

# Búsqueda con información.

## Introducción a la Inteligencia Artificial

# Estrategias de Búsqueda:

## BÚSQUEDA SIN INFORMACIÓN



El agente sólo puede diferenciar un nodo que es meta de uno que no lo es. No posee información respecto a cuántos pasos necesita dar, o a qué distancia está de la meta.

## BÚSQUEDA RESPALDADA POR INFORMACIÓN



El agente posee información sobre el problema como para poder elegir operadores más convenientes.

# Marvin Minsky

- **Búsqueda sin Información**

En pequeños dominios, podemos intentar aplicar todos nuestros métodos de mindless search...pero no es práctico porque la búsqueda se vuelve enorme.

- **Búsqueda con información**

Para reducir la extensión de la búsqueda desinformada debemos incorporar conocimiento - incorporando experiencia en resolución de problemas durante la tarea de resolución de problemas.

# Búsqueda con Información

- Búsqueda Primero el Mejor
- Búsqueda A\*
  - Heurísticas
- Escalada (Ascenso a la Cima)
- Enfriamiento Simulado

# DISTINTAS ESTRATEGIAS DE BUSQUEDA

*EVALUACION DE:*

- ✓ Completitud
- ✓ Complejidad (temporal y espacial)
- ✓ Solución óptima

# MÉTODOS DE BÚSQUEDA CON INFORMACIÓN BÚSQUEDA HEURÍSTICA

*Al contar con información específica sobre un espacio de estados, se evitan emprender búsquedas a ciegas*

MÉTODOS GENERALES + HEURÍSTICAS DE  
PROPÓSITO ESPECIAL



**MAYOR EFICIENCIA**

# ALGORITMO DE BÚSQUEDA GENERAL.

## BÚSQUEDA GENERAL

responde con SOLUCIÓN o FALLA

LISTA-NODOS  $\leftarrow$  ESTADO INICIAL

bucle hacer

si LISTA-NODOS está vacía contestar FALLA

tomo NODO de LISTA-NODOS

si NODO es meta contestar con NODO

LISTA-NODOS  $\leftarrow$  expansión NODO

FIN

# MÉTODOS DE BÚSQUEDA CON INFORMACIÓN

Se utiliza una **FUNCIÓN HEURISTICA** para representar lo deseable que es la expansión de un nodo

**$f$  heurística**: rep. de estados  $\longrightarrow R$

El problema de búsqueda se puede considerar como la minimización de una función.

- Si  $f$  está bien diseñada guía la búsqueda eficientemente.
- $f$  ideal establece el camino a la meta.
- Problema: elegir buenas heurísticas!!!



# MÉTODOS DE BÚSQUEDA CON INFORMACIÓN

## BUSQUEDA PRIMERO EL MEJOR (Best first search)

*Los nodos se ordenan de tal manera que se **expande el nodo de mejor valor de la función heurística  $f$**  (mínimo, máximo), esta función puede incorporar conocimiento del dominio.*

### **Algoritmo:**

BUSQUEDA GENERAL donde LISTA-NODOS se ordena de acuerdo al valor de  $f(\text{nodo})$

# MÉTODOS DE BÚSQUEDA CON INFORMACIÓN

## BUSQUEDA PRIMERO EL MEJOR

*Distintas funciones evaluadoras  $f$*



*FAMILIA DE ALGORITMOS*

*BÚSQUEDA PRIMERO EL MEJOR*

- *Si  $f(n) = g(n)$  costo de ruta*

***BÚSQUEDA DE COSTO UNIFORME***

- *Si  $f(n) = h(n)$   $h$  heurística:*

***BÚSQUEDA DE COSTO MINIMO***

***(Avara - Greedy search)***

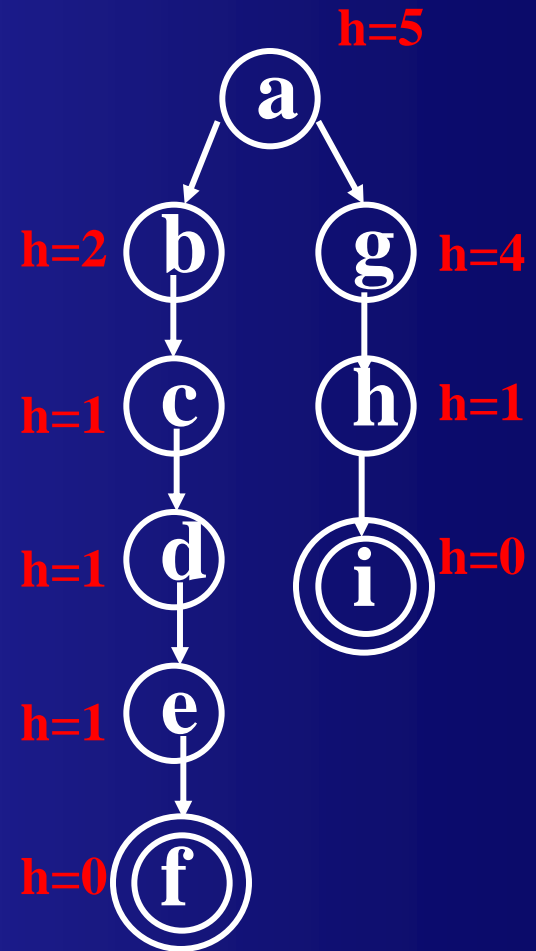
# BÚSQUEDA DE COSTO MINIMO (AVARA)

*Implementa Búsqueda primero el mejor, buscando **el mínimo de una función que representa el costo estimado para lograr una meta.***

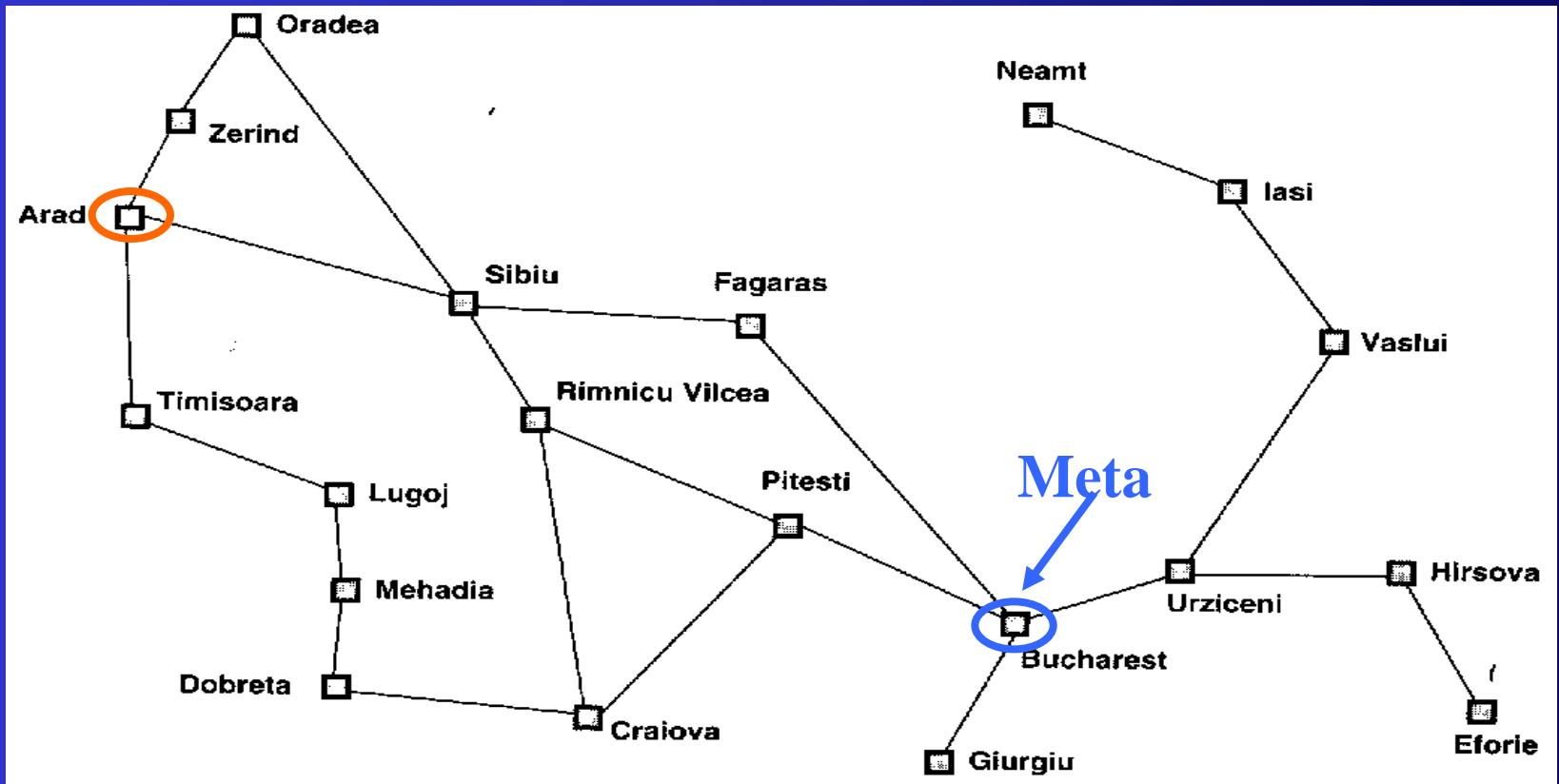
- *$h(n)$  = costo estimado de la ruta más barata que une al estado  $n$  con un estado meta.*
- *$h(meta) = 0$*
- *$h(n) = \infty$  significa que  $n$  es un nodo hoja desde el cual el goal no puede ser alcanzado.*

# Búsqueda Avara - Ejemplo

- Función de evaluación  $f(n) = h(n)$   
(la lista de los nodos se ordena de tal forma que **el nodo de mejor evaluación sea el primero**).
- Selecciona el nodo a expandir con menor valor de  $f = h$
- No es óptima. Greedy search encontrará el goal  $f$  (costo de 5), mientras que la solución óptima tiene un costo de 3.
- No es completa.



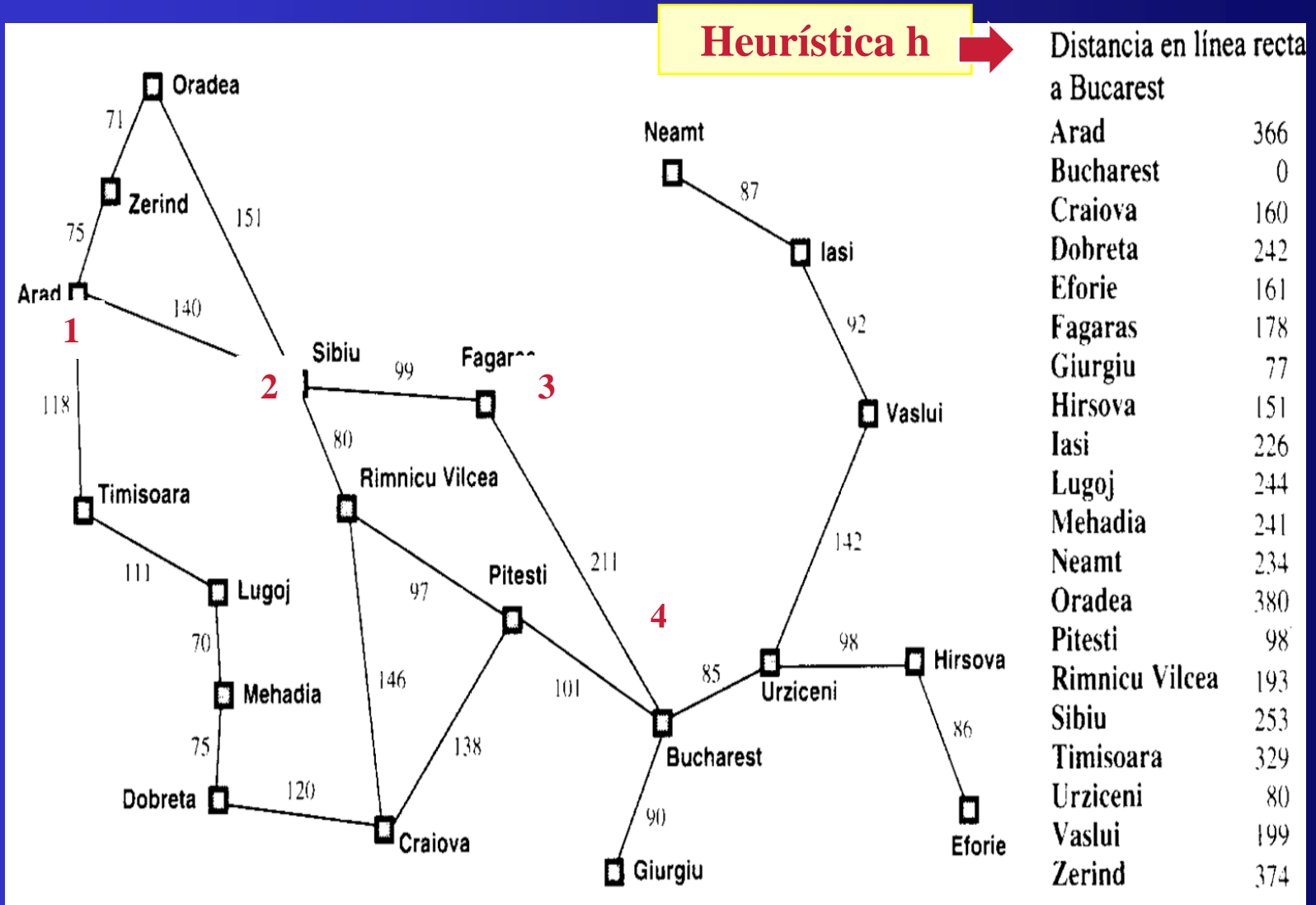
# Ejemplo: Encontrar Camino de Arad a Bucarest



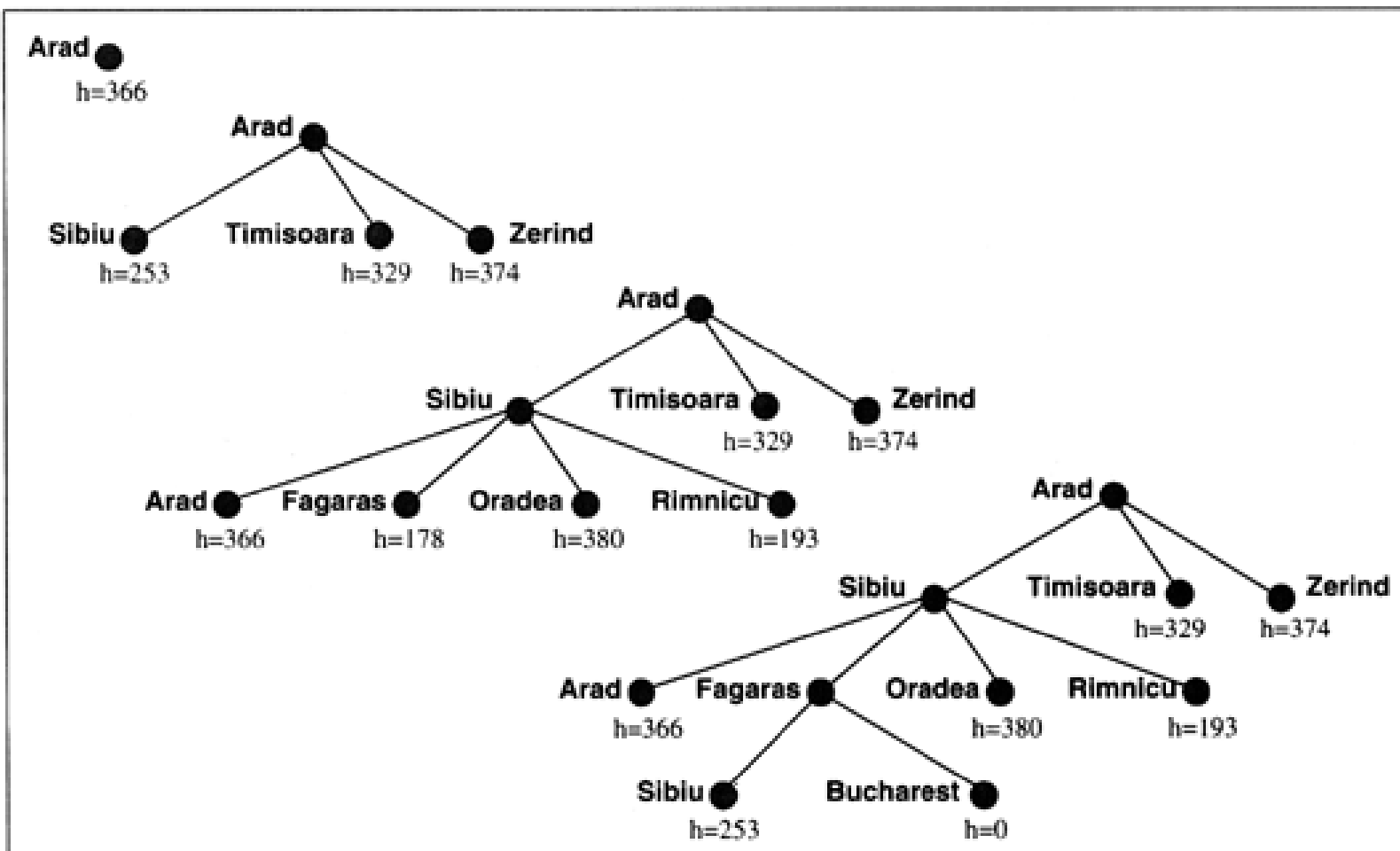
En problemas de búsqueda de ruta una buena  $h$  es  $h_{DLR}$  que evalúa la distancia en línea recta

# BUSQUEDA AVARA

Ejemplo: Encontrar Camino de Arad a Bucarest



# ÁRBOL DE BÚSQUEDA PARCIAL (Arad a Bucarest).



**Figura 4.3** Etapas de una búsqueda avara para llegar a Bucarest. Se utiliza la distancia en línea recta a Bucarest y la función heurística  $h_{DLR}$ . Los nodos se identifican con sus valores  $h$  correspondientes.

# Búsqueda Avara

- En este ejemplo, la búsqueda avara produce un costo de búsqueda mínimo -no expande nodos fuera de la ruta solución-
- No es la ruta óptima (ruta por Rimnicu-Pitesti es más corta)
- Desempeño bastante bueno, tienden a encontrar soluciones rápidamente
- Susceptible a **pasos en falso** (ej. Iasi  $\Rightarrow$  Fagaras) la heurística sugiere ir hacia Neamt, **ruta muerta sin salida**



# BUSQUEDA AVARA

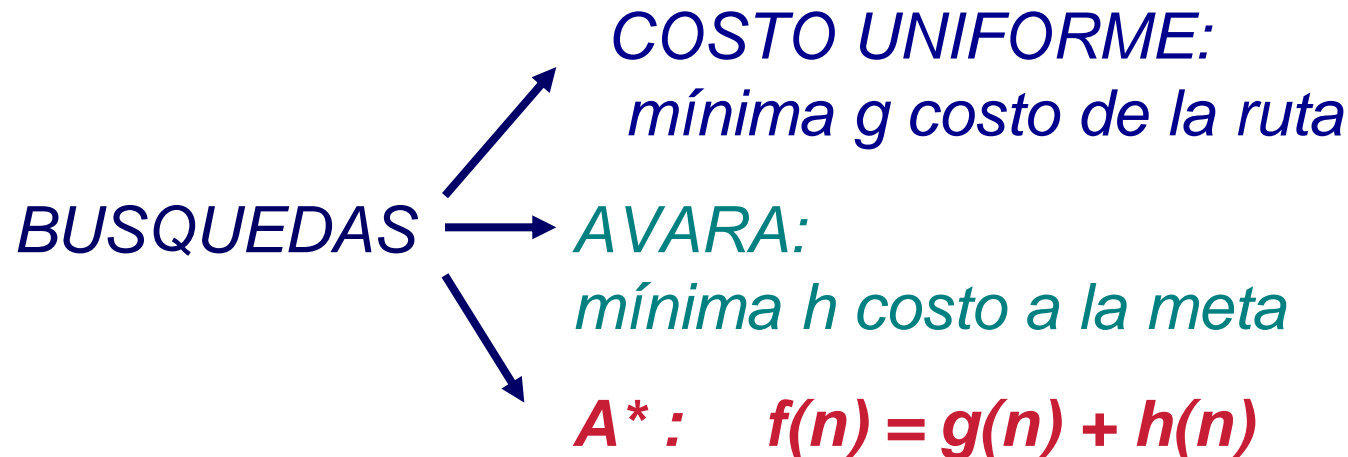
*Similar a BPP*

- **No es completa**
  - Puede colgarse en algún bucle

Pasa a ser completa en espacio finito si se hace una verificación de estado repetido
- **No es óptima**
  - lo vimos en el ejemplo
- **Complejidad espacial y temporal es  $O(b^m)$**  -
  - Mantiene todos los nodos en memoria
  - *Si  $h$  es buena la complejidad disminuye*

# BUSQUEDA A\*

*Implementa Búsqueda primero el mejor buscando el mínimo costo total, combinando el costo de ruta hasta  $n$  y el costo de  $n$  hasta una meta.*



➤  $f(n)$  = costo estimado de la solución más barata que pasa por  $n$

# Búsqueda A\*

- Función de evaluación

$$f(n) = g(n) + h(n)$$

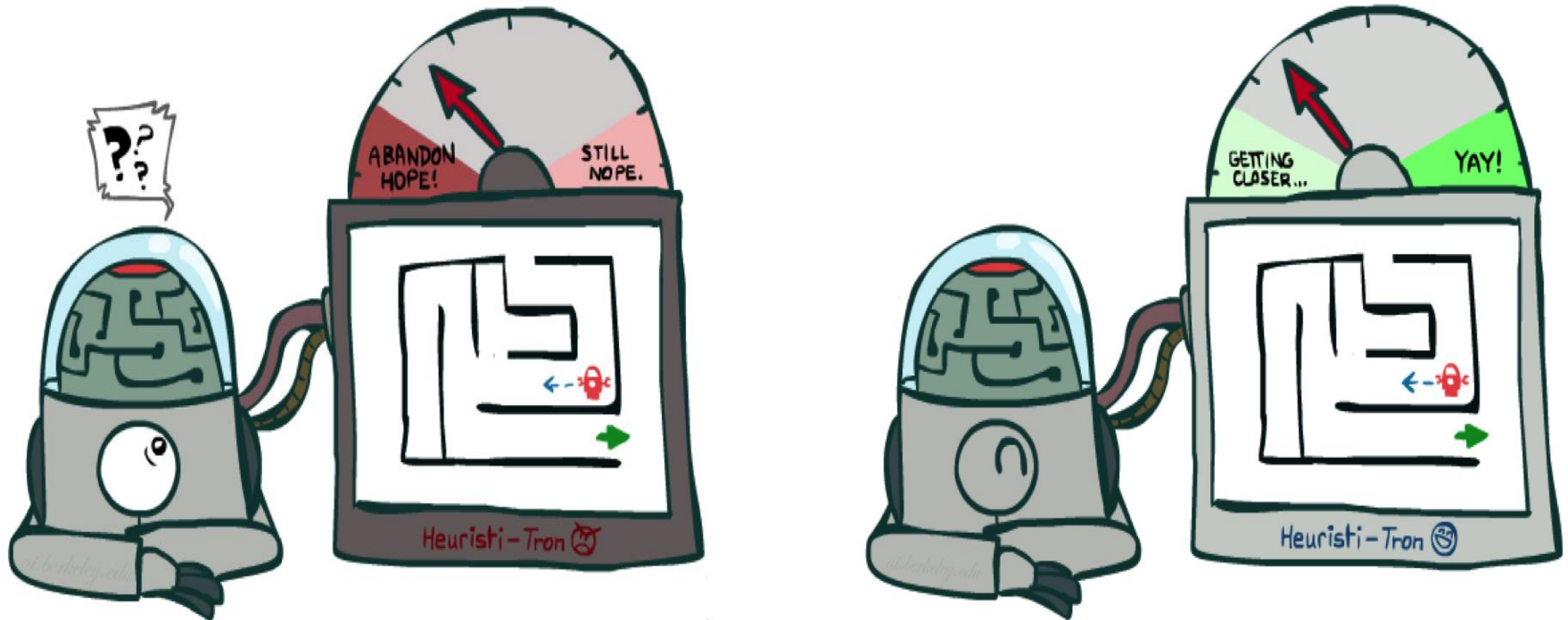


- $g(n)$  = costo hasta llegar a  $n$
- $h(n)$  = costo estimado hasta la meta desde  $n$
- $f(n)$  = costo total de ruta pasando por  $n$  hasta la meta

# Búsqueda A\*

- Una **heurística admisible** nunca sobreestima el costo de llegar a la meta.
  - Una estimación optimista del costo de la solución de un problema, es menor que el real.
- Si  $h$  es admisible,  $f(n)$  **nunca sobreestima** el costo real de la mejor solución pasando por  $n$
- La búsqueda A\* - con  $h$  admisible
  - **completa y óptima**

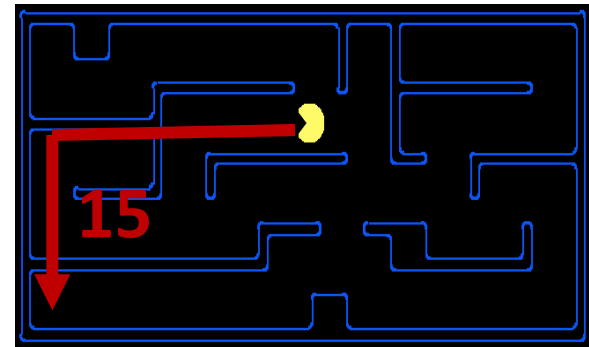
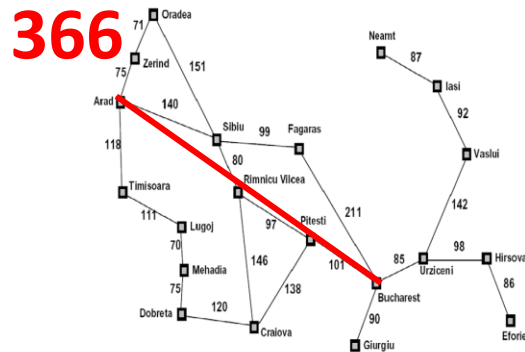
# Idea: Admisibilidad



Admisible (optimista) heurística que desalienta a los planes malos pero que nunca sobrestima los costos reales

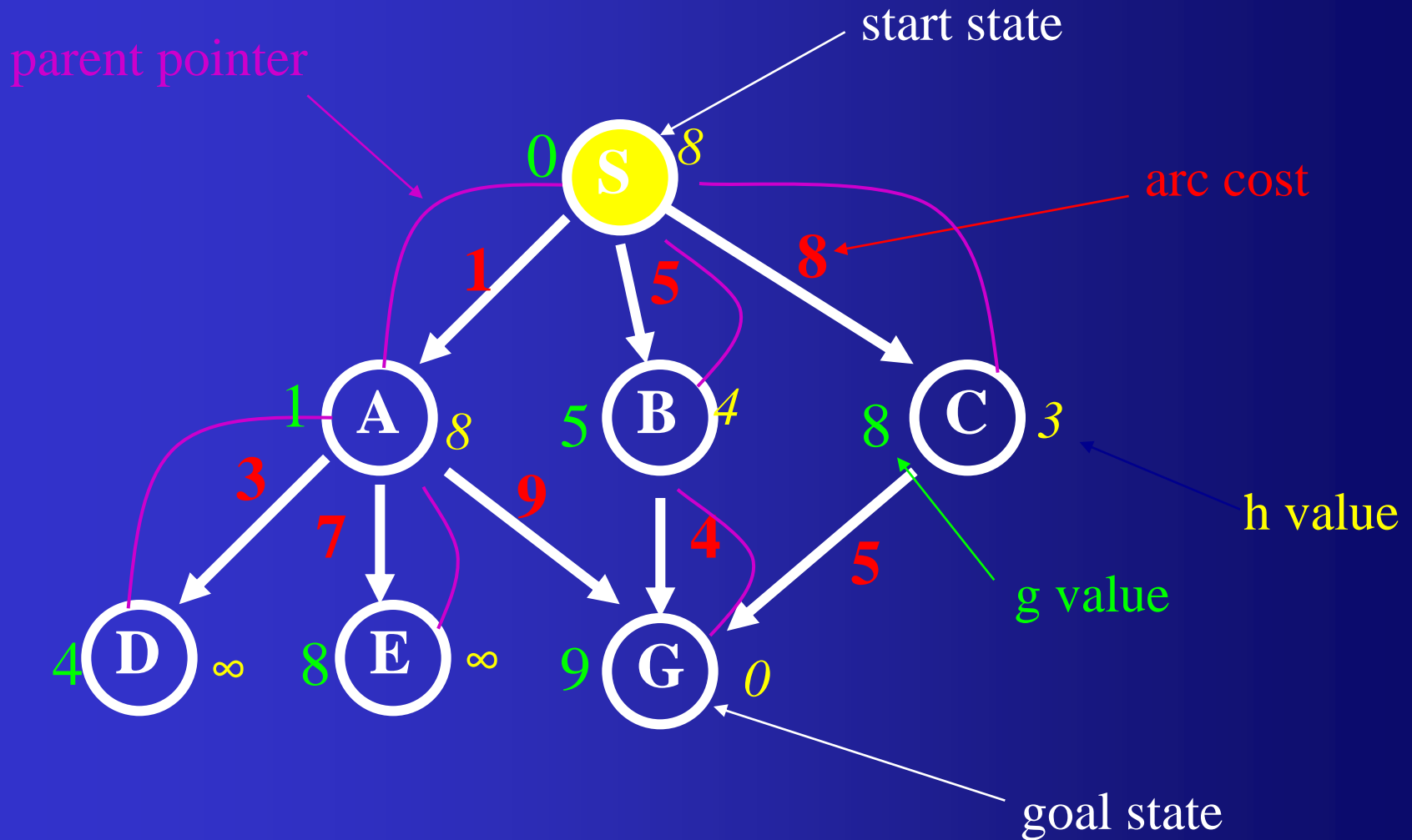
# Creando Heurísticas Admisibles

- El mayor trabajo para resolver problemas de búsqueda duros óptimamente es encontrar heurísticas admisibles
- Generalmente, estas heurísticas son soluciones de *problemas relajados*, donde se permiten nuevas acciones



- Heuristics inadmisibles también pueden ser útiles

# Ejemplo: espacio de búsqueda



# Ejemplo

n	g(n)	h(n)	f(n)	h*(n)
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	inf	inf	inf
E	8	inf	inf	inf
G	9	0	9	0

- $h^*(n)$  es la heurística perfecta (hipotéticamente).
- Como  $h(n) \leq h^*(n) \forall n$ ,  $h$  is admisible
- Camino óptimo S B G con costo 9.



# Algoritmo Greedy

$$f(n) = h(n)$$

node exp.	OPEN list
	{S( 8)}
S	{C(3) B(4) A(8)}
C	{G(0) B(4) A(8)}
G	{B(4) A(8)}

- El camino de solución es S C G con costo 13.
- 3 nodos expandidos.
- Rápida, pero no óptima.

**A\***

$$f(n) = g(n) + h(n)$$

**node exp.      OPEN list**

**{S(8)}**

**S            {A(9) B(9) C(11)}**

**A            {B(9) G(10) C(11) D(inf) E(inf)}**

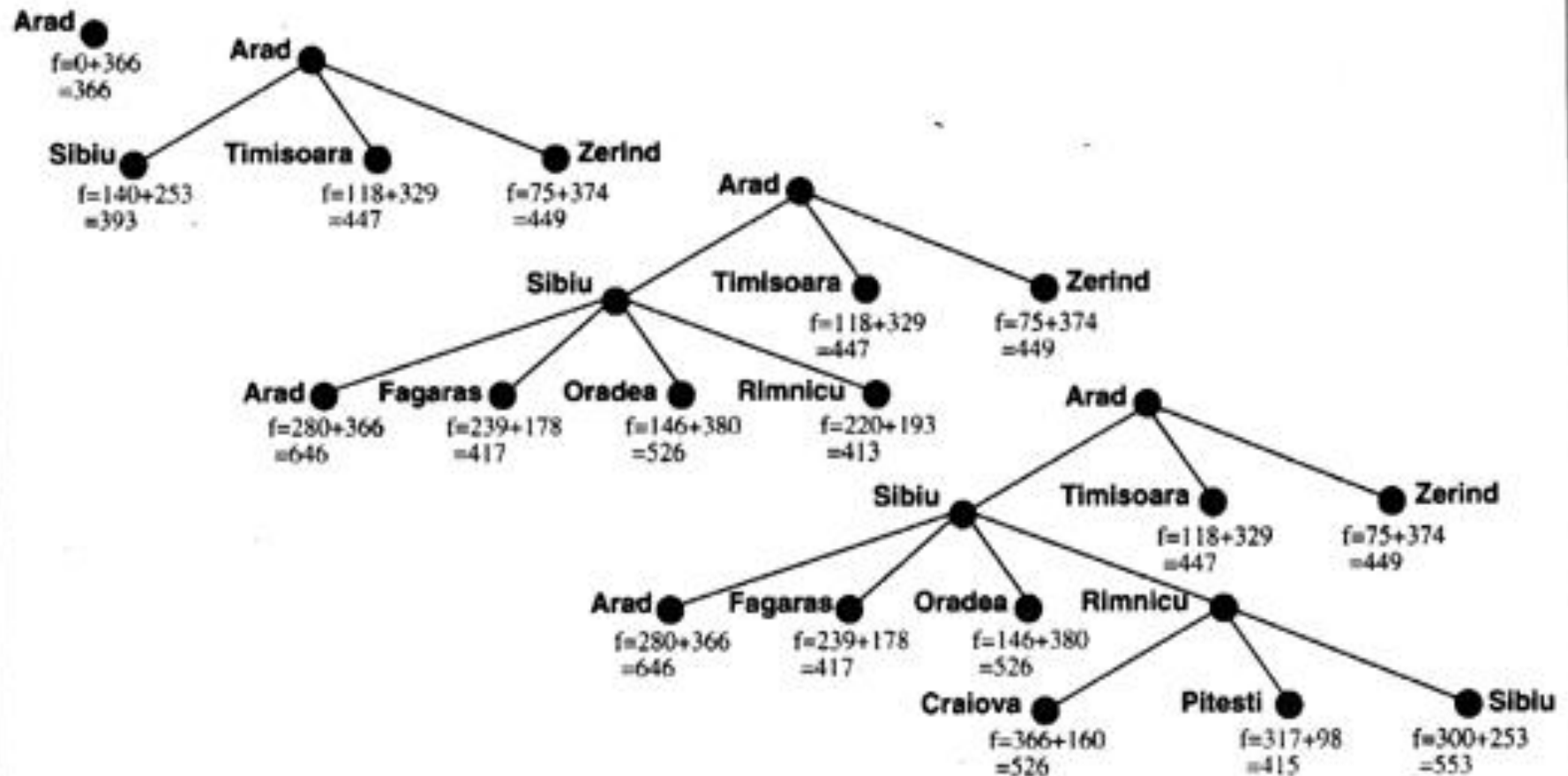
**B            {G(9) G(10) C(11) D(inf) E(inf)}**

**G            {C(11) D(inf) E(inf)}**

- El camino de solución es S B G con costo 9
- 4 nodos expandidos.
- Bastante rápido y óptimo!

# BUSQUEDA A\*

Ejemplo: Encontrar Camino de Arad a Bucarest

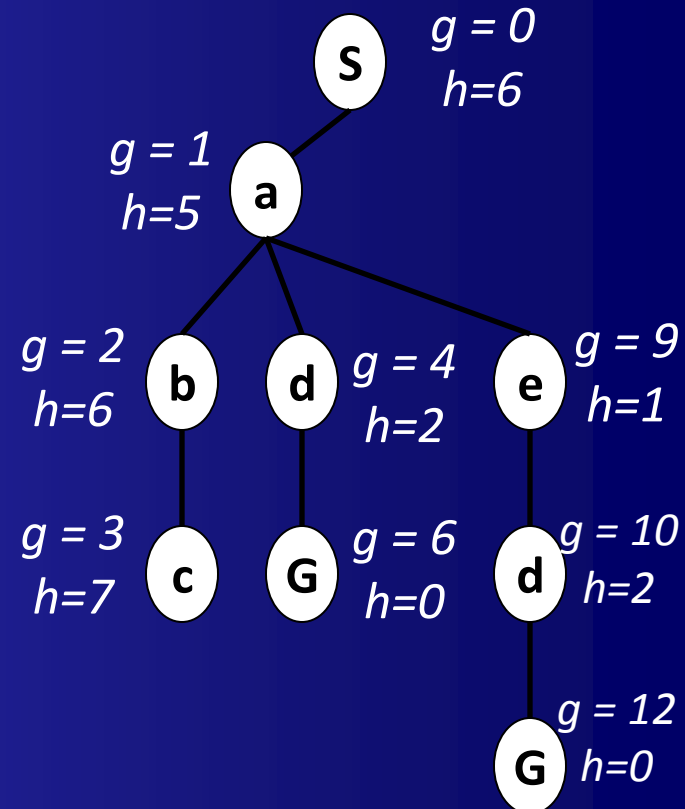
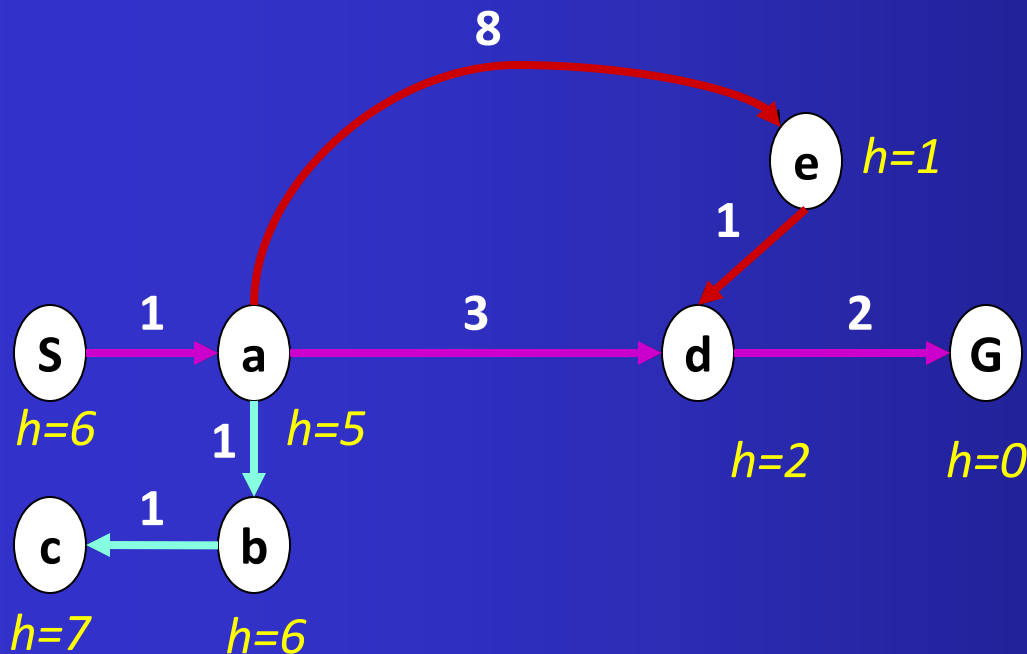


**Figura 4.4** Etapas de una búsqueda A\* para llegar a Bucarest. Los nodos se identifican con  $f = g + h$ . Los valores de  $h$  corresponden a las distancias en línea recta a Bucarest de la figura 4.1.

# Ejemplo: Costo unif., Greedy y A\*

Uniform-cost orders by path cost  $g(n)$

Greedy orders by goal proximity, or *forward cost*  $h(n)$



A\* Search orders by the sum:  $f(n) = g(n) + h(n)$

Example: Teg Grenager

# BUSQUEDA A\*

Condiciones para que A\* sea completa y óptima:

- ***h sea admisible:***

*h no sobreestime el costo a la meta*

*f no sobrestima el costo real de la solución*

*Ejemplo:  $h$  = distancia en línea recta*

- ***f es monótona (consistente):***

*si nunca disminuye a través de una ruta que parte de la raíz:  $f(\text{padre}(n)) \leq f(n)$*

# BUSQUEDA A\*

*Condiciones para que A\* sea completa y óptima:*

## □ ***En grafos consistencia:***

*Si para cada nodo  $n$  y sucesor  $n'$*

$$h(n) \leq c(n, n') + h(n')$$

- *Toda heurística consistente es admisible.*
- *No se puede garantizar que toda heurística admisible sea consistente (aunque casi siempre lo es)*

# Conducta de la búsqueda A\*

**Si el costo de f nunca decrece: ES MONOTONA**

(esto suele darse en las heurísticas admisibles)

**Si f no es monótona (caso raro)**

$f(n) = g(n) + h(n) = 3+4 = 7$  siendo n nodo padre

$f(n') = g(n') + h(n') = 4+2 = 6$  siendo n' nodo hijo

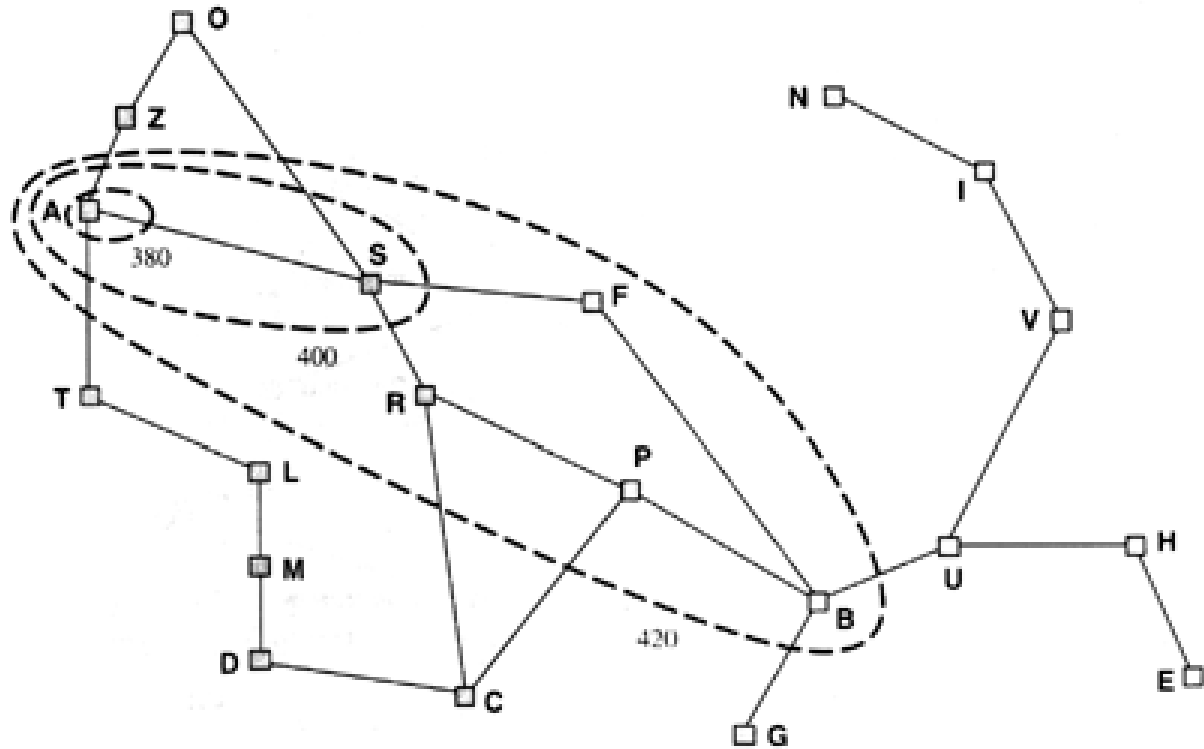
**Realizar entonces una corrección que restituya la monotonicidad de f**

$$f(n') = f(n)$$

## BUSQUEDA A\*

*f monótona* 

## CONTORNOS en el espacio de estados / $f(n) \leq C$



**Figura 4.5** Mapa de Rumania en donde se muestran los contornos correspondientes a  $f = 380$ ,  $f = 400$  y  $f = 420$ . Arad es el estado de partida. El costo de los nodos comprendidos dentro de un contorno es  $f$  veces menor que el de ese contorno.

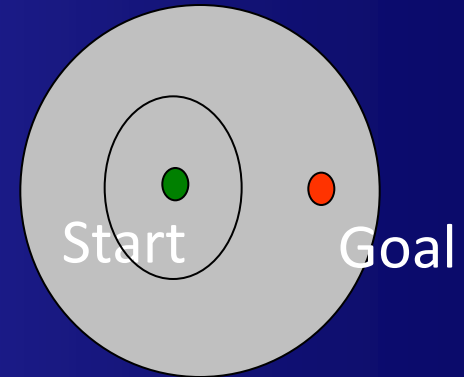


# Conducta de la búsqueda A\*

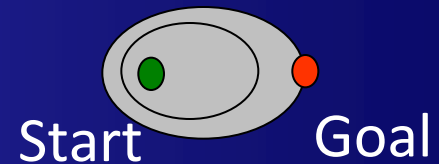
- Con una búsqueda de costo uniforme (A\* usando  $h = 0$ ), las zonas cubiertas entre dos contornos son anillos circulares alrededor del estado de inicio.
- Con una heurística ( $h > 0$ ) incorporada, los contornos se estirarán hacia el estado meta y poco a poco irán delimitando más la ruta óptima, enmarcándola más ajustadamente.

# UCS vs A\* Contornos

Costo Unif. Expande  
igualmente en todas las  
“direcciones”



A\* expande principalmente  
en dirección a la meta,  
pero se asegura no perder  
la optimalidad



# Optimalidad de $A^*$

- Definir  $f^*$  - el costo de la solución óptima para la ruta
  - $A^*$  expande todos los nodos con  $f(n) < f^*$
  - $A^*$  podría expandir algunos de los nodos para los cuales  $f(n) = f^*$ , antes de seleccionar el estado meta.
- La primera solución encontrada debe ser la óptima, dado que los nodos de todos los contornos subsiguientes tendrán un costo  $f$  más alto y con ello un costo  $g$  más alto (todos los estados meta tienen  $h(n) = 0$ ).

# Prueba de la optimalidad de A\*

- $*$
- -----
- -----
- $* n$
- $* G1$
- $* G2$

Sea una meta subóptima  
G2 que está en la cola  
de espera

Sea n un nodo sin  
expandir en el camino  
más corto hacia una  
meta óptima G1

A\* nunca va a elegir G2  
para su expansión

# Optimalidad de $A^*$

**Teorema:** Sea  $h^*(n)$  el costo real desde  $n$  hasta la meta. Si  $h$  es admisible, entonces  $A^*$  siempre va a encontrar un nodo meta óptimo.

**Prueba:** Sea  $G1$  el nodo meta de mínimo costo. Se supone que  $A^*$  seleccione un nodo meta subóptimo  $G2$ , donde  $g(G1) < g(G2)$

Sea  $n$  un nodo sin expandir en la ruta desde el nodo inicio y el nodo meta óptimo  $G1$ . Notar que ese nodo sin expandir necesariamente existe, de acuerdo con la suposición previa (en el otro caso,  $G1$  ya habría sido elegido como el nodo meta).

# Optimalidad de $A^*$

Puesto que  $n$  no ha sido elegido para su expansión en su ruta hacia  $G2$ , se sigue que:

$$f(n) = g(n) + h(n) \geq f(G2) = g(G2)$$

Dado que  $f$  es monótona,

$$f^* \geq g(n) + h(n) = f(n), \text{ y entonces}$$

$$f^* \geq f(n) \geq f(G2) = g(G2)$$

lo cual implica que

$$g(G1) \geq g(G2)$$

$\Rightarrow$  Esto contradice la suposición previa, que  $G2$  es una meta subóptima.

# Optimalidad de A\* Tree Search



# Optimality of A\* Tree Search

Assume:

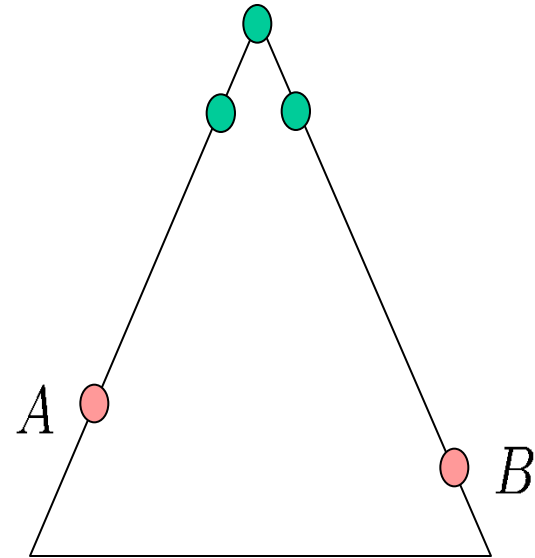
A is an optimal goal node

B is a suboptimal goal node

$h$  is admissible

Claim:

A will exit the fringe before B





# Optimalidad de A\* Tree Search:

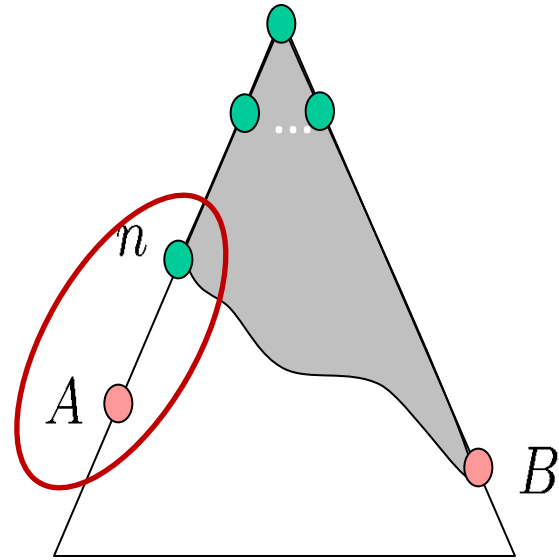
Proof:

Imagine B is on the fringe

Some ancestor  $n$  of A is on the fringe, too (maybe A!)

Claim:  $n$  will be expanded before B

1.  $f(n)$  is less or equal to  $f(A)$



$$f(n) = g(n) + h(n)$$

$$f(n) \leq g(A)$$

$$g(A) = f(A)$$

# Optimalidad de A\* Tree Search:

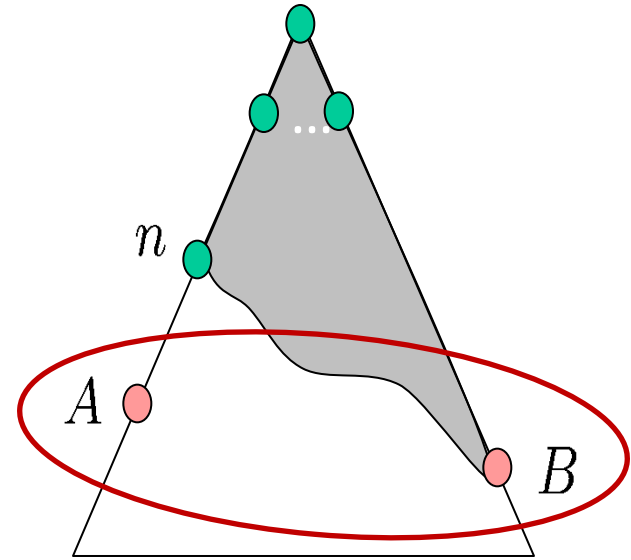
Proof:

Imagine B is on the fringe

Some ancestor  $n$  of A is on the fringe, too (maybe A!)

Claim:  $n$  will be expanded before B

1.  $f(n)$  is less or equal to  $f(A)$
2.  $f(A)$  is less than  $f(B)$



$$g(A) < g(B)$$

$$f(A) < f(B)$$

# Optimalidad of A\* Tree Search:

Proof:

Imagine B is on the fringe

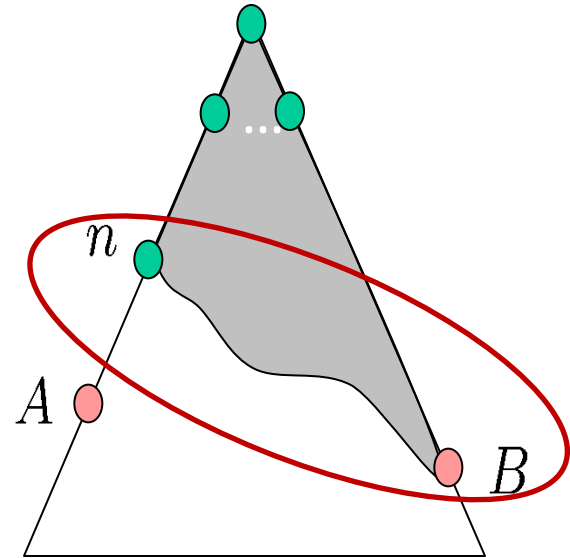
Some ancestor  $n$  of A is on the fringe, too (maybe A!)

Claim:  $n$  will be expanded before B

1.  $f(n)$  is less or equal to  $f(A)$
2.  $f(A)$  is less than  $f(B)$
3.  $n$  expands before B

All ancestors of A expand before B - A expands before B

A\* search is optimal



$$f(n) \leq f(A) < f(B)$$

# Casos límites de $A^*$ : $f = g+h$

- Si  $h=0$  y  $g=0 \Rightarrow$
- Si  $h=0$  y  $g=d \Rightarrow$
- Si  $h=1/d$  y  $g=0 \Rightarrow$
- Si  $h=0 \Rightarrow$
- Si  $g=0 \Rightarrow$
- Si  $h(n) > h^*(n) \Rightarrow$
- Si  $h(n) \ll h^*(n) \Rightarrow$

???

# Casos límites de $A^*$ : $f = g+h$

- Si  $h=0$  y  $g=0 \Rightarrow$  Búsqueda aleatoria
- Si  $h=0$  y  $g=d \Rightarrow$  BPA
- Si  $h=1/d$  y  $g=0 \Rightarrow$  BPP
- Si  $h=0 \Rightarrow$  Búsqueda de costo uniforme
- Si  $g=0 \Rightarrow$  Búsqueda avara
- Si  $h(n) > h^*(n) \Rightarrow$  se puede perder la ruta óptima (no admisible)
- Si  $h(n) \ll h^*(n) \Rightarrow$  es óptima pero puedo expandir nodos de más

# BÚSQUEDA A\*

- *Cuanto más precisas sean las heurísticas, los contornos se concentran más en torno de la ruta óptima*

## **Algoritmo A\***



### **Completo**

**A\* expande nodos en orden creciente de  $f$ , con lo cual expandirá hasta llegar al estado meta**

- **salvo que haya una cantidad infinita de nodos con  $f(n) < f^*$** 
  - una ruta con costo de ruta finito pero con un número infinito de nodos a lo largo de ella
  - un nodo con un factor de ramificación infinito

# BÚSQUEDA A\*

## **Algoritmo A\***

- **Óptimo**
- **Optimamente eficiente**

***Ningún otro algoritmo óptimo expandirá menos nodos que A\****

- Cualquier algoritmo, que no expanda todos los nodos en los contornos existentes entre el contorno del inicio y el de la meta, corre el riesgo de no encontrar la solución óptima

- **Complejidad es exponencial  $O(b^d)$**

*Es subexponencial si el error de  $h$  es muy pequeño:  $|h(n) - h^*(n)| \leq O(\log h^*(n))$ ,  $h^*$  es el costo real para ir de  $n$  a la meta*

# METODOS DE BUSQUEDA CON INFORMACION

## BUSQUEDA A\*



- el uso de una heurística buena provee ventajas enormes
- usualmente A\* se queda sin espacio antes de quedarse sin tiempo, puesto que mantiene a todos los nodos en memoria
- ❖ *Problema de memoria > problema del tiempo*
- ❖ *Problema de encontrar buenas heurísticas h !!!*



# METODOS DE BUSQUEDA CON INFORMACION BUSQUEDA A\*



Cómo manejar mejor la memoria?

## ***VARIANTES DEL A\*:*** ***(Diseñados para conservar memoria)***

- *A\*PI : A\* por búsqueda iterativa*
- *A\*SRM : A\* acotada por memoria simplificada*

# A\* con Profundización Iterativa - IDA\*

BPP en el subárbol cuyos nodos tienen un valor de  $f$  menores o iguales al  $f$  límite

- ¿Gran problema de A\*?  $\Rightarrow$  mucho requisito de memoria.
- ¿Cuál es lo mejor para economía de memoria? BPP (DFS)
- ¿De qué forma se mejoraban los defectos de BPP sin empeorar más que un poco sus requisitos de memoria?  $\Rightarrow$  búsqueda por profundización iterativa
- De allí: iterative deepening A\* search (IDA\* o A\*PI)
  - cada iteración es una búsqueda en profundidad, ahorrativa, usando un límite basado en el costo  $f$  y no en el límite de profundidad

# Búsqueda limitada por Memoria

## A\* Simplificada y Limitada por Memoria (SMA\*)

- Según exigencias de memoria, descarta nodos de ella que tengan valores de  $f$  altos.
- Los valores de  $f$  descartados quedan memorizados en ancestros
- Mecanismo de regeneración de nodos descartados solo si todo el resto de rutas son peores.
- Optima y completa si la solución más cercana entró en la memoria.
  - en el otro caso, entrega la mejor solución alcanzable

# En qué consiste SMA\*

- IDA\* emplea demasiado poca memoria y no ocupa todo su potencial, con lo cual se malgasta esfuerzo.
- SMA\* usa en cambio toda la memoria  $M$  disponible para realizar la búsqueda.
  - evita estados repetidos dentro de la disponibilidad de  $M$
  - completa si  $M \geq d$  (*nodo meta más superficial*), óptima si  $M \geq d^*$  (*solución óptima es alcanzable*)

**HAY MUCHOS ALGORITMOS DE LA FLIA A\***

# Funciones Heurísticas ???

5	4	
6	1	8
7	3	2

*estado inicial*

1	2	3
8		4
7	6	5

*Estado meta*

- Problema de los 8 números-Restricciones: no avanzar dos o más pasos por turno, no avanzar diagonalmente, no superponer números, etc

# Funciones Heurísticas ???

5	4	
6	1	8
7	3	2

*estado inicial*

1	2	3
8		4
7	6	5

*Estado meta*

- Problema de los 8 números-Restricciones: no avanzar dos o más pasos por turno, no avanzar diagonalmente, no superponer números, etc

$h1(n)$  = números fuera de orden

$h2(n)$  = suma de distancias de Manhattan

## Comparación de Costos de búsqueda y factor de ramificación

$d$	Costo de búsqueda			Factor de ramificación efectivo		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

**Figura 4.8** Comparación de los costos de búsqueda y de los factores de ramificación efectivos de la BÚSQUEDA-POR PROFUNDIZACIÓN-ITERATIVA y los algoritmos  $A^*$ , correspondientes a  $h_1$  y  $h_2$ . Los datos se promediaron entre 100 resultados posibles del problema de las ocho placas, correspondientes a longitudes de solución diversas.

# Funciones heurísticas


Encontrar una buena heurística para un problema

- **Utilizar Problema relajado** - menos restricciones impuestas a los operadores.- El costo de una solución a un problema relajado es a menudo una buena heurística para el problema original.
- Siempre será mejor usar una **función heurística mayor**, sin sobreestimar - *heurística compuesta* -  
$$h(n) = \max(h_1(n), \dots, h_m(n))$$
- La evaluación heurística debiera ser **eficiente**.
- Costo de búsqueda  $\Rightarrow$  hay que considerar también el **costo de usar  $h$**  en un nodo



# ALGORITMOS DE MEJORAMIENTO ITERATIVO

No interesa la ruta a la solución



*La idea básica consiste en comenzar con una configuración completa y luego modificarla. El objetivo de una mejora iterativa es explorar en búsqueda de las cimas más altas (soluciones óptimas).*

*Sirven como solución a muchos problemas de diseño-configuración.*

## EJEMPLOS

- **ESCALADA (HILL CLIMBING)**
- **ENDURECIMIENTO SIMULADO**

# ALGORITMOS DE BÚSQUEDA LOCAL MEJORAMIENTO ITERATIVO

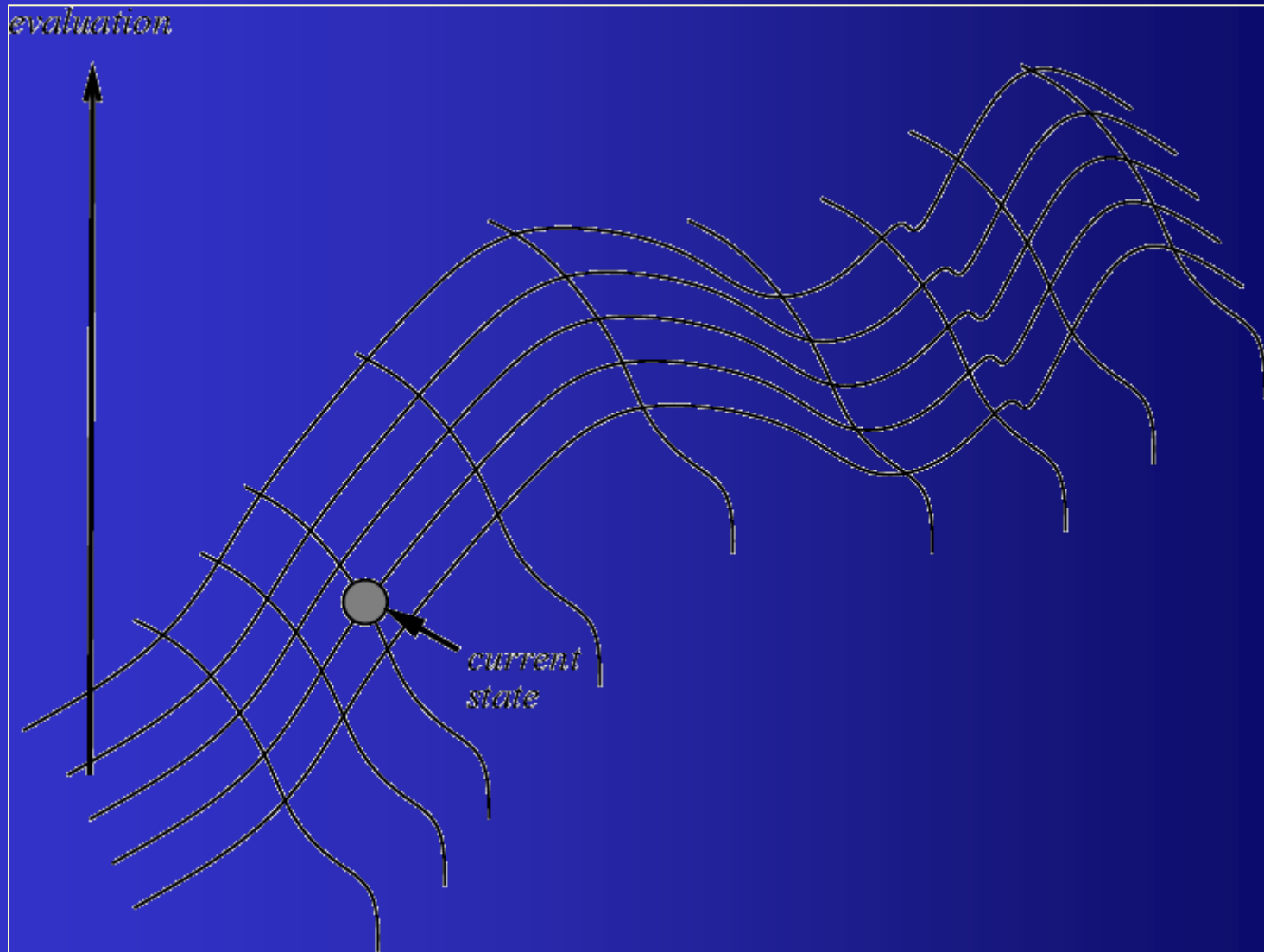
## ESCALADA - HILL CLIMBING

*Utiliza una función de evaluación en la prueba de la meta.*

*A partir de un estado, se realiza un bucle que constantemente se desplaza en la dirección ascendente, hasta encontrar una solución o atascarse.*

# Hill Climbing

## Superficie de espacio de estados



# Ascenso a la Cima (Hill-Climbing)

- continuamente se desplaza en la dirección de valor que más crece - elegir el mejor siguiente *estado inmediato*)

## Características

- no mantiene un árbol de búsqueda
- descarta información de ruta
- depende de la estructura de la “superficie” del espacio de estados

# ALGORITMO DE ESCALADA – HILL CLIMBING

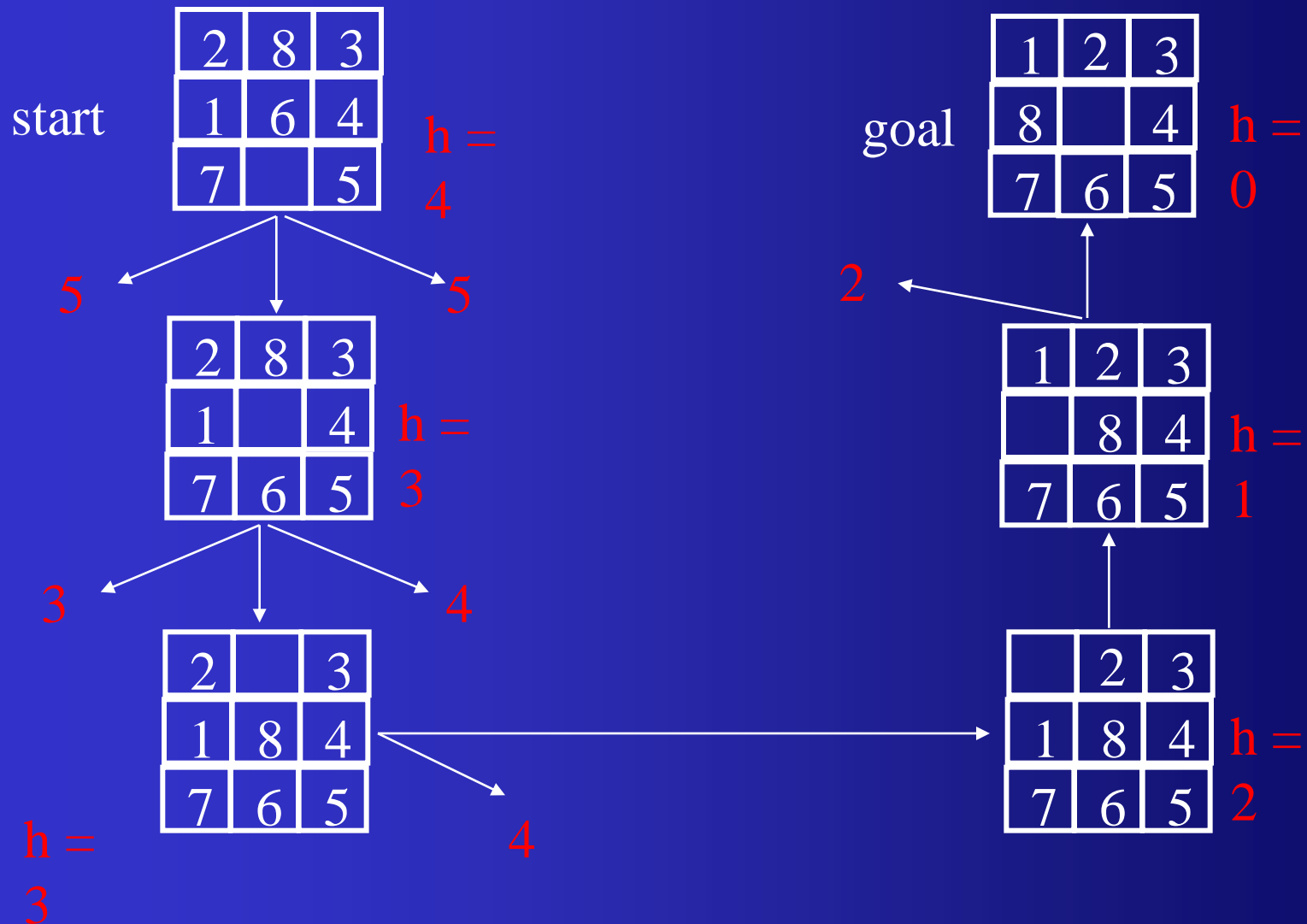
## PROBLEMAS:

- ✓ **Máximos locales:** *el algoritmo para aunque no ha encontrado la solución.*
- ✓ **Mesetas:** *la función de evaluación devuelve valores iguales, la búsqueda no tiene dirección.*
- ✓ **Crestas:** *si los operadores no se desplazan por la cima de la cresta, la búsqueda avanza poco.*

## SOLUCIONES:

- Arrancar con otra configuración inicial .
- Backtrack y tomar otra dirección
- Saltar a otra sección del espacio
- Aplicar dos o más reglas antes de evaluar

# Hill climbing ejemplo



$h(n) = (\text{cantidad de números fuera de lugar})$

# Endurecimiento simulado

- *En metalurgia y termodinámica se menciona el forjado o endurecimiento como el proceso de inicio a alta temperatura y enfriamiento gradual para obtener transiciones de fase más estables que las obtenidas por enfriamiento rápido.*
- ENDURECIMIENTO SIMULADO proceso de búsqueda global u optimización global en sistemas de comportamiento estocástico, con alguna probabilidad que es función de una “temperatura” (un cierto parámetro que desciende) con lo cual la conducta no es completamente determinística. La temperatura arranca siendo alta, y va descendiendo con un programa preestablecido (lentamente).

# Endurecimiento simulado

- Si el programa de enfriamiento es demasiado rápido (las transiciones de fase ocurren desordenadamente), mientras que si el programa de temperatura es suave, se logra mayor estabilidad, -- mínimo global en vez de alguno local-
- Pertenece a la familia de los métodos de búsqueda heurísticos, admite que haya pasos aparentemente en falso, que no mejoran la evaluación, pero esos pasos van disminuyendo en su probabilidad a lo largo del tiempo.
- La tasa con que se admiten aparentes pasos en falso (decrecientes) está regulado por un programa de enfriamiento.
- El método garantiza un óptimo global y no un subóptimo local si la temperatura baja con suavidad.



# Endurecimiento Simulado

- Elegir un movimiento al azar
  - si es mejor, ejecutarlo
  - si es peor, ejecutarlo con probabilidad que decrezca exponencialmente con lo “malo” del movimiento y la “temperatura”
- La temperatura cambia de acuerdo con un *programa*
  - si el programa hace descender la T en forma lenta, el algoritmo va a encontrar un óptimo global

# Endurecimiento Simulado

- Función de energía  $E(C)$  definida sobre el espacio de las configuraciones posibles  $\Omega$

- Distribución de Gibbs:

La probabilidad de una determinada configuración  $C$  decrece exponencialmente con su costo

- $$P(C) = 1/Z e^{-E(C)/T}$$

$Z$  es un factor de normalización

- $$P(C_{i+1})/P(C_i) = e^{-\Delta E/T}$$

- $$q = \min \{ 1, P(C_{i+1})/P(C_i) \}$$

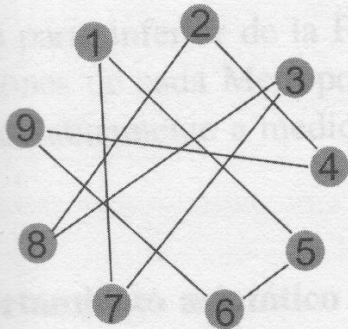
# Endurecimiento Simulado

```
Algoritmo SIMULATED_ANNEALING {  
   $C^0 \leftarrow \text{INICIALIZAR\_CONFIG};$   
   $T \leftarrow \text{INICIALIZAR\_TEMP};$   
   $t \leftarrow 0;$   
  Repetir {  
     $C \leftarrow C^t;$   
    Repetir {  
       $C' \leftarrow \text{GENERAR}(\mathcal{N}(C));$   
       $\Delta E \leftarrow E(C') - E(C);$   
       $q \leftarrow \min\{1, e^{-\Delta E/T}\};$   
      Si ( $\text{RANDOM}(0,1) < q$ ) {  $C \leftarrow C'$  };  
    } Hasta TEST_EQUILIBRIO;  
     $t \leftarrow t + 1;$   
     $C^t \leftarrow C;$   
     $T \leftarrow \text{ENFRIAR}(T, t);$   
  } Hasta  $T \approx 0;$   
  Devolver  $C^t \approx C^* = \arg \min_{C \in \Omega} E(C);$   
}
```

# Endurecimiento Simulado – Ejemplo

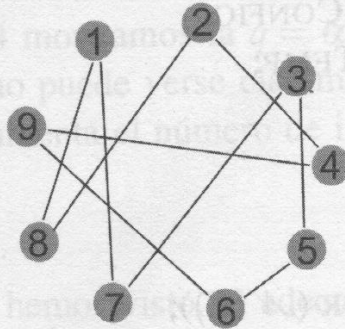
## TSP con 9 ciudades

C0 = [2 4 9 6 5 1 7 3 8]



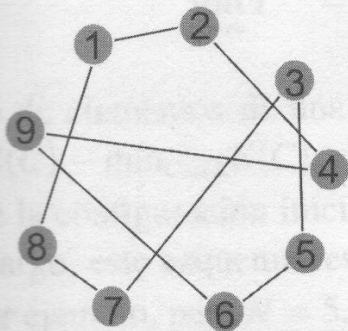
E0=21,5

C1 = [2 4 9 6 5 3 7 1 8]



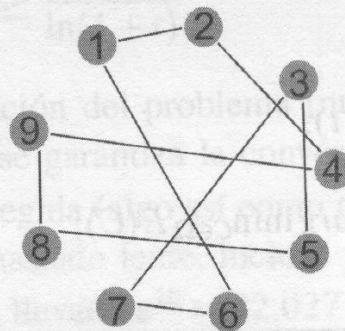
E1=18,5  
(Aceptar)

C2 = [2 4 9 6 5 3 7 8 1]



E2=16,0  
(Aceptar)

C3 = [2 4 9 8 5 3 7 6 1]



E3=18,5  
(Inc = 2,5, q=0.66, Aceptar)

En E3  $\Delta E = 2.5$   
Probabilidades de aceptar  
esta configuración,  
dependen de T (y r):

- T=6  $q = e^{-2.5/6} = 0.66$
- T=3  $q = 0.43$
- T=1  $q = 0.08$

# BUSQUEDA MEDIANTE LA SATISFACCION DE RESTRICCIONES

## Constraint Satisfaction Problem (CSP)

*Los estados se definen mediante los valores de un conjunto de variables y el objetivo se especifica mediante un conjunto de restricciones que los valores deben satisfacer.*

***La solución del problema:*** especificar valores para todas las variables tal que satisfagan todas las restricciones.

# CONSTRAINT SATISFACTION PROBLEM (CSP)

*Definición:  $(X, D, R)$*

- Variables  $V = \{X_1, \dots, X_n\}$
- Dominios  $D_1, \dots, D_n$  de cada variable

(contínuos- discretos y finitos)

- Restricciones  $R = \{R_1, \dots, R_k\}$   $R_j \subset D_{j1} \times \dots \times D_{jk}$

(de k-variables-binarias)(obligatorias– preferencia)

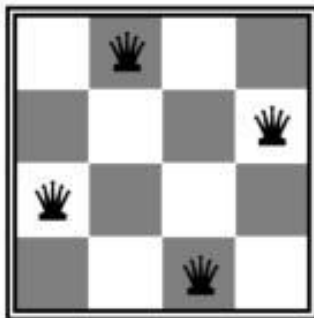
✓ Cada restricción plantea relaciones entre los valores de una/algunas variables.



# Ejemplos de CSP:

- ✓ Criptoaritmética
- ✓ El problema de las N-reinas
- ✓ Problemas de diseño y planificación
- ✓ 3SAT
- ✓ Alimentación de ganado vacuno

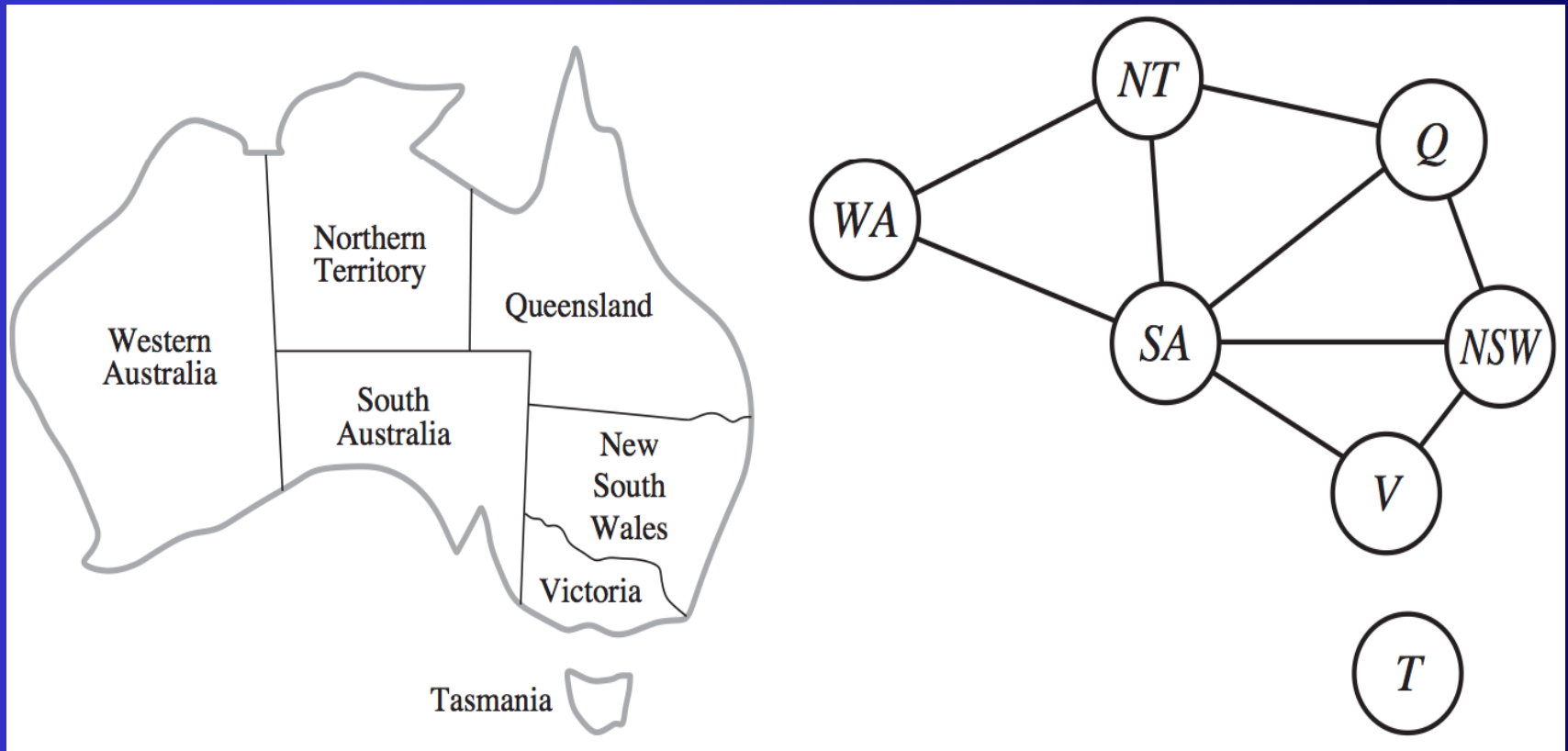
# Constraint Satisfaction Problems



8			4		6			7
						4		
	1					6	5	
5		9		3		7	8	
				7				
	4	8		2		1		3
	5	2					9	
		1						
3			9		2			5



# Ejemplos de CSP: coloreo de grafos



# EJEMPLOS 1

- Variables: s,e,n,d,m,o,r,y
- Dominios: s,e,n,d,m,o,r,y  $\in \{0, \dots, 9\}$
- Restricciones

$$\begin{array}{r} \text{s e n d} \\ + \text{m o r e} \\ \hline \text{m o n e y} \end{array}$$

$$10^3(s+m) + 10^2(e+o) + 10(n+r) + d+e = 10^4m + 10^3o + 10^2n + y$$

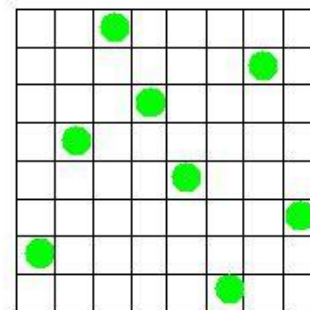
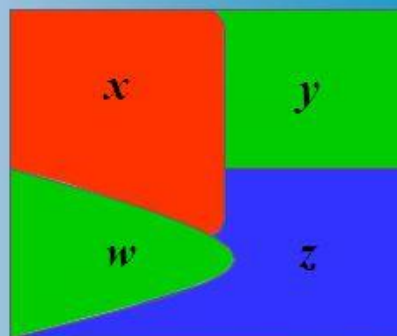
## Objetivos

- Consistencia
- Soluciones

## Coloreado de Mapas

- Variables: x,y,z,w
- Dominios: x,y,z,w : {r,v,a}
- Restricciones: *binarias*

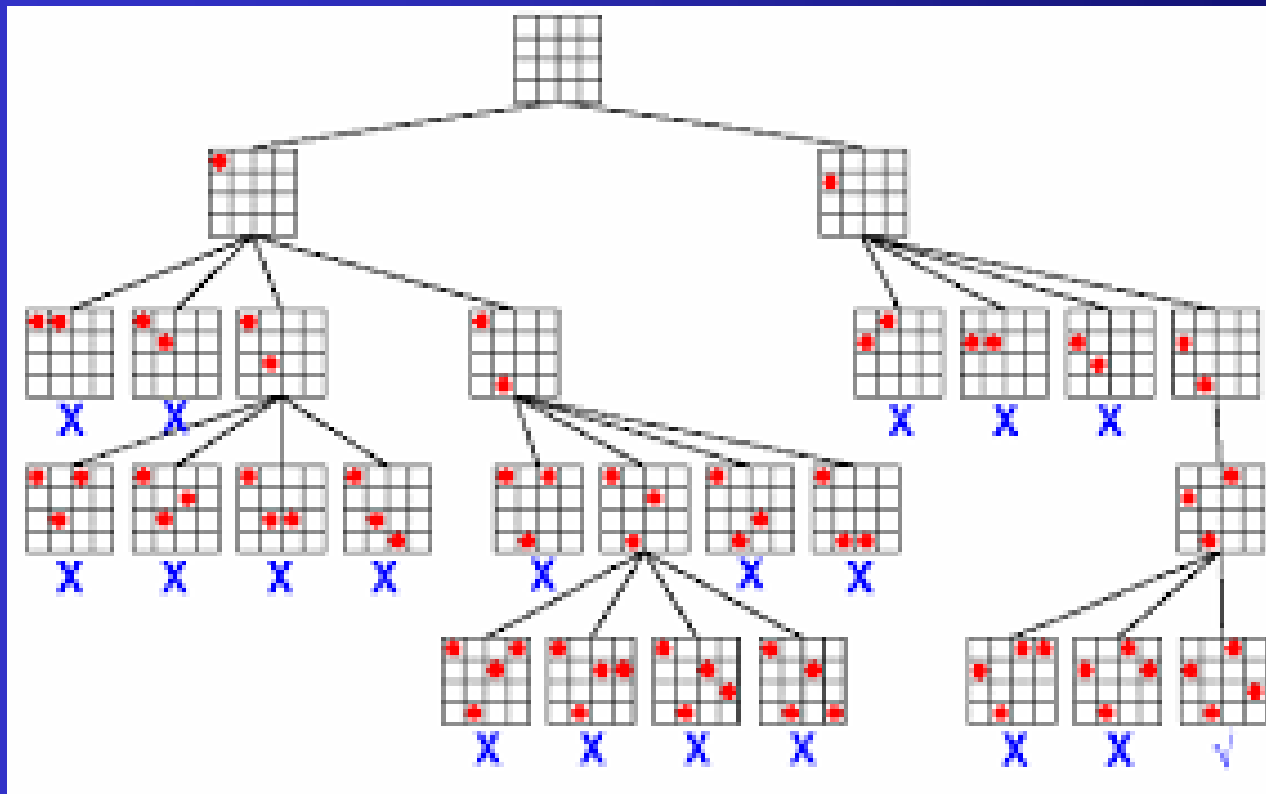
$$x \neq y, y \neq z, z \neq w, \dots$$



**El Problema de las 8 Reinas...**

# Formulación incremental

## Búsqueda con Backtracking



## Complejidad de los CSP: NP-completos

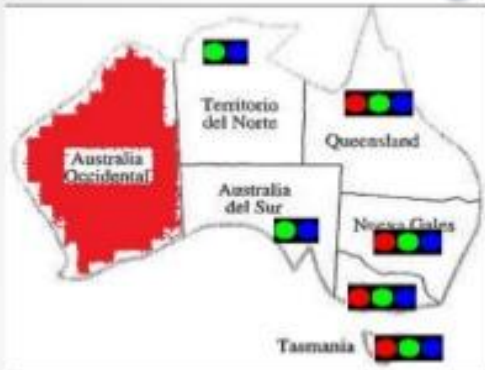
*En muchos de los problemas reales se aprovecha la estructura del problema para reducir el espacio de búsqueda.*

## Algoritmos:

*se pueden utilizar algoritmos de búsqueda general, pero tienen mayor eficiencia algoritmos diseñados especialmente (verificación anticipada, consistencia de arco, etc)*

# Verificación Anticipada

## COMPROBACIÓN HACIA ADELANTE



1- AO = ROJO

\* La Comprobación hacia adelante suprime rojo de los dominios de TN y AS

Despues de AO=rojo

AO	TN	Q	NGS	V	AS	T
R V A	R V A	R V A	R V A	R V A	R V A	R V A
R	V A	R V A	R V A	R V A	V A	R V A
R						R V A
R						R V A

# Arco consistencia

26

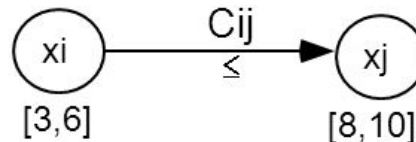
## Consistencia: Niveles 2-consistencia

---

Consistencia de arco (2-consistencia):

Un arco  $(x_i \{c_{ij}\} x_j)$  es consistente si y solo si para cada asignación de  $x_i$  en su dominio, existe una asignación para  $x_j$ , tal que la restricción  $\{c_{ij}\}$  se satisface.

Por ejemplo el arco:



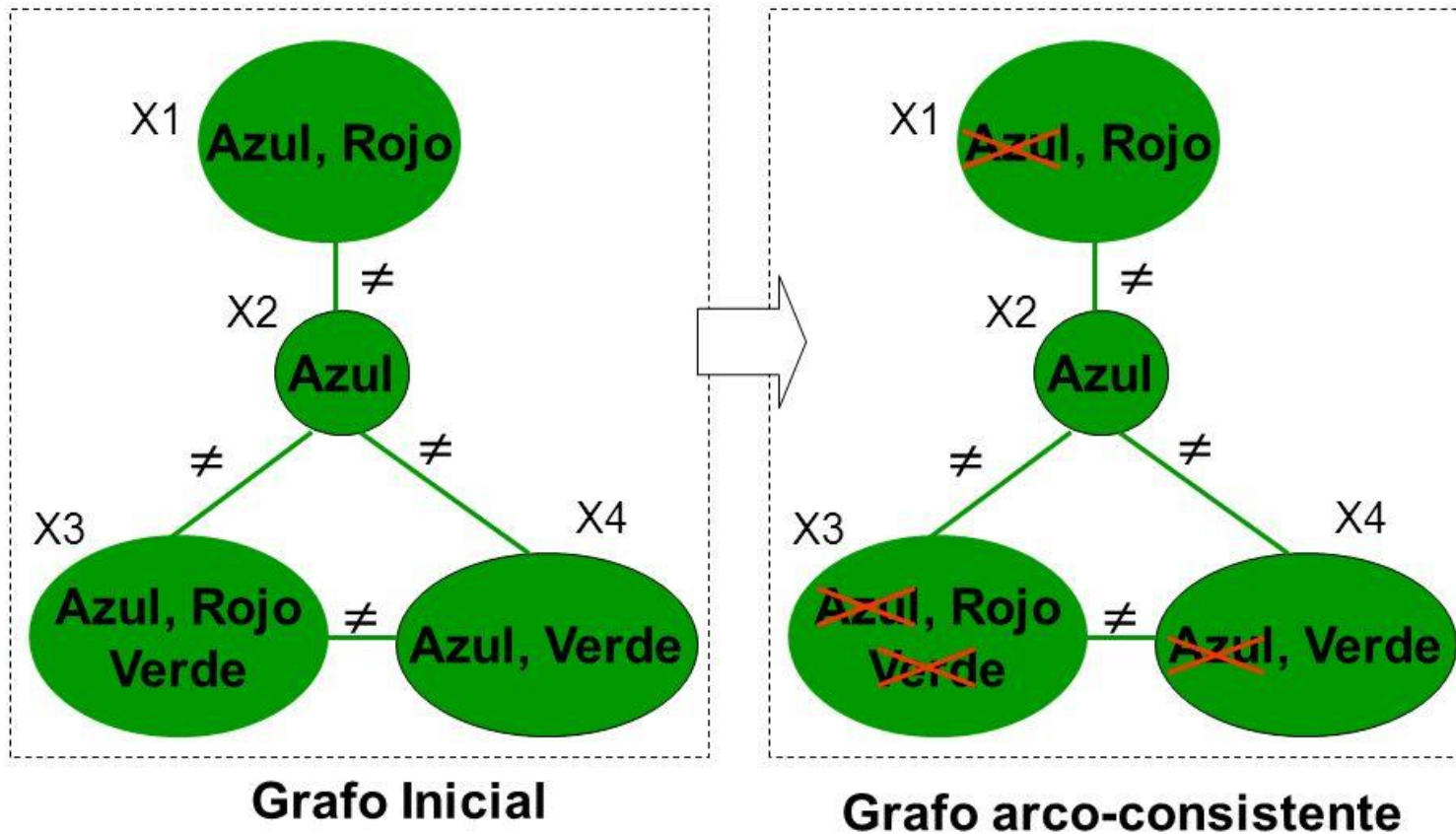
es consistente, pero no lo sería si  $c_{ij}$  en vez de  $\leq$  fuese  $\geq$

Un grafo es *arco-consistente* si todos sus arcos son consistentes.

$$\forall c_{ij} \subseteq \text{CSP}, \forall v_i \in d_i \exists v_j \in d_j / (x_i \ c_{ij} \ x_j) \text{ se cumple para } x_i=v_i, x_j=v_j$$

# Consistencia: Niveles

## 2-consistencia





# Conclusiones - Búsqueda

- Si no sabemos cómo llegar a  $X$ , creemos un espacio de estados donde sepamos que va a estar incorporado  $X$  y luego busquemos a  $X$  dentro de ese espacio.
- Mérito de esta formulación  $\Rightarrow$  siempre es posible encontrar un espacio donde esté contenida la respuesta o solución.
- Cuanto menos conocimiento tengamos, tanto más grande será el espacio de búsqueda.
  - **La incorporación de heurísticas lo reduce !!!**



# Bibliografía

- Inteligencia Artificial, Un enfoque moderno. S. Russell & P. Norvig, Prentice Hall,
- Inteligencia Artificial. Modelos, Técnicas y Areas de Aplicación. Escolano F. et al., Thomson, 2003.
- <http://pub.ufasta.edu.ar/ohcop/ayuda44.html>
- Materiales AI CS188 Univ. Berkeley