



PONTIFICIA  
**UNIVERSIDAD**  
**CATÓLICA**  
DEL PERÚ

## *INTELIGENCIA ARTIFICIAL (INF371)*

### *RESOLUCIÓN DE PROBLEMAS CON BÚSQUEDA: ESTRATEGIAS DE BÚSQUEDA CON INFORMACIÓN*

Dr. Edwin Villanueva Talavera

- Estrategias de Búsqueda con Información
  - ▣ Búsqueda codiciosa
  - ▣ Búsqueda  $A^*$
  - ▣ Heurísticas

## Bibliografía:

Capítulo 3.5 y 3.6 del libro:

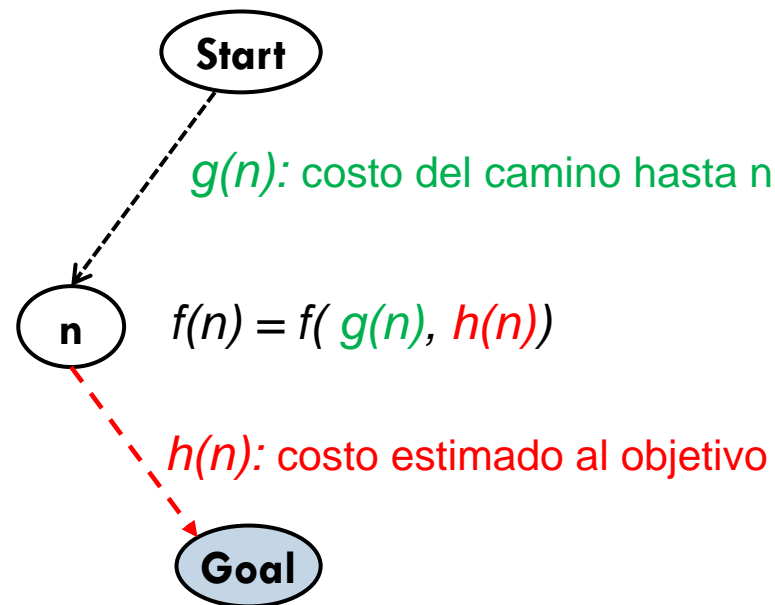
Stuart Russell & Peter Norvig “[Artificial Intelligence: A modern Approach](#)”,  
Prentice Hall, Third Edition, 2010

- Utiliza conocimiento específico sobre el problema para encontrar soluciones de forma mas eficiente que la búsqueda ciega.
  - ▣ Conocimiento específico adicional a la definición del problema.
- Enfoque general: **búsqueda por la mejor opción.**
  - ▣ Utiliza una función de evaluación para cada nodo.
  - ▣ Expande el nodo que tiene la función de evaluación más baja.

# Búsqueda por la mejor opción



- Idea: usar una **función de evaluación**  $f(n)$  para cada nodo.
  - ▣  $f(n)$  es una estimación de cuán deseable es el nodo  $n$
  - ▣ Se expande el nodo mas deseable que aún no fue expandido
  - ▣  $f(n)$  es normalmente una combinación del **costo de camino**  $g(n)$  y de una **función de heurística**  $h(n)$  que mide el costo estimado para llegar al objetivo desde  $n$



# Búsqueda por la mejor opción



- Implementación:
  - ▣ Similar que Búsqueda de Costo Uniforme. La diferencia es que los nodos de la frontera son ordenados por la función de evaluación  $f(n)$ , en vez del costo de camino.
- La forma de  $f(n)$  determina la estrategia de búsqueda:
  - ▣ Busca codiciosa por la mejor opción
  - ▣ Busca  $A^*$

# Búsqueda por la mejor opción



## Implementación

```
function BEST-FIRST-GRAPH-SEARCH(problem, f) returns a solution, or failure  
  node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE  
  frontier  $\leftarrow$  a priority queue ordered by f, 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 And child.STATE is not in frontier then  
        frontier  $\leftarrow$  INSERT(child, frontier)  
      else if child.STATE is in some frontier node n with  $f(n) > f(child)$  then  
        replace frontier node n with child
```

- Es un tipo de búsqueda por la mejor opción donde la función de evaluación:  $f(n) = h(n)$  (**heurística**)  
= estimado del costo del camino mas barato para llegar al objetivo desde  $n$ .

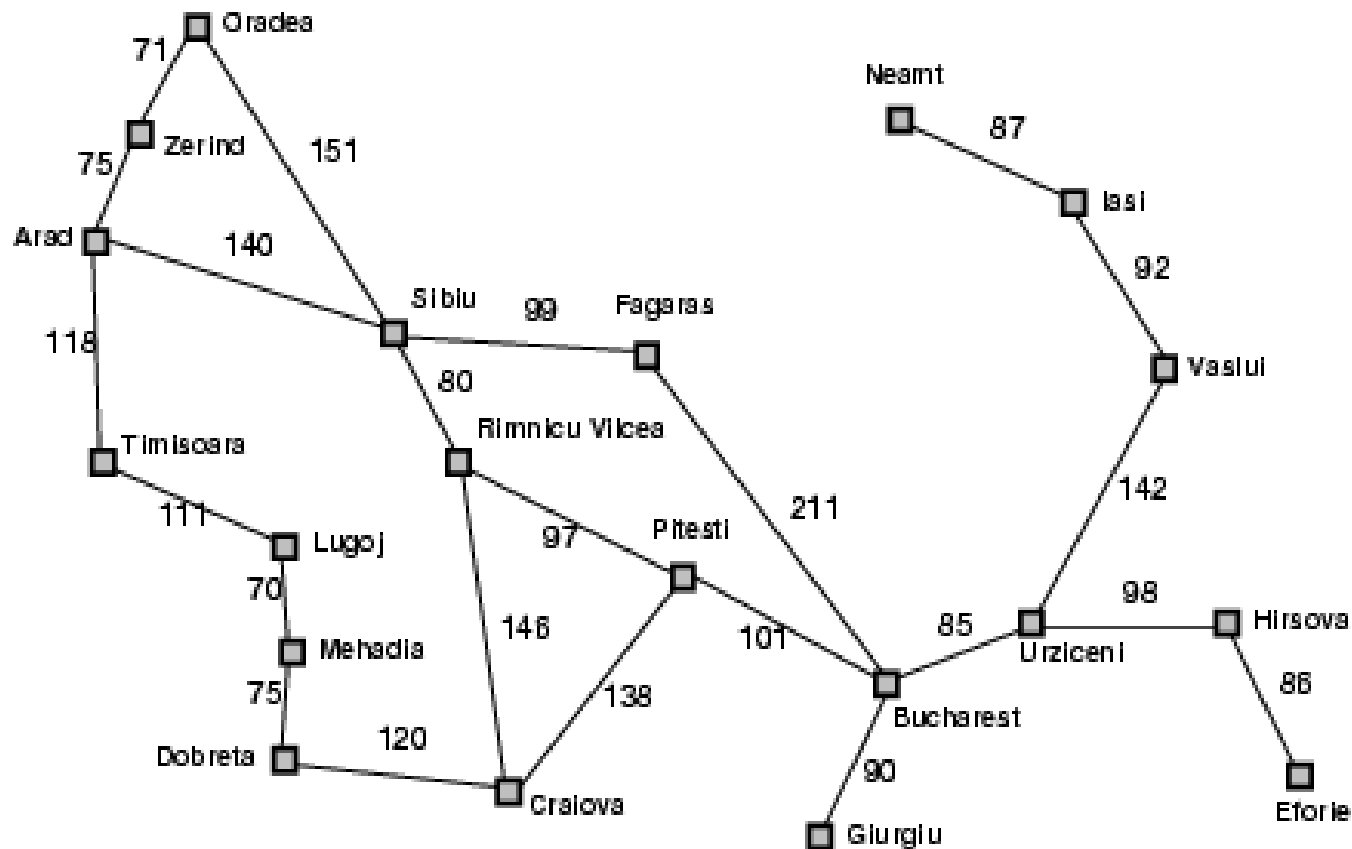
- ▣  $h(\text{nodo\_objetivo}) = 0$

Ejemplo:  $h(n) = h_{DLR}(n) = \text{distancia en línea recta desde } n \text{ hasta el objetivo.}$

- La búsqueda codiciosa o voraz expande el nodo en la frontera con menor  $h(n)$ , osea, el que **parece** que está mas próximo al objetivo de acuerdo a la función heurística.

# Búsqueda Voraz

- Ejemplo de búsqueda voraz en el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



Distancia en  
línea recta hasta  
Bucarest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



# Búsqueda Voraz



- Ejemplo de búsqueda voraz en el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



# Búsqueda Voraz

- Ejemplo de búsqueda voraz en el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



# Búsqueda Voraz

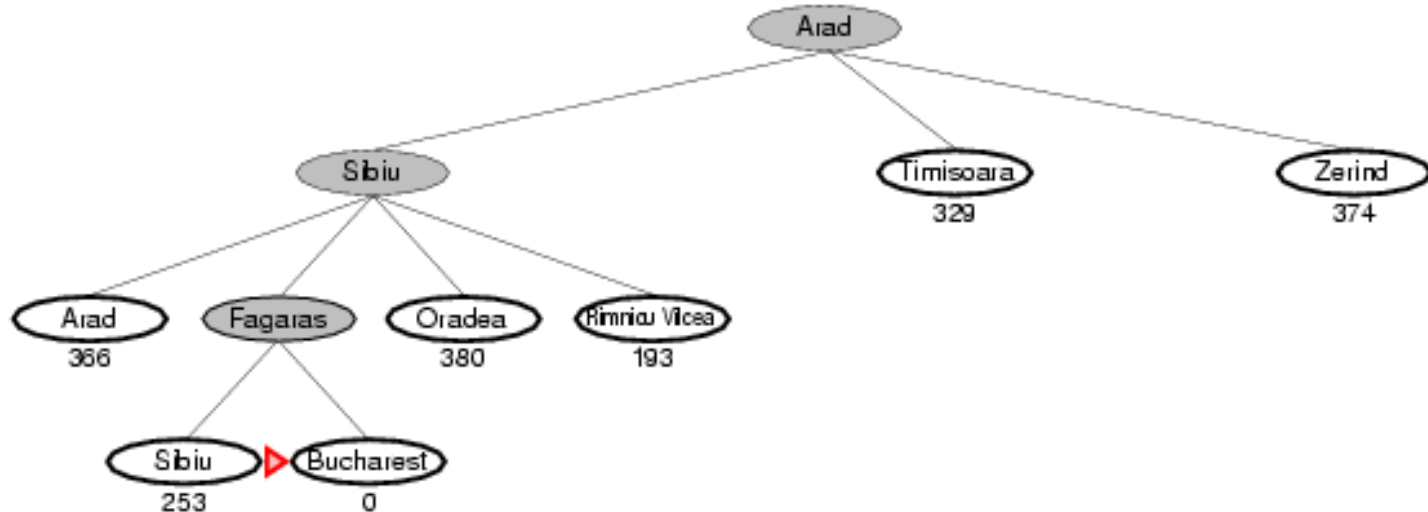


- Ejemplo de búsqueda voraz en el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



# Búsqueda Voraz

- Ejemplo de búsqueda voraz en el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



El camino via Rimnicu Vilcea y Pitesti es 32 km mas corto!

## Propiedades de búsqueda voraz

- **Completa?** **Si**, si chequea estados repetidos.
- **Complejidad de tiempo:**  $O(b^m)$  en el peor caso, pero una buena función heurística puede llevar a una reducción substancial
- **Complejidad de espacio:**
  - ▣  $O(b^m)$ , mantiene todos los nodos en memoria.
- **Optima?** **NO**. Puede haber un camino mejor siguiendo algunas opciones peores en algunos nodos del árbol de búsqueda

# Búsqueda A\*

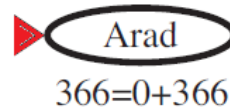
- Es la forma mas común de búsqueda por la mejor opción
- Função de avaliação  $f(n) = g(n) + h(n)$ 
  - ▣  $g(n)$  = costo del camino para alcanzar  $n$
  - ▣  $h(n)$  = costo estimado del camino mas barato de  $n$  hasta el objetivo
  - ▣  $f(n)$  = costo total estimado del camino mas barato hasta el objetivo pasando por  $n$

# Búsqueda A\*



PONTIFICIA  
UNIVERSIDAD  
CATÓLICA  
DEL PERÚ

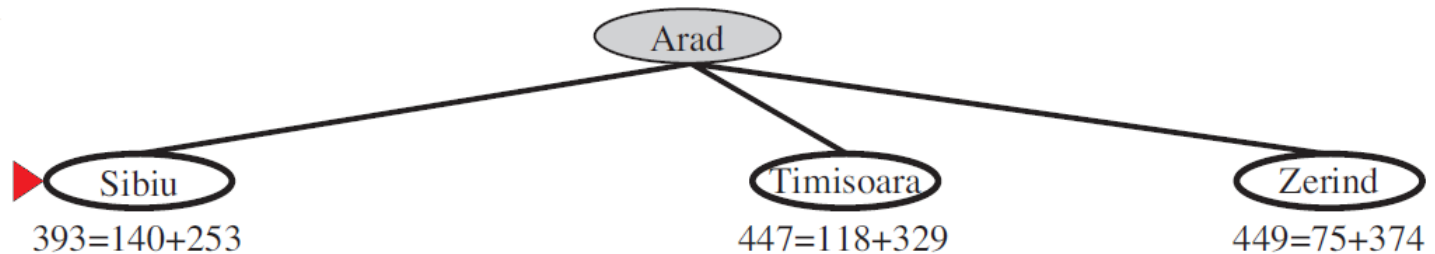
- Ejemplo de búsqueda A\* el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



# Búsqueda A\*



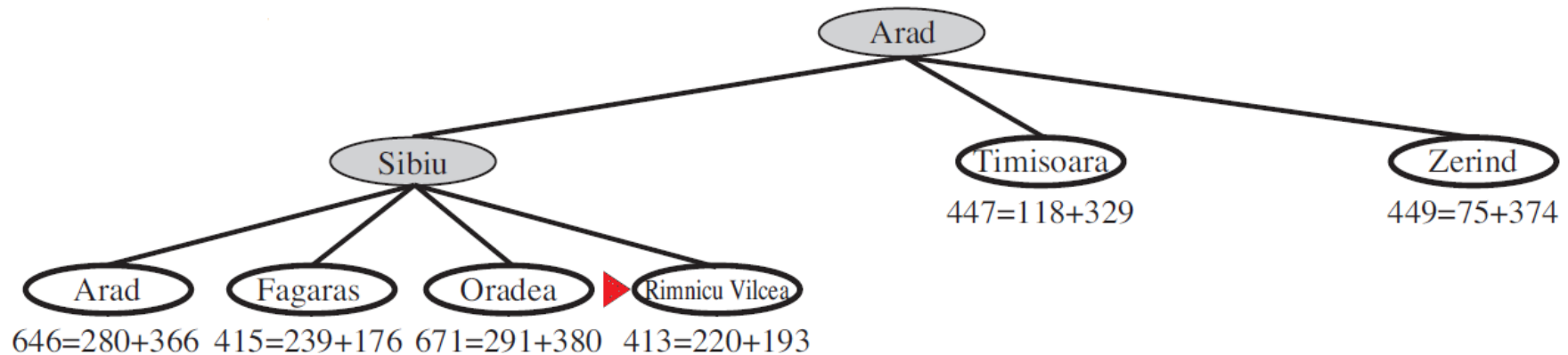
- Ejemplo de búsqueda A\* el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :





# Búsqueda A\*

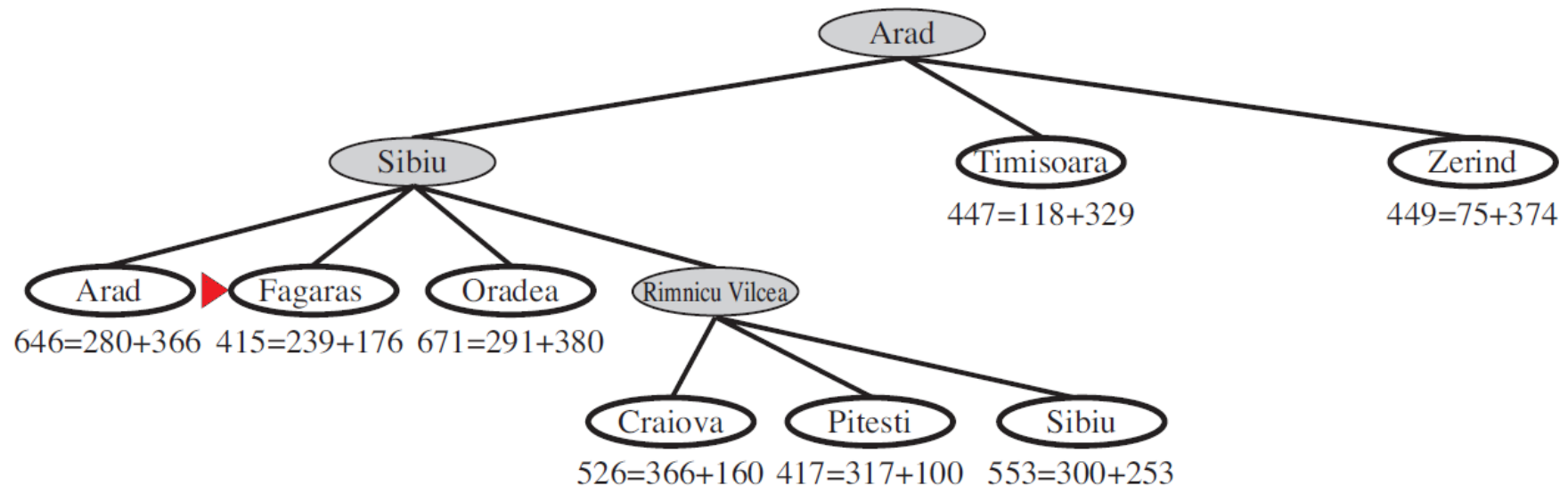
- Ejemplo de búsqueda A\* el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



# Búsqueda A\*



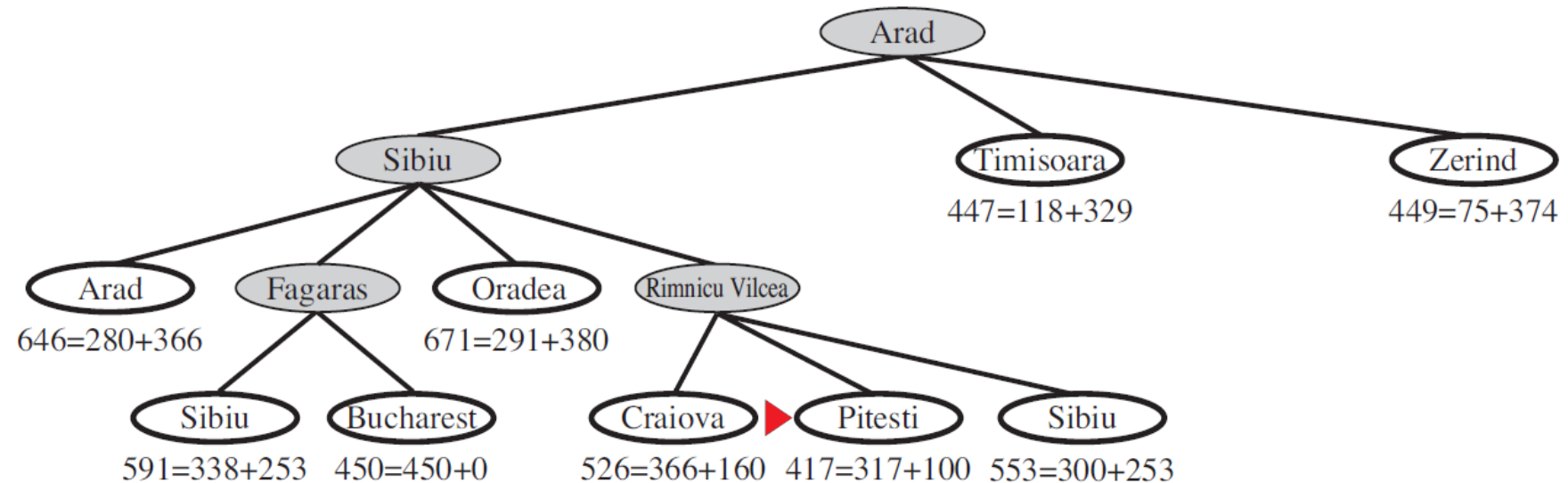
- Ejemplo de búsqueda A\* el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



# Búsqueda A\*



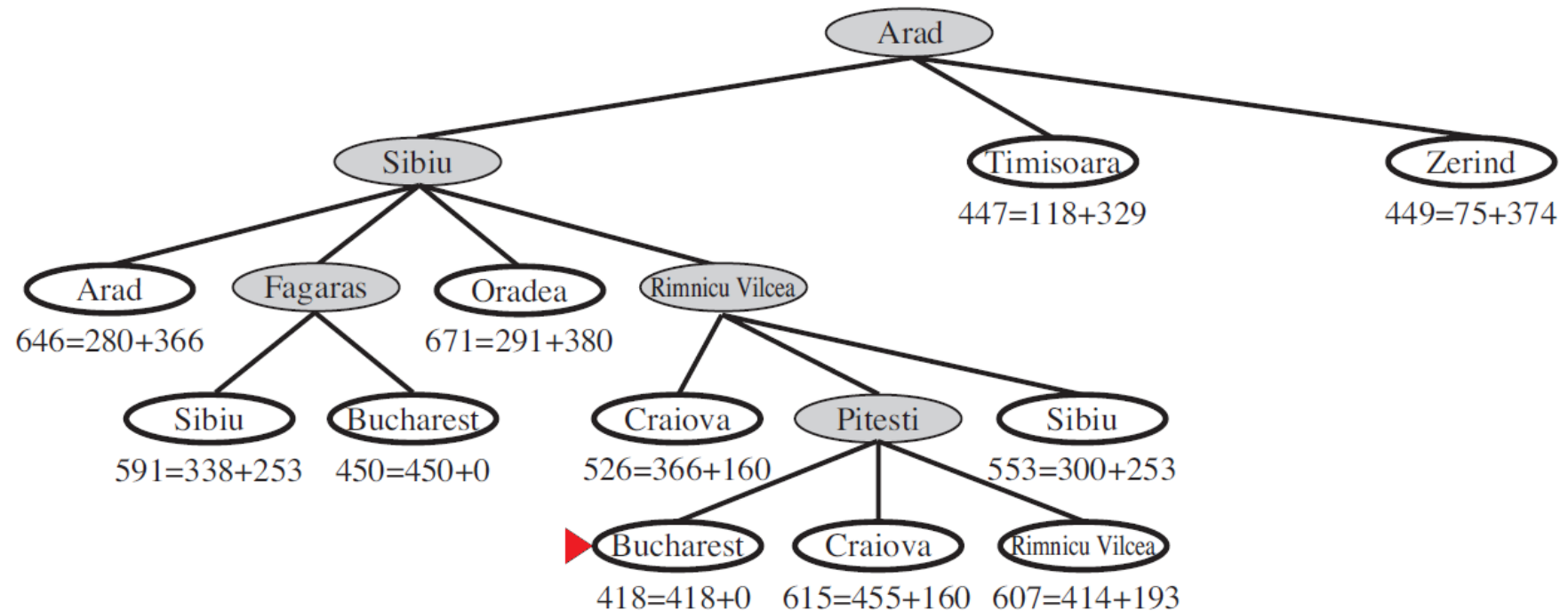
- Ejemplo de búsqueda A\* el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



# Búsqueda A\*



- Ejemplo de búsqueda A\* el mapa de Rumania siendo el objetivo Bucharest y heurística  $h_{DLR}(n)$ :



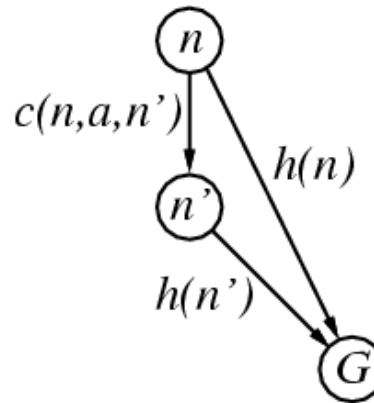
## Condiciones de Optimalidad: Admisibilidad

- Una heurística  $h(n)$  es **admisible** si para cada nodo  $n$  se verifica  $h(n) \leq h^*(n)$ , donde  $h^*(n)$  es el costo **verdadero** de alcanzar el estado objetivo a partir de  $n$
- Una heurística admisible **nunca sobreestima** el costo de alcanzar el objetivo, es decir, la heurística siempre es **optimista**.
  - ▣ Ejemplo:  $h_{DLR}(n)$  (distancia en línea recta nunca es mayor que distancia por las calles).
- **Teorema:** Si  $h(n)$  es admisible, A\* es optimo con búsqueda en árbol (sin memoria de estados visitados).
- Cuando se usa BEST-FIRST-GRAPH-SEARCH se necesita una condición mas estricta para garantizar optimalidad: **Consistencia**

## Condiciones de Optimalidad: Consistencia

- Una heurística  $h(n)$  es **consistente (o monotónica)** si para cada nodo  $n$  y sucesor  $n'$  generado por acción  $a$  se verifica que:

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



Es una forma de la desigualdad triangular (cada lado del triángulo no puede ser mayor que la suma de los otros lados).

- Toda heurística consistente es también admisible

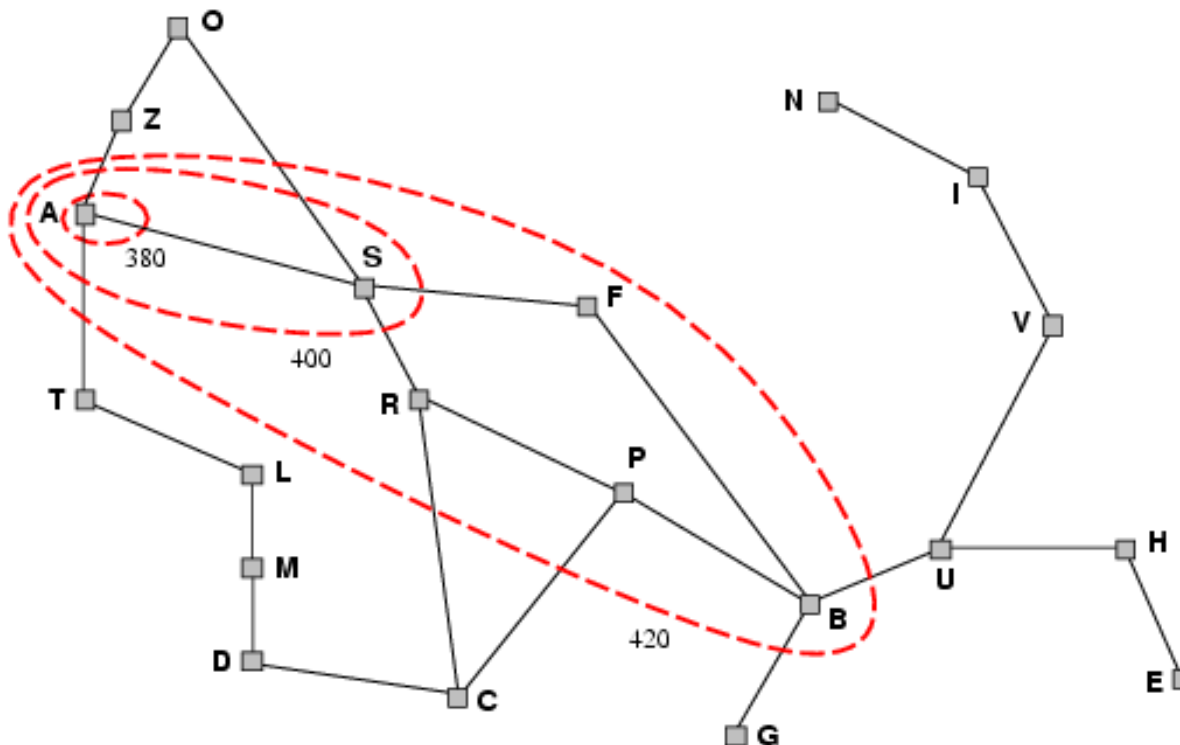
## Prueba de que heurísticas consistentes garantizan optimalidad

1. Probar que si  $h(n)$  es consistente entonces los valores  $f(n)$  no decrecen a lo largo de cualquier camino:
2. Probar que siempre que A\* selecciona un nodo  $n$  para expansión, el camino optimo hasta ese nodo ha sido encontrado.
  - Si no fuese el caso entonces existiría otro  $n'$  en la frontera que está en el camino optimo, el cual A\* lo habría escogido antes, ya que  $f(n') < f(n)$

# Búsqueda A\*

## Contornos de valores $f$ en el espacio de estados que A\* traza

- A\* expande nodos en orden creciente de valores de  $f$
- Gradualmente adiciona contornos de nodos
- Los estados fuera del contorno  $i$  tienen  $f > f_i$ , donde  $f_i < f_{i+1}$
- No expande nodos con  $f(n) > C^*$  (costo de la solución óptima)



Si  $h(n)=0$  tenemos una búsqueda de costo uniforme  $\Rightarrow$  círculos concéntricos.

Cuanto mejor la heurística mas direccionados son las elipses hacia el objetivo



## Propiedades de A\*

- **Completa?** Si, a menos que exista una cantidad infinita de nodos con  $f(n) < C^*$
- **Complejidad de tiempo:** Exponencial en el peor de los casos (heurística no apropiada)
- **Complejidad de espacio:** También exponencial en el peor caso ya que mantiene todos los nodos en memoria.
- **Optima?** SI, si heurística es admisible y consistente (Graph-search)
- **Óptimamente Eficiente:** Ningún otro algoritmo de búsqueda garantiza expandir un numero menor de nodos que A\* y encontrar la solución optima. Esto porque cualquier algoritmo que no expande todos los nodos con  $f(n) < C^*$  corre el riesgo de omitir una solución optima.

## Ejemplo de heurísticas admisibles

- Para el rompecabezas de 8 piezas:
  - $h_1(n)$  = número de piezas fuera de posición
  - $h_2(n)$  = distancia “Manhattan” total (para cada pieza calcular la distancia en movidas verticales y horizontales hasta su posición objetivo)

## Ejercicio

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $\underline{h_1(S)} = ?$
- $\underline{h_2(S)} = ?$

## Factor de ramificación efectiva de las heurísticas

- Factor de ramificación efectiva ( $b^*$ ):
  - Si el numero total de nodos generados por  $A^*$  es  $N$  y la profundidad de la solución  $d$ , entonces  $b^*$  es el factor de ramificación que un árbol uniforme de profundidad  $d$  tendría para contener  $N+1$  nodos:
$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$
  - Por ejemplo, si  $A^*$  encuentra la solución a un profundidad  $d=5$  expandiendo 52 nodos entonces  $b^* = 1.92$
  - Mejores heurísticas tienen valores mas próximos a 1

# Heurísticas

Factor de ramificación efectiva de las heurísticas  $h_1$  y  $h_2$  para el rompecabezas de 8 piezas

	Search Cost (nodes generated)			Effective Branching Factor		
$d$	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	3644035	227	73	2.78	1.42	1.24
14	—	539	113	—	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

## Dominancia:

- Una heurística  $h_2$  **domina** a otra heurística  $h_1$  si para todo nodo  $n$ :

$$h_2(n) \geq h_1(n)$$

- $h_2$  **es mejor que**  $h_1$  cuando  $h_2$  **domina**  $h_1$ 
  - ▣ La heurística dominante  $h_2$ , al tener mayores valores de  $h$  en cada nodo, expande nodos mas próximos del objetivo de que  $h_1$ , significando menos nodos expandidos.

## Como crear heurísticas admisibles: problema relajado

- El costo de la solución óptima de una simplificación del problema (problema relajado) puede ser una heurística para el original.

Por ejemplo, en el problema del rompecabezas de 8 piezas, la formulación original es:

7	2	4
5		6
8	3	1

Una pieza puede moverse del cuadrado A al cuadrado B si

A es horizontalmente o verticalmente adyacente a B y B es blanco

Podríamos generar los siguientes problemas relajados:

- Una pieza puede moverse del cuadrado A al cuadrado B
- Una pieza puede moverse del cuadrado A al cuadrado B si A es adyacente a B

El costo de la solución óptima del problema (I) sería  $h_1$ , (# piezas fuera de lugar), mientras que el costo de la solución óptima del problema (II) sería  $h_2$  (distancia Manhattan)

## Como crear heurísticas admisibles: problema relajado

- Las heurísticas generadas con problemas relajados son **admisibles**, ya que el costo de la solución del problema relajado nunca va ser mayor que el del problema original
- También son **consistentes** para el problema original si lo son para el problema relajado
- Si se tiene una colección de heurísticas admisibles  $h_1 \dots h_m$  y ninguna domina a las demás, se puede generar una nueva heurística  $h$  que domina a todas:

$$h(n) = \max\{h_1(n), \dots, h_m(n)\}$$

## Como crear heurísticas admisibles: Sub-problemas

1. Usar el costo de la solución optima de un sub-problema del problema original.

Ejemplo: Costo de colocar los 4 primeros números en sus lugares, sin importarse como queden los otros números \* (aunque sus movidas si cuentan en el costo)

*	2	4
*		*
*	3	1

Start State

	1	2
3	4	*
*	*	*

Goal State





Preguntas?