

## 8.2 Búsqueda A\* . Búsqueda óptima

- Una función heurística  $h(n)$  se dice que es *admisible* (garantiza la obtención de un camino de coste mínimo hasta un objetivo) cuando se cumple:

$$h(n) \leq h^*(n) \quad \forall n$$

- Decimos entonces que un algoritmo A que **utiliza una función heurística admisible** es un **algoritmo A\***
- Cuanto más correctamente estimemos  $h(n)$  menos nodos de búsqueda generaremos
- Problema: si nuestra función heurística nos devuelve un valor superior a  $h^*$ , para algún nodo, no se puede garantizar que encontremos la solución óptima

## 8.2 Pseudocódigo A\*

```

Alg A*
  listaInterior = vacío
  listaFrontera = inicio

  mientras listaFrontera no esté vacía
    n = obtener nodo de listaFrontera con menor f(n) = g(n) + h(n)
    listaFrontera.del(n)
    listaInterior.add(n)

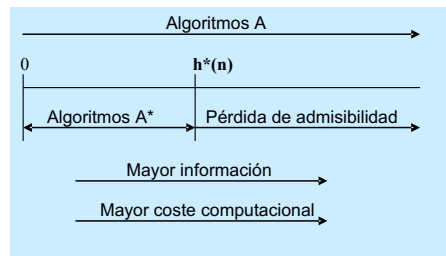
    si n es meta
      devolver
      reconstruir camino desde la meta al inicio siguiendo los punteros
    fsi

    para cada hijo m de n que no esté en listaInterior
      g'(m) = n.g + c(n, m) //g del nodo a explorar m

      si m no está en listaFrontera
        almacenar la f, g y h del nodo en (m.f, m.g, m.h)
        m.padre = n
        listaFrontera.add(m)
      sino si g'(m) es mejor que m.g //Verificamos si el nuevo camino es mejor
        m.padre = n
        recalcular f y g del nodo m
      fsi
    fpara
  fmientras
    devolver no hay solución
falga
  
```

## 8.3 Nivel de información heurístico

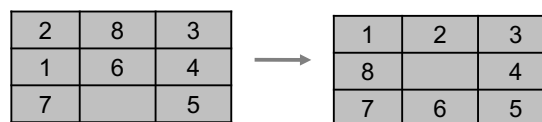
- En general el **nivel de información** de las heurísticas permite **encontrar antes la solución**, pero tiene la desventaja de requerir un **mayor coste computacional** para su cálculo. La figura muestra los **límites de la admisibilidad** en los algoritmos tipo A:



- $h(n) = 0$
- $h(n) \leq h^*(n)$
- $h(n) > h^*(n)$

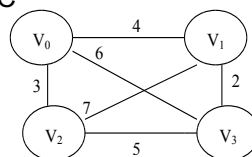
## 8.4 Ejemplos de heurísticas

- Heurísticas para el 8-puzzle



- $h_1$  = número de piezas mal colocadas
- $h_2$  = suma de las distancias de las piezas a sus posiciones en el objetivo (distancia de Manhattan)

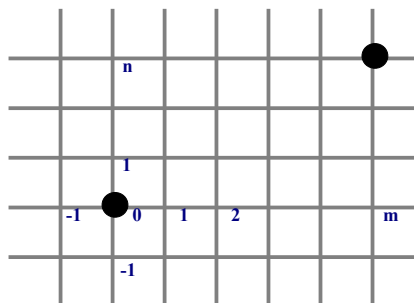
- Heurística para el VC



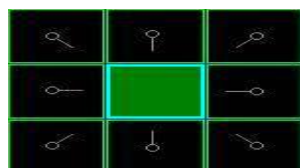
$h$  = suma de las distancias de las ciudades aun no visitadas a sus vecinos más cercanos

## 8.4 Problemas de camino mínimo

- Coste actual óptimo
  - $g^*((x, y)) = |x| + |y|$
- Heurística admisible
  - $h_1((x, y)) = \sqrt{(m-x)^2 + (n-y)^2}$
- Heurística óptima
  - $h^*((x, y)) = |m-x| + |n-y|$



## 8.4 Un ejemplo detallado (I)



$g(n)$  : coste de moverse entre nodos.  
recto 10, diagonal 14

$h(n)$  : distancia de Manhattan  
 $|m-x| + |n-y|$   
Atención, no admisible para 8-con.  
utilizada por simplicidad pero...

## 8.4 Un ejemplo (II)



T2. Estrategias de búsqueda

7

## 8.4 Un ejemplo (III)

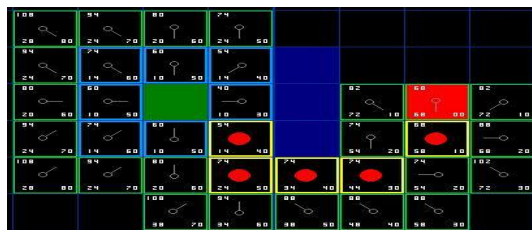


Ejercicio: continua la traza...

T2. Estrategias de búsqueda

8

## 8.4 Un ejemplo (IV)



T2. Estrategias de búsqueda

9

## 8.5 Inconvenientes de mantener la admisibilidad

- El **mantenimiento de la admisibilidad** fuerza al algoritmo a **consumir mucho tiempo** en discriminar caminos cuyos costes no varían muy significativamente
- Principal desventaja: se queda sin espacio debido a que mantiene todos los nodos generados en memoria
- No es práctico para problemas grandes
- Dos soluciones
  - Algoritmos que mejoran el coste espacial: A\*PI (A\* por profundización iterativa), A\* SRM (A\* acotada por memoria simplificada), búsqueda primero el mejor recursiva
  - Aumentar la velocidad a costa de una pérdida acotada de calidad → **técnicas de relajación de la restricción de optimalidad**

T2. Estrategias de búsqueda

10

## 8.6 Relajación de la restricción de optimalidad

- Técnica de ajuste de pesos
  - El objetivo de esta técnica es definir una función  $f()$  ponderada,  $f_w()$ , como alternativa a la utilizada en  $A^*$ 

$$f_w(n) = (1-w)g(n) + w h(n)$$
    - $g(n)$ 
      - Proporciona la componente en anchura de la búsqueda.
    - $h(n)$ 
      - Nos indica la proximidad al objetivo.
  - Variando de forma continua  $w$  dentro del rango  $0 \leq w \leq 1$  obtenemos **estrategias mixtas intermedias**.
  - Si  $h(n)$  es admisible tenemos que:
    - En el rango  $0 \leq w \leq 1/2$ ,  $A^*$  con  $f_w(n)$  también es admisible.
    - Dependiendo de la diferencia existente entre  $h(n)$  y  $h^*(n)$ ,  $A^*$  con  $f_w(n)$  puede perder la admisibilidad en el rango  $1/2 < w \leq 1$

## 8.6 Relajación de la restricción de optimalidad

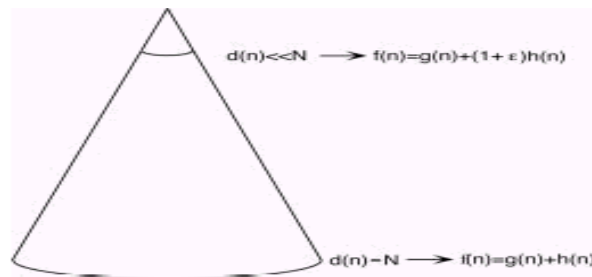
- Técnica de la admisibilidad- $\epsilon$ 
  - Objetivo: aumentar la velocidad de búsqueda a costa de obtener una solución subóptima
  - Un algoritmo es **admisible- $\epsilon$**  cuando para cualquier grafo termina siempre dando como resultado una solución cuyo coste no excede del coste óptimo,  $C^*$ , por un factor  $1+\epsilon$ :
 
$$f(sol) \leq (1+\epsilon)C^*$$
- Técnicas de admisibilidad- $\epsilon$ 
  - Ponderación Dinámica
  - Estimación de coste de búsqueda

## 8.6 Relaxación de la restricción de optimalidad

- Técnica de ponderación dinámica o APD

$$f(n) = g(n) + h(n) + \epsilon [1 - d(n)/N] h(n)$$

- $d(n)$  es la profundidad del nodo  $n$  y  $N$  nos proporciona la profundidad de un nodo solución (se supone conocida).
- ¿Qué ocurre en los niveles iniciales?
- ¿Y en los cercanos a la solución?



## 8.6 Relaxación de la restricción de optimalidad

- Algoritmo de estimación de coste de búsqueda  $A_\epsilon^*$ .

- Utiliza una lista adicional denominada **Lista\_focal ( $L_f$ )**
- Esta lista es una sublista de Lista\_Frontera (LF) que contiene únicamente aquellos nodos cuya  $f(n)$  no excede del mejor valor de cualquier  $f(n)$  dentro de la Lista\_Frontera por un factor  $(1+\epsilon)$ :
  - $$L_f = \{n: f(n) \leq (1+\epsilon) \min(f(m))\} \quad m \in LF$$
- $A_\epsilon^*$  opera de forma idéntica al algoritmo  $A^*$  salvo que **selecciona aquel nodo de Lista\_focal con menor valor de  $H_f(n)$ , una segunda heurística**, además de  $h(n)$ , que estima el coste computacional requerido para completar la búsqueda a partir del nodo  $n$

## 8.6 Relajación de la restricción de optimalidad

- Comparación de algoritmos
  - El algoritmo de ponderación dinámica es más sencillo, **pero únicamente es aplicable a problemas donde se conoce la profundidad** en la cual nos va a aparecer la solución, o disponemos de una cota superior de dicha profundidad. Sólo en estos casos se garantiza la admisibilidad- $\epsilon$
  - En cuanto al algoritmo  $A_\epsilon^*$  la separación en dos heurísticas permite incorporar estimaciones de coste no integradas con las funciones  $g(n)$  y  $h(n)$  (por ejemplo, proximidad a la solución)

## Tema 2. Estrategias de búsqueda.

### Bibliografía

- Stuart Russell, Peter Norvig. "Inteligencia Artificial. Un enfoque Moderno" Ed. Pearson. Prentice Hall. 2004.