria podrían no permitirle realizar la búsqueda de un estado objetivo

Arquitectura percepción/planificación/actuación



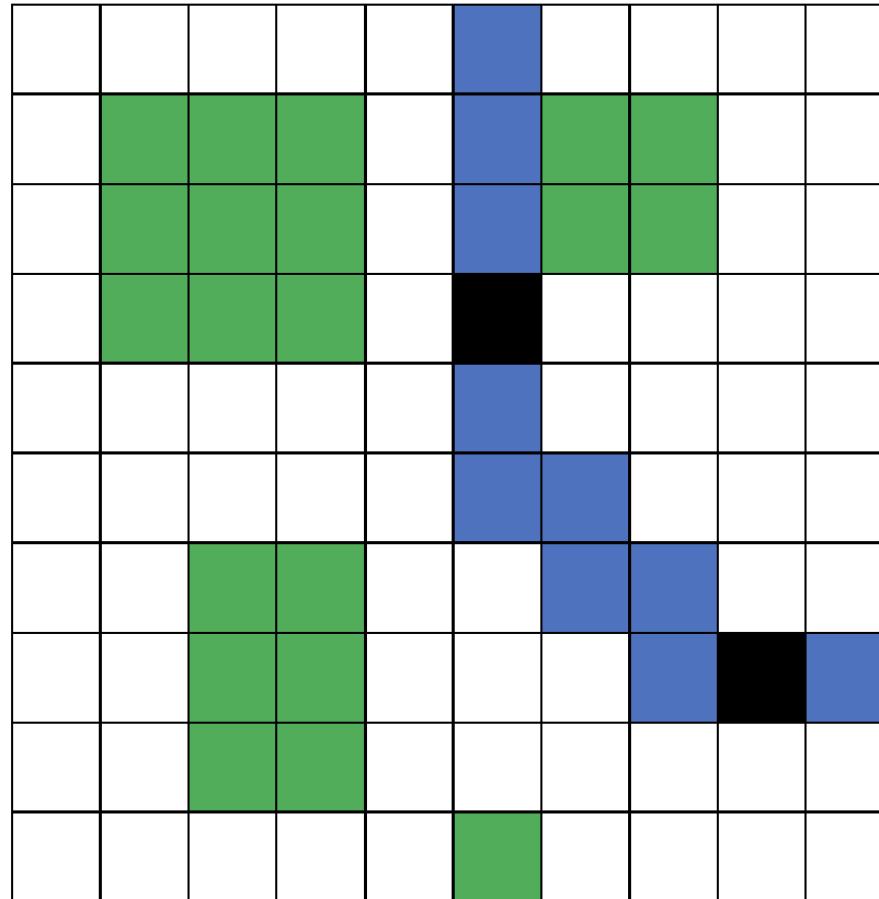
© 1998 Morgan Kaufman Publishers

Otro modelo

PLAN:{Acción_1, Acción_2, ..., Acción_n}

- Se selecciona una acción del plan y se elimina
 - Si la acción se puede aplicar se hace
 - En otro caso se intenta
 - reparar el plan o
 - Replanificar.
- Se elige la siguiente acción

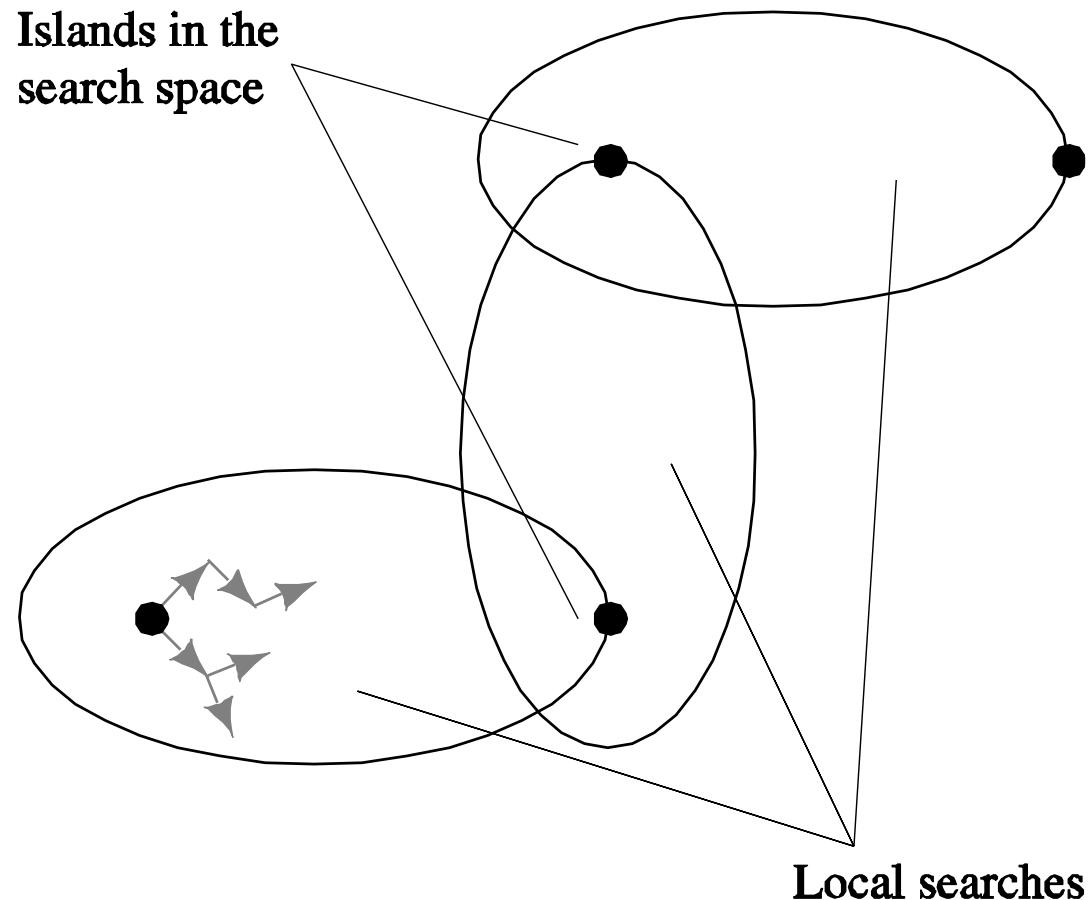
Ejemplo de funcionamiento



Heurísticas sobre el proceso de búsqueda

- Búsqueda orientada a subobjetivos
- Búsqueda con horizonte
- Búsqueda jerárquica

Búsqueda orientada a subobjetivos

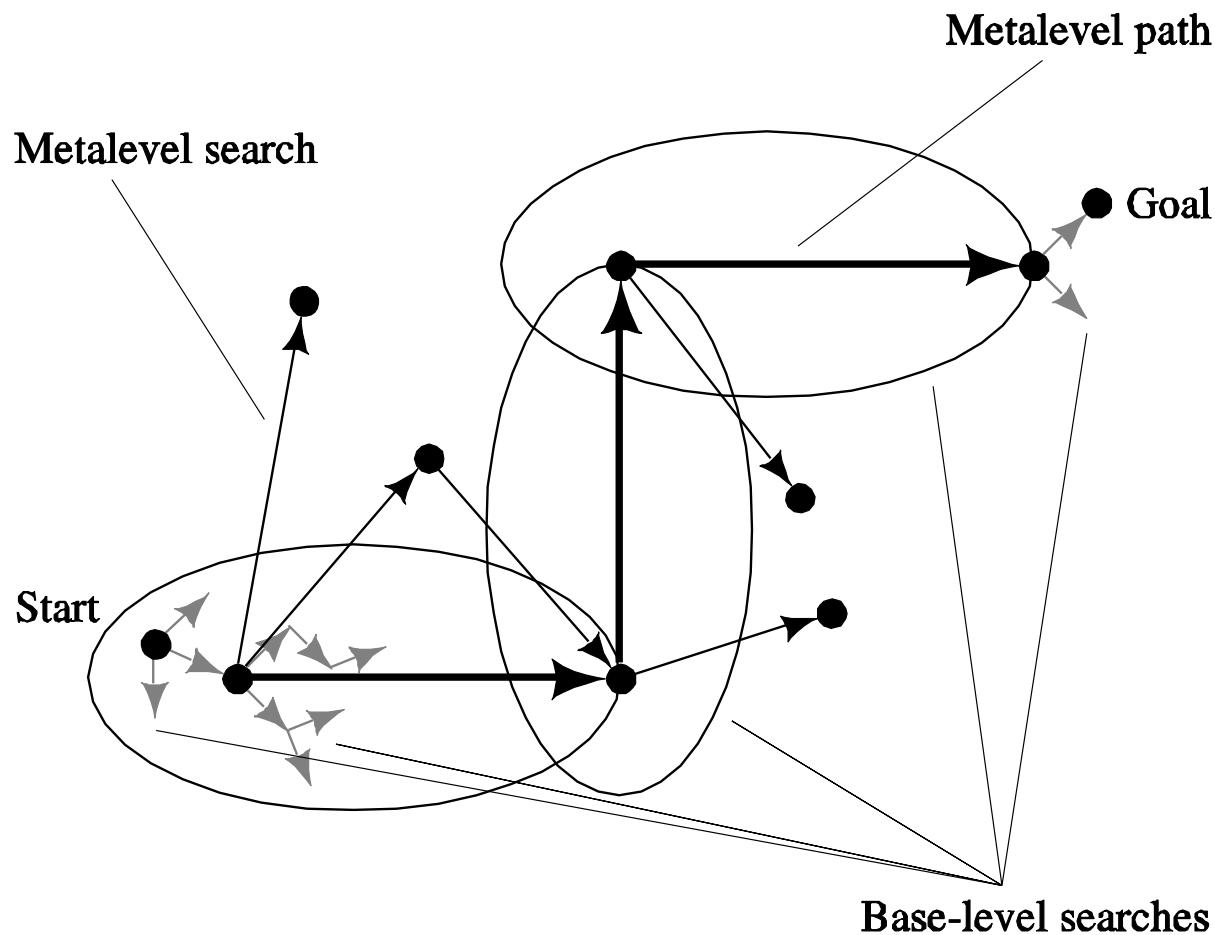


© 1998 Morgan Kaufman Publishers

Búsqueda con horizonte

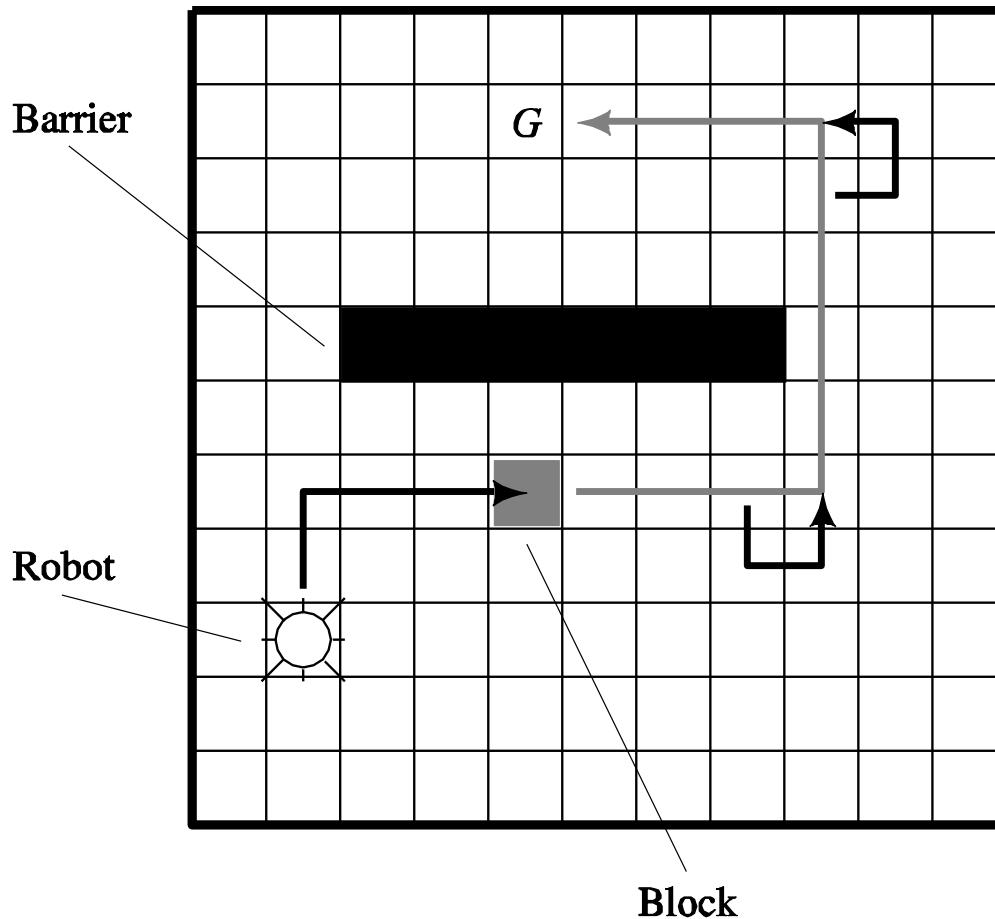
- Se establece una profundidad máxima (horizonte) y se realiza la búsqueda con esa profundidad máxima
- A veces es necesario cambiar el criterio de búsqueda del objetivo

Búsqueda jerárquica



© 1998 Morgan Kaufman Publishers

Búsqueda jerárquica



© 1998 Morgan Kaufman Publishers

Contenido

- Diseño de un agente deliberativo: búsqueda
- Sistemas de búsqueda y estrategias
- Búsqueda sin información
- Búsqueda con información
- **Problemas descomponibles y búsqueda**

Problemas descomponibles

- Base de datos inicial (C,B,Z)
- Operadores

R1: C → (D,L)

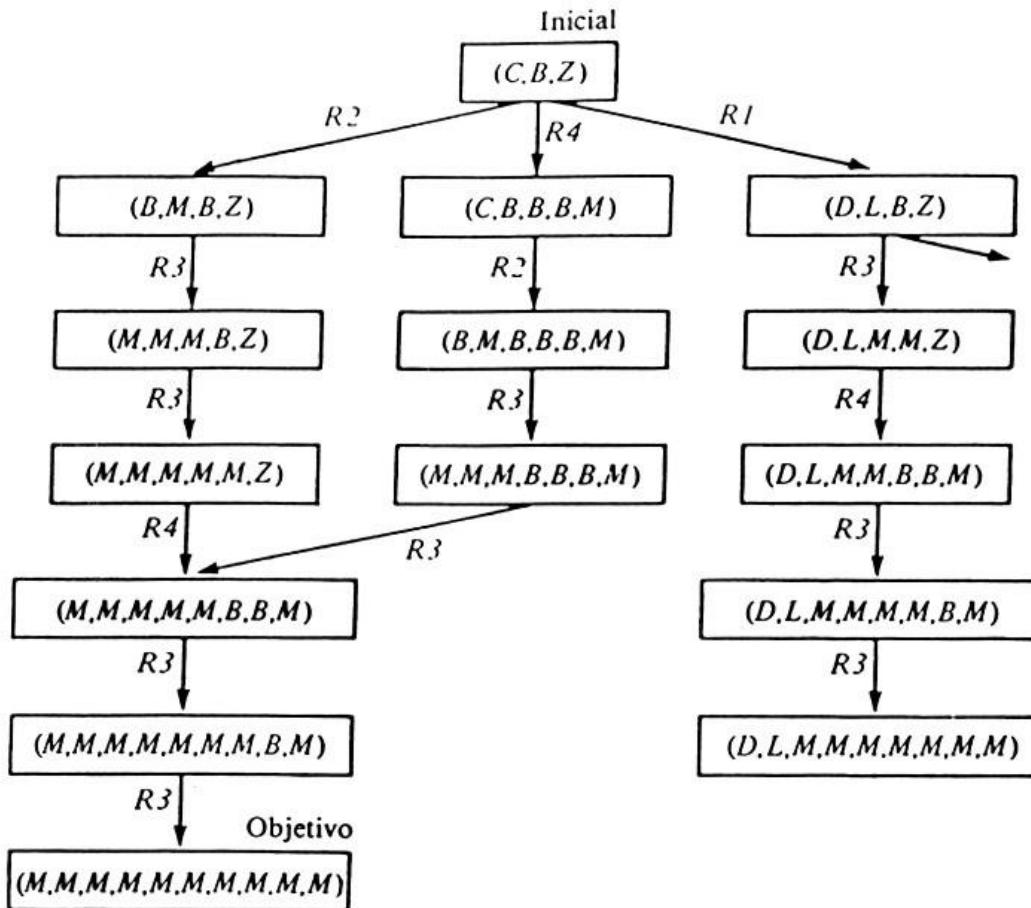
R2: C → (B,M)

R3: B → (M,M)

R4: Z → (B,B,M)

- Objetivo

Resolución del problema

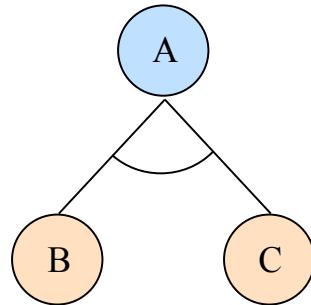


Grafo Y/O

- Descomposición de problemas: arcos Y
- Resolución de problemas: arcos O
- Concepto de solución: subgrafo solución

Grafo Y/O

- **Grafo Y:** Para completar el objetivo/tarea **A**, es necesario terminar antes los objetivos/tareas **B** y **C**.

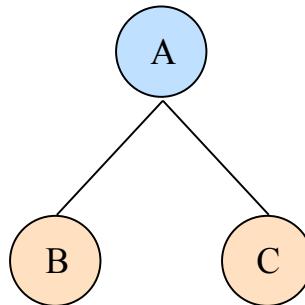


- En el cálculo proposicional, la expresión del grafo Y anterior correspondiente sería de la siguiente forma:

$$B \cdot C \rightarrow A$$

Grafo Y/O

- **Grafo O:** Para completar el objetivo/tarea **A**, es necesario terminar antes o bien el objetivo/tarea **B**, o bien el objetivo/tarea **C**.

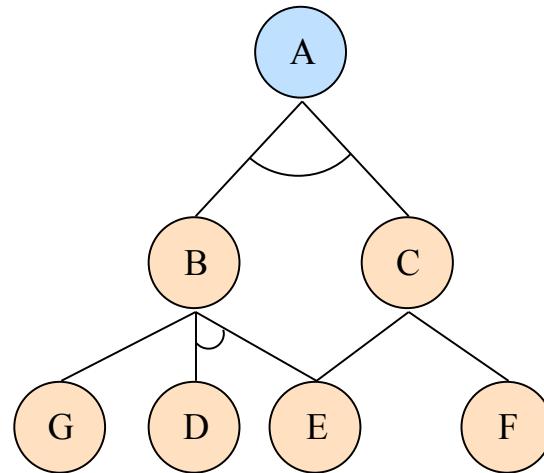


- En el cálculo proposicional, la expresión del grafo O anterior correspondiente sería de la siguiente forma:

$$B+C \rightarrow A$$

Grafo Y/O

- **Grafo Y/O:** Combinación de grafos Y y grafos O que indican el orden de consecución de tareas a realizar para alcanzar el objetivo.



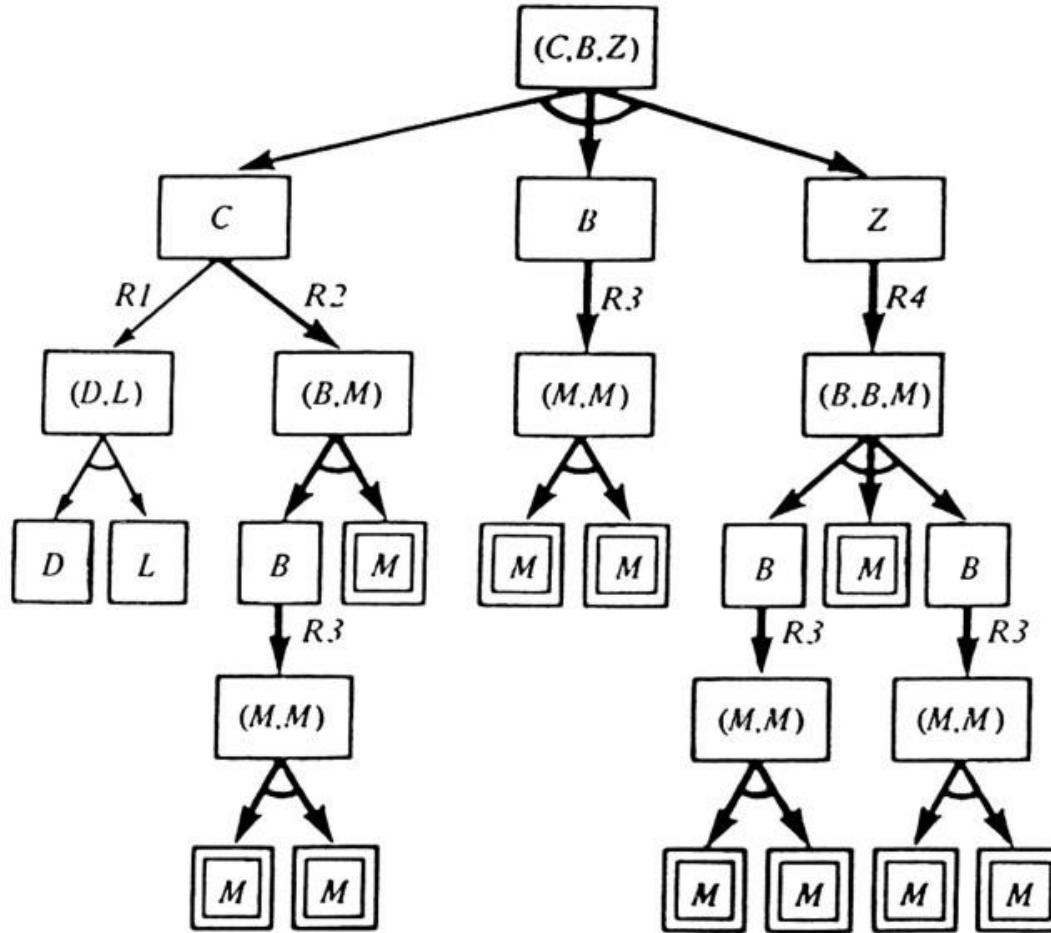
- En el cálculo proposicional, la expresión del grafo Y/O anterior correspondiente sería de la siguiente forma:

$$B \cdot C \rightarrow A; G + D \cdot E \rightarrow B; E + F \rightarrow C$$

Grafo Y/O

- **Para resolver un grafo Y/O**, cada nodo se resuelve de la siguiente manera:
 - Si es un nodo Y: Resolver todos sus hijos. Combinar la solución y solucionar el nodo. Devolver su solución.
 - Si es un nodo O: Resolver un hijo y ver si devuelve solución. En caso contrario, resolver el siguiente hijo, etc. Cuando ya esté resuelto algún hijo, combinar la solución en el nodo y devolverla.
 - Si es un nodo terminal: Resolver subproblema asociado y devolverla.
- **Mejora:** Para seleccionar el orden de resolución de nodos hijos, se puede utilizar alguna medida de estimación del coste de resolución.

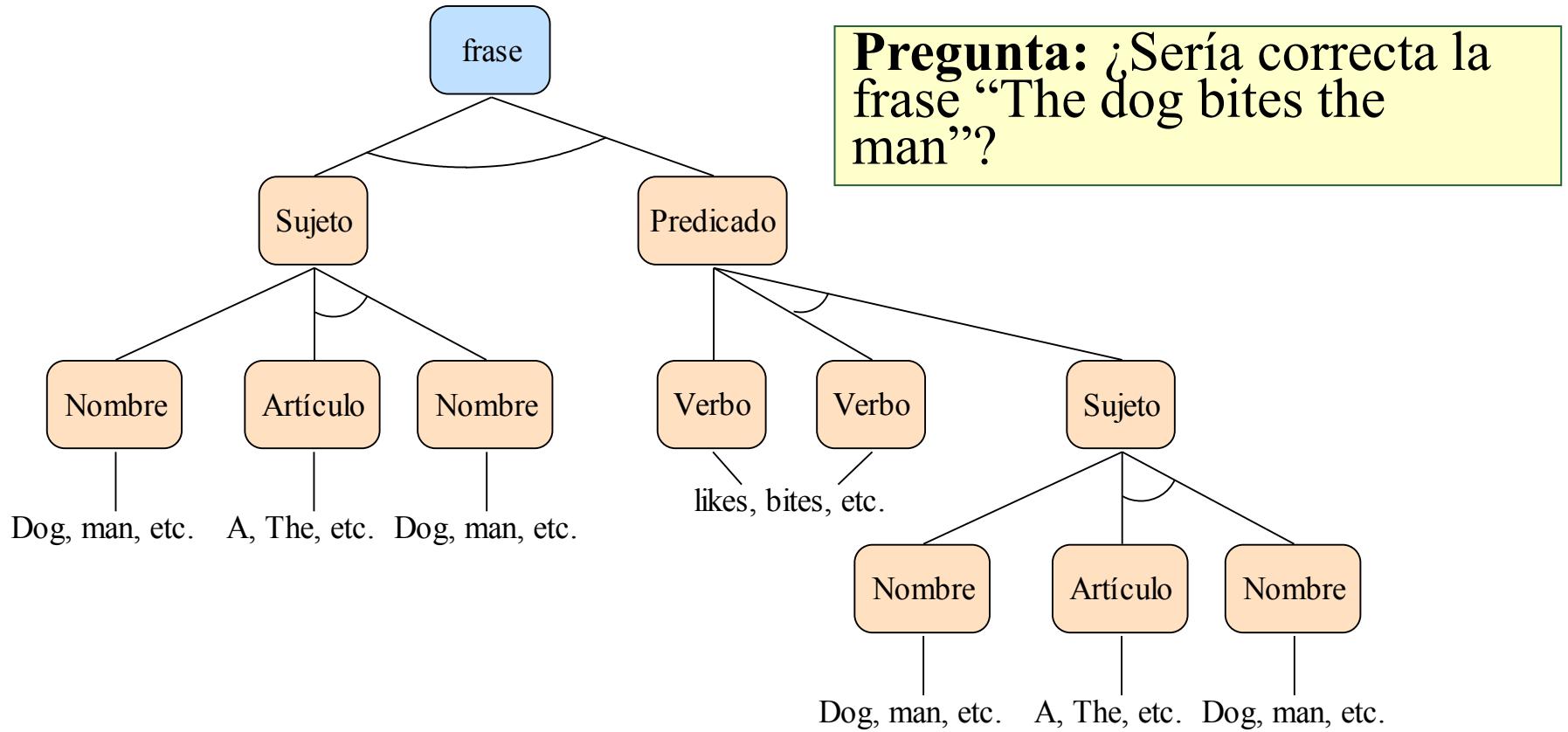
Nueva resolución del problema



Reconocimiento de frases de lengua inglesa

- Una frase está formada por un sujeto seguido de un predicado.
- El sujeto puede ser un sustantivo o un artículo seguido de un sustantivo.
- El predicado puede ser un verbo, o un verbo seguido de un complemento directo cuya estructura es idéntica a la del sujeto de la frase.

Reconocimiento de frases de lengua inglesa

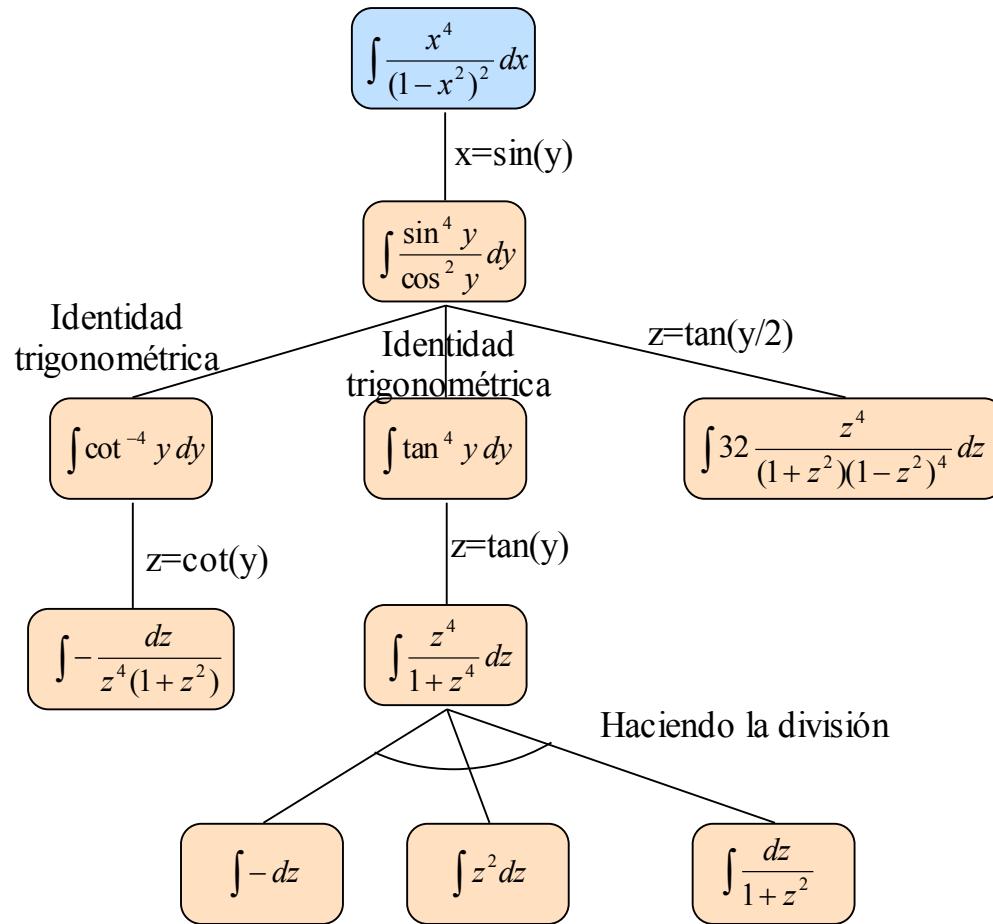


Resolución de integrales

- Para simplificar, supongamos que el computador conoce las transformaciones y técnicas de integración, incluidas en una Base de Datos o de Conocimiento.
- Esta técnica es la que implementa el programa MACSYMA, muy utilizado por matemáticos.
- Supongamos que queremos hacer la siguiente integración:

$$\int \frac{x^4}{(1-x^2)^2} dx$$

Resolución de integrales



Búsqueda con acciones no deterministas

