Heurísticas

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

6 de julio de 2021

Antecedentes

Temas

Búsquedas ciegas

Tabla: Evaluación de estrategias de búsqueda.

Criterio	Primero en amplitud	Costo uniforme	Primero en profundidad	Profundidad limitada	Profundidad iterativa	Bidireccional (si aplicable)
¿Completa?	Sí[a]	Sí[a,b]	No[a2]	No	Sí[a]	Sí[a,d]
Tiempo	$O(b^{d+1})$	$O(b^{C^*/\epsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Espacio	$O(b^{d+1})$	$O(b^{C^*/\epsilon})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
¿Óptima?	Sí[c]	Sí	No	No	Sí[c]	Sí[c,d]

b	Factor de ramificación
d	Profundidad de la solución más cercana a la raíz
m	Máxima profundidad del árbol de búsqueda, puede ser ∞
l	Límite impuesto a la profundidad

a	Completa si b es finito
b	Completa si el costo por un paso es $\geqslant \epsilon > 0$
c	Óptima si todos los costos son idénticos
d	Si ambas direcciones utilizan búsqueda en amplitud

Definiciones

Temas

Función heurística

Definición

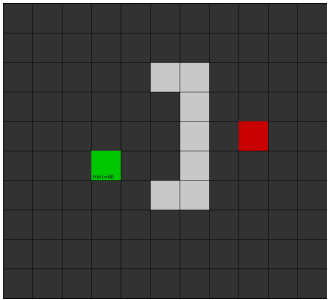
- Una **función heurística** $h(n)$, $h : S \to \mathbb{R}$ donde S es el espacio de estados, es el costo **estimado** del camino más barato desde el nodo n a un nodo objetivo.
- Evidentemente, si n es un nodo meta $h(n) = 0$.

Temas

Heurística admisible

Definición

Una **heurística admisible** es aquella que nunca sobrestima el costo de alcanzar un objetivo.



Posibles heurísticas: Ignorando los obstáculos.

- *Línea recta*
  $d = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}$
- *Distancia Manhattan*
  $d = |x_f - x_i| + |y_f - y_i|$
  no es admisible.

TIP: *Admisible* $\rightarrow$ *optimista*.

Temas

Heurística consistente

Definición

Una heurística es **consistente** o **monótona** si para cada nodo n y cada sucesor n' de n generado por cualquier acción a:

$$h(n) \leqslant c(n, a, n') + h(n') \tag{1}$$

donde $c(n, a, n')$ es el costo de alcanzar el objetivo desde n pasando por n'.

Si $h(n)$ es consistente, entonces los valores de $f(n)$, a lo largo de cualquier camino, no disminuyen.

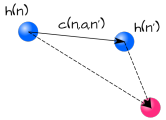
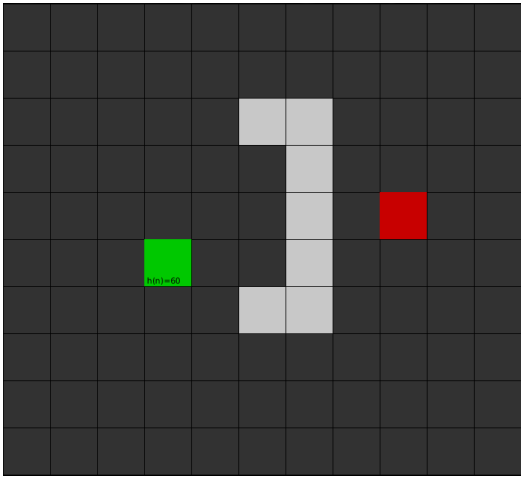


Figura: **TIP:** *Consistente → desigualdad del triángulo.*

Mapa en rejilla



Posibles heurísticas: Ignorando los obstáculos.

- *Línea recta*
  $d = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}$
- *Distancia Manhattan*
  $d = |x_f - x_i| + |y_f - y_i|$

Ambas heurísticas son consistentes.

Diseño de heurísticas

Temas

Diseño de heurísticas

- Para diseñar una heurística se pueden seguir los siguientes pasos:
  1. Simplificar el problema reduciendo las restricciones (**problema relajado**).
  2. Calcular el costo de resolver el problema relajado.
  3. El costo de una solución óptima en un problema relajado es una heurística admisible y consistente para el problema original.
- Al probar diferentes combinaciones de restricciones eliminadas se pueden obtener heurísticas distintas.

Temas

Ejemplo: Tablillas deslizantes de 8×8

La regla de transición es:

> *"Una tablilla se puede mover del cuadro A al cuadro B si A es horizontal o verticalmente adyacente a B y B está vacío".*

Las condiciones simplificadas eliminan alguna o ambas restricciones:
Una tablilla se puede mover del cuadro A al cuadro B si:

1. A es horizontal o verticalmente adyacente a B (distancia Manhattan).
2. B está vacío (tablillas en el lugar equivocado: puedo teletransportar la tablilla que va en el hueco).
3. *siempre* (tablillas en el lugar equivocado).

Figura: Tablero para el juego de las tablillas deslizantes.

- $h(n)$ = suma de las distancias de las piezas a sus posiciones objetivo. A la suma de las distancias horizontales y verticales se le llama **distancia en la ciudad** o **distancia Manhattan**.
- $h(n)$ = número de piezas mal colocadas.

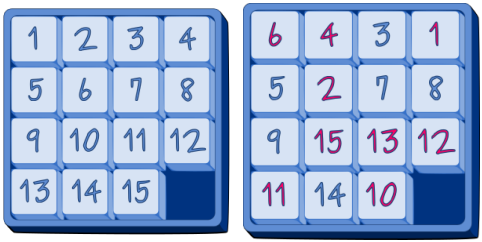


Figura: Las heurísticas ofrecen un estimado de la distancia entre el estado actual y la meta.

Algoritmos de búsqueda local

Temas

Algoritmos de búsqueda local

- Los algoritmos de **búsqueda local** funcionan con un sólo **estado actual** y
- generalmente se mueve sólo a los vecinos del estado;
- el camino seguido para encontrar la solución es irrelevante y es olvidado.
- Cada estado tiene un valor asociado dado por una **función objetivo**,
- el objetivo del algoritmo es encontrar el estado con el mejor valor.

Temas

4. Algoritmos de búsqueda local
   - Características
   - Ascenso/descenso de colinas
   - Recocido simulado

Ascenso/descenso de colinas

Búsqueda local voraz

- Siempre se mueve cuesta arriba guiado por el valor de la heurística.
- La función de evaluación de un nodo n es $f(n) = h(n)$.
- No tiene memoria y termina cuando no hay vecinos con un mejor valor.
- Si más de un valor sucesor tiene el valor óptimo, elige uno al azar.
- Cuando funciona, es el más eficiente tanto en tiempo como en espacio.

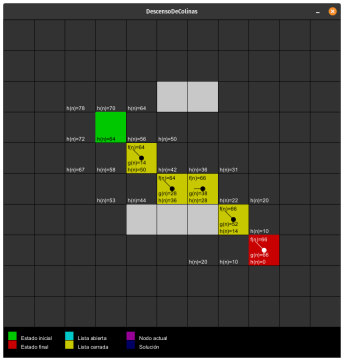


Figura: Descenso de colinas es un algoritmo voraz.

Desventajas

- Se atasca en máximos locales por lo que puede fallar en encontrar un máximo global.
- Le cuesta trabajo caminar sobre crestas (sucesión de máximos locales desconectados).
- No sabe cómo elegir un camino en una meseta.

Por ejemplo se puede aplicar al problema de las reinas con la heurística $\mathfrak{h} =$ número de pares de reinas que se atacan la una a la otra.

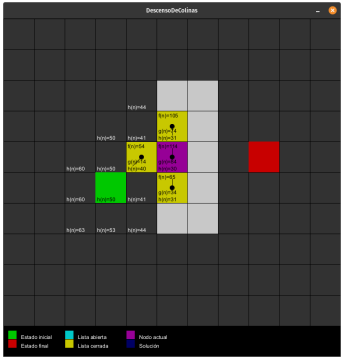


Figura: Descenso de colinas se atora en los mínimos locales.

Temas

4. Algoritmos de búsqueda local
   - Características
   - Ascenso/descenso de colinas
   - Recocido simulado

Recocido

"El recocido es un proceso de tratamiento térmico utilizado para reducir la dureza, aumentar la flexibilidad (ductilidad) y ayudar a eliminar las tensiones internas creadas en el metal durante el temple."

Figura: **Izquierda:** Horno para recocido. **Derecha:** Microestructura de latón recocida.

Fuentes:

- https://www.bodycote.com/es/servicios/tratamiento-termico/annealing-normalising/recocido/
- https://www.templesindustrialesalcala.es/recocido/
- https://commons.wikimedia.org/wiki/File:Microstructure_of_rolled_and_annealed_brass;_magnification_400X.jpg
- https://commons.wikimedia.org/wiki/File:Annealing_furnace_Saltford_Brass_Mill.jpg

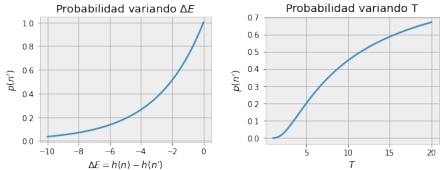
Recocido simulado *Simulated annealing*

- Es similar al **ascenso** de colinas, pero en vez de escojer siempre el mejor movimiento, hace una elección aleatoria.
- Al inicio, puede aceptar movimientos con peores calificaciones, pero esta probabilidad disminuye con el tiempo, conforme se decrementa la temperatura T.

$$p(n') = e^{\frac{\Delta E}{T}}$$

$$\Delta E = h(n') - h(n)$$

donde $p(n')$ determina la probabilidad de elegir al estado n' cuando su valor $h(n')$ es menor que el que se tenía en n.



Algoritmo

Algoritmo 1 Recocido simulado

1: **while** $h(n) > \varepsilon$ **do**
2: Elige a un vecino n' **al azar**.
3: **if** $h(n')$ es un mejor estado **then**
4: $n \leftarrow n'$ ▷ Se mueve a n'
5: **else** ▷ n' no es un mejor estado
6: $\Delta E < 0 \Rightarrow$ decidir con probabilidad $p(n')$ si se mueve a ese estado.

Para realizar un **descenso** basta con redefinir:

$$\Delta E = h(n) - h(n')$$

Búsqueda primero el mejor

Temas

Búsqueda primero el mejor

Búsqueda primero el mejor se refiere a una familia de algoritmos que busca expandir siempre **al mejor candidato**.

- Utilizan una *función de evaluación* que define la estrategia de selección.

$$f : S \to \mathbb{R} \tag{2}$$

- f puede utilizar a la función heurística.
- Un ejemplo es el *costo total estimado* $f(n)$:

$$f(n) = g(n) + h(n) \tag{3}$$

Donde $g(n)$ es el costo actual de llegar desde el nodo inicial al nodo n y $h(n)$, el valor de la heurística para el nodo n.

Temas

Búsqueda en amplitud (ciega)

Si el costo de las acciones es uniforme, o sólo importa obtener la solución con el menor número de acciones, sea:

$$f(n) = g(n) \tag{4}$$

con $g(n)$ el número de pasos, entonces *búsqueda en amplitud* nos dará la mejor solución.

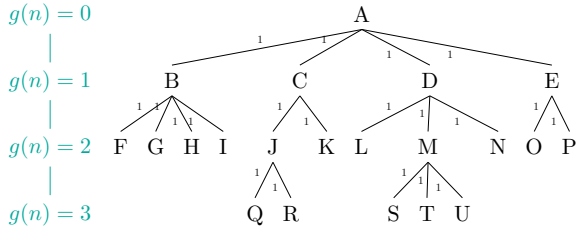


Figura: Amplitud encuentra primero la solución más cercana a la raíz.

Costo uniforme (ciega)

Si las acciones tienen un costo distinto, sea $g(n)$ el costo acumulado desde el nodo inicial hasta el nodo actual.

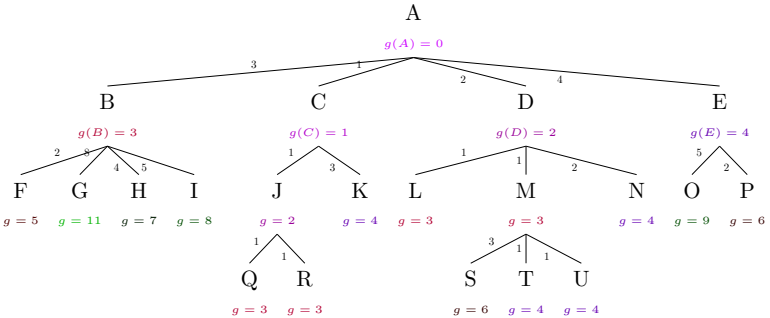


Figura: Trazar curvas de nivel usando el costo acumulado $g(n)$ muestra cómo son visitados los nodos por *Costo uniforme*.

Temas

A*

- A^* es el algoritmo de búsqueda primero el mejor que, aún siendo completo, garantiza utilizar la menor cantidad posible de recursos tanto en memoria como en tiempo si la heurística utilizada es **admisible** y **consistente**.
- Luego entonces, la eficiencia final dependerá de la eficiencia de la heurística.

Cómo programar A*

Se sugiere utilizar las estructuras siguientes:

- La *lista abierta* debe ser una **cola de prioridades**.
- La *lista cerrada* puede ser una **tabla hash**.
- Un *Nodo* que contenga los elementos siguientes:
  - *Estado.* La representación de un estado en el espacio de estados.
  - *Nodo padre.* Una referencia a su nodo predecesor inmediato en el árbol de búsqueda.
  - *Acción.* La acción que, al realizarse en el estado del nodo padre produce el estado de este nodo.
  - *Costo del camino.* El costo total del camino que lleva a este nodo.
  - *Profundidad.* La profundidad de este nodo en el árbol de búsqueda.

Algoritmo

Algoritmo 2 A*

```
1:  listaAbierta ← nodo inicial                ▷ La lista abierta es una cola de prioridades sobre f(n)
2:  repeat
3:      nodoActual ← listaAbierta.pop()
4:      listaCerrada ← nodoActual
5:      if objetivo(nodoActual) then return nodoActual
6:      for all n vecino de nodoActual do
7:          if n ∉ listaCerrada then
8:              if n ∉ listaAbierta then
9:                  padre(n) ← nodoActual
10:                 n ← F, G, H
11:                 listaAbierta ← n
12:             else if G(n) < n.G_anterior then
13:                 padre(n) ← nodoActual
14:                 n ← F, G
15:                 reordenar(listaAbierta)
16: until openList = ∅
17: return Fallo
```

Propiedades de A*: Optimalidad

- Si $h(n)$ es admisible entonces $f(n)$ nunca sobre-estima el costo verdadero de una solución que pasa por n.

Teorema

A utilizado como búsqueda en un árbol es óptimo si la heurística $h(n)$ es admisible.*

Es decir, está garantizado que A* encontrará el camino más corto entre el nodo inicial y un nodo meta.

A* con y sin heurística admisible

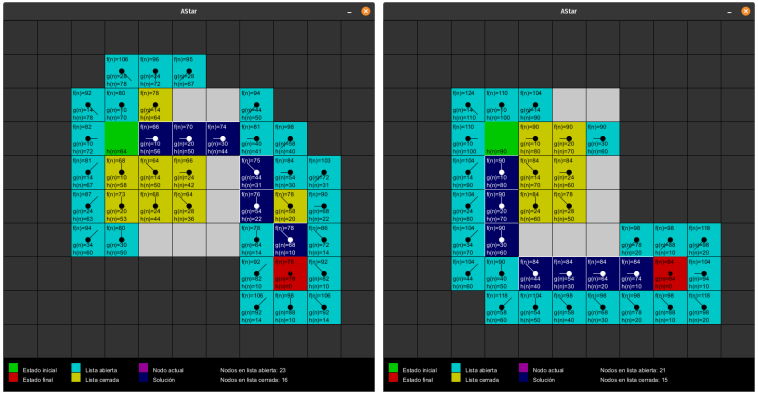


Figura: La optimalidad de la ruta depende de que la heurística sea admisible. Izquierda: heurística admisible, costo final = 78. Derecha: heurística no admisible, costo final = 84.

Propiedades de A*: Completez

- A* es completo. Si existe una solución, A* la encontrará.

Definición

Un **contorno** es un conjunto de estados que pueden ser alcanzados con un cierto costo.

- Para poder dibujar un contorno es necesario que f sea creciente.
- A* comienza en el nodo inicial, donde el valor de $f(n)$ es mínimo.
- Cada nodo expandido tiene valores de $f(n+1) \geqslant f(n)$, visitando primero a los que tienen valores más pequeños.
- Por lo tanto se visita a todos los nodos en un contorno antes de proceder al siguiente.
- Esto es una sucesión creciente, si existe una solución a una distancia finita d, eventualmente $f(n) = d$ y encontrará ese nodo.

Propiedades de A*: Eficiencia óptima

- A* es *optimamente eficiente* en términos del número de nodos expandidos dada una función heurística concreta: está garantizado que ningún otro algoritmo óptimo expandirá menos nodos que A*.
- Cualquier algoritmo que no expanda a todos los nodos con $\mathrm{f(n)} < \mathrm{C*}$, donde $\mathrm{C*}$ es el valor del camino óptimo, corre el riego de pasar de largo la solución óptima.

Propiedades de A*: Requerimientos de espacio

Sin embargo, en términos de espacio, A* consume bastante:

- El peor caso en A* tiene complejidad en tiempo y espacio de $O(b^l)$, donde b es el factor de ramificación[1] y l es la longitud del camino a la meta más cercana. Esto es crecimiento exponencial.
- Frecuentemente la frontera crece exponencialmente.

[1] El número promedio de sucesores por nodo.

Referencias I

- Ghallab, Malik, Dana Nau y Paolo Traverso (2004). *Automated Planning, Theory and Practice*. Morgan Kaufmann Publishers.
- Lester, Patrick (jul. de 2005). *A* Pathfinding for Beginners*. URL: http://homepages.abdn.ac.uk/f.guerin/pages/teaching/CS1013/practicals/aStarTutorial.htm.
- Russell, Stuart y Peter Norving (2010). *Artificial Intelligence, A Modern Approach*. Ed. por Michael Hirsch. 2a. Pearson Prentice Hall.

Licencia