Búsqueda heurística

Prof. Constantino Malagón

Búsqueda heurística

- **Los métodos de búsqueda heurística disponen de alguna información sobre la proximidad de cada estado a un estado objetivo, lo que permite explorar en primer lugar los caminos más prometedores.**
- **Son características de los métodos heurísticos:**
  - No garantizan que se encuentre una solución, aunque existan soluciones.
  - Si encuentran una solución, no se asegura que ésta tenga las mejoresas propiedades (que sea de longitud mínima o de coste óptimo).
  - En *algunas ocasiones* (que, en general, no se podrán determinar a priori), encontrarán una solución (aceptablemente buena) en un tiempo razonable.

Búsqueda heurística

- En general, los métodos heurísticos son preferibles a los métodos no informados en la solución de problemas difíciles para los que una búsqueda exhaustiva necesitaría un tiempo demasiado grande. Esto cubre prácticamente la totalidad de los problemas reales que interesan en Inteligencia Artificial.
- La información del problema concreto que estamos intentando resolver se suele expresar por medio de *heurística*s.
- El concepto de heurística es difícil de aprehender. Newell, Shaw y Simon en 1963 dieron la siguiente definición: "Un proceso que puede resolver un problema dado, pero que no ofrece ninguna garantía de que lo hará, se llama una heurística para ese problema".

Búsqueda heurística

- **Si nos planteamos seguir concretando como aprovechar la información sobre el problema en sistemas de producción, la siguiente idea consiste en concentrar toda la información heurística en una única función que se denomina *función de evaluación heurístic*a. Se trata de una función que asocia a cada estado del espacio de estados una cierta cantidad numérica que evalúa de algún modo lo prometedor que es ese estado para acceder a un estado objetivo. Habitualmente, se denota esa función por h(e).**

Función heurística

- La función heurística puede tener dos interpretaciones. Por una parte, la función puede ser una estimación de lo próximo que se encuentra el estado de un estado objetivo. Bajo esta perspectiva, los estados de menor valor heurístico son los preferidos. Pero en otros casos puede suceder que lo que convenga sea maximizar esa función.

Ejemplos de funciones heurísticas

- **Veamos ejemplos de heurísticas para algunos problemas concretos. Para el problema del 8-puzzle tenemos la siguientes heurísticas:**
  - a) La basada en la distancia Manhattan (o distancia taxi). Se asocia a cada casilla un número que es la suma de las distancias horizontal y vertical a su posición en el tablero objetivo (esto es, la suma de diferencias de sus coordenadas x e y). La función heurística es la suma de las distancias de cada una de las casillas (excluyendo la que se encuentra vacía).

	2	3
1	8	4
7	6	5

Fig. 1

$H(E_i)=2$ (1 de la casilla 1 más 1 de la casilla 8)

Ejemplos de funciones heurísticas

- b) Otra heurística, mucho más simple, consiste en contar el número de casillas que están fuera de su sitio (respecto al tablero objetivo). Es una heurística más pobre que la anterior, puesto que no usa la información relativa al esfuerzo (número de movimientos) necesario para llevar una pieza a su lugar.

- **Ejercicio**. Otra heurística posible para el 8-puzzle, si el estado objetivo es el recogido en la figura 1, es la siguiente: h(e) = 3 * seq(e), donde seq(e) cuenta 1 si hay un dígito central en e y 2 por cada dígito x no central que no es seguido (en el sentido de la agujas del reloj) por su sucesor x+1 (imponemos por convenio que 8 + 1 = 1).

  - **Calcular el valor de esta heurística para los estados que aparecen en la figura 1.**

Ejemplos de funciones heurísticas

- **El problema del viajante.**
  - **Estado inicial: un viajante se encuentra en una capital de provincia.**
  - **Estado meta: quiere viajar a otra capital por la mejor ruta posible (la más corta)**
  - **Medios: Las capitales de provincia colindantes están unidas por carreteras; se dispone de un mapa con la disposición de las provincias y sus "coordenadas" en kilómetros respecto al "centro" (por ejemplo, Madrid, con coordenadas (0,0)).**
- Una función heurística para ese problema consiste en asignar a cada estado un valor que es la distancia aérea (en línea recta) con el estado objetivo. Dicha distancia es la distancia euclídea entre las coordenadas de dos ciudades.
- Se elige una ciudad como siguiente en el camino cuando la suma de la distancia a la ciudad actual más la distancia aérea a la meta sea la menor.

Ejemplos de funciones heurísticas

- Una persona quiere viajar de Madrid a Santander considerando el mapa de carreteras de la fig. 2.

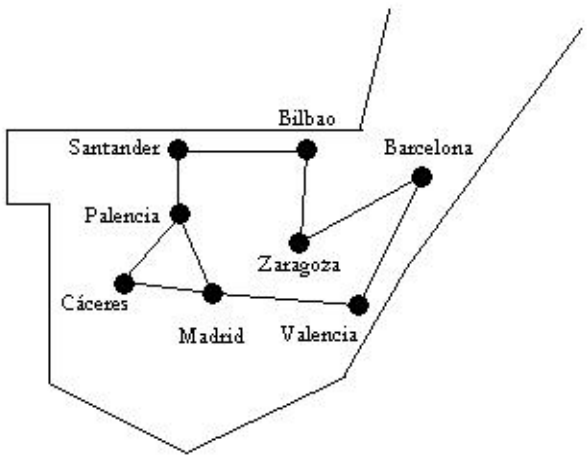


Fig. 2

Ejemplos de funciones heurísticas

- **Cuatro alternativas en la primera etapa: Cáceres, Palencia, Zaragoza y Valencia.**
  - Función heurística considerando la distancia aérea:

    Distancia (Madrid, Santander)=Distancia(Madrid, Palencia)+Distancia aérea (Palencia, Santander)

    que es más pequeña que la obtenida a través de Cáceres

    Distancia (Madrid, Santander)=Distancia(Madrid, Cáceres)+Distancia aérea (Cáceres, Santander)
  - Función heurística considerando distancias parciales:

    Distancia (Madrid, Santander)=Distancia(Madrid, Palencia)+Distancia (Palencia, Santander)

Métodos de escalada

- **Se llaman de escalada (o de ascensión a la colina) porque tratan de elegir en cada paso un estado cuyo valor heurístico sea mayor que el del estado activo en ese momento.**
- **Se dividen en dos grupos:**
  - **Los métodos irrevocables**, que no preven la vuelta a un lugar del espacio de estados si el camino resulta inadecuado.
  - **Los métodos tentativos** en los que sí podemos volver hacia atrás si prevemos que el camino elegido no es el más adecuado.

Métodos de escalada irrevocables

- La lista ABIERTA no puede poseer más de un elemento, es decir, sólo se mantiene en expectativa un único camino.
- El modo en que se determina qué estado es el que sigue a uno dado, da lugar a dos variantes de los esquemas en escalada irrevocables:
  - **La escalada simple.** En este método, en el momento es que se encuentra un estado E que es más favorable que el que se está expandiendo, dicho estado E es devuelto sin generar el resto de estados hijos.
  - **La escalada por la máxima pendiente.** En este caso, se generan todos los hijos de un estado, se calculan sus valores heurísticos y se determina uno de los estados de mejor valor heurístico; se compara dicho valor con el de el estado expandido y si es mejor, se devuelve ese estado como expansión.

Métodos de escalada irrevocables

- En ambos casos, si ningún estado hijo es mejor que el estado que está siendo expandido, no se devuelve nada, lo que conllevará que la búsqueda termine sin haber encontrado un objetivo. Nótese que la escalada simple es mucho más dependiente que la escalada por la máxima pendiente del orden de disparo de las reglas (pese a que ésta última también lo es: el valor heurístico mejor puede ser alcanzado en varios hijos y, en ese caso, el método no dice nada sobre qué estado elegir).

Algoritmo de escalada simple

- **El algoritmo:**
  - 1. Denominar *m* al estado inicial y evaluarlo. Si es estado objetivo, entonces devolverlo y terminar, si no, convertirlo en estado actual.

    Asignar *m* a una varibla llamada *elegido.*
  - 2. Repetir hasta que se encuentre solución o hasta que una iteración completa no produzca un cambio en el estado actual:
    - 2.1 Expandir *m.* Para ello,

      1) Aplicar cualquier operador aplicable al estado actual *m* y obtener un nuevo estado Ei.

      2) Evaluar Ei:

      2.1 Si Ei es estado objetivo, devolverlo y terminar.

      2.1 Si Ei no es estado objetivo no es así, evaluar si Ei es mejor que el estado actual: (H(Ei) ->H*(elegido*)), en cuyo caso hacer *m*=Ei.
    - 2.2 Si *f(elegido) # f(m)* asignar *m=elegido, y volver a* ***2.***

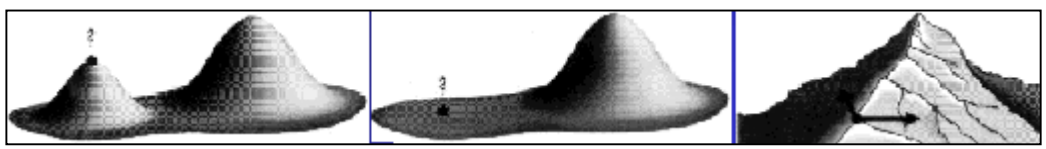
Algoritmo de escalada por la máxima pendiente

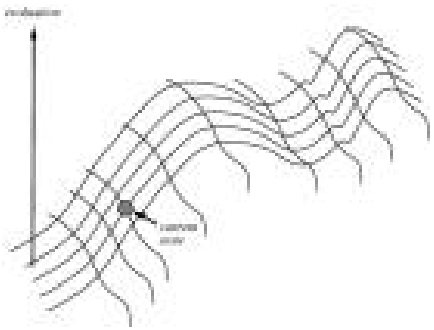
- **El algoritmo:**
  - 1. Denominar m al estado inicial y evaluarlo. Si es estado objetivo, entonces devolverlo y terminar, si no, convertirlo en estado actual.

    Asignar *m* a una varibla llamada *elegido.*
  - 2. Repetir hasta que se encuentre solución o hasta que una iteración completa no produzca un cambio en el estado actual:
    - 2.1 Expandir *m* creando el **conjunto de todos sus sucesores.**
      A. Aplicar cada operador aplicable al estado actual *m* y conseguir nuevos estados E1, ... En.
      B. Evaluar E1 .... En. Si alguno es objetivo, devolverlo y terminar Si no es así, seleccionar el mejor H(Em)
      C. Si el mejor estado Em es mejor que el estado actual, hacer que Em sea el estado actual. Volver al paso **2.**
- **Frente a la escalada simple, considera todos los posibles estados nuevos (no el primero que es mejor, como hace la escalada simple)**

Inconvenientes de los métodos de escalada

- **El proceso de búsqueda puede no encontrar una solución cuando se encuentra con:**
  - **Máximo local**: un estado que es mejor que todos sus vecinos, pero no es mejor que otros estados de otros lugares. Puesto que todos los estados vecinos son peores el proceso se queda paralizado
  - **Meseta**: los estados vecinos tienen el mismo valor. El proceso, basado en comparación local, no discrimina la mejor dirección
- El origen de estos problemas es que el método de la escalada se basa en **comparaciones locales**, no explora exhaustivamente todas las consecuencias. Frente a otros métodos menos locales, tiene la ventaja de provocar una explosión combinatoria menor
- **Soluciones parciales:**
  - Almacenar la traza para poder ir hacia atrás, hasta un estado "prometedor" (o tan bueno como el que hemos dejado)
  - Dar un gran salto para seguir la búsqueda. Útil en el caso de mesetas





Algoritmo A*

Algoritmo A*

- **Usaremos dos listas de nodos (ABIERTA Y CERRADA)**
  - **Abierta**: nodos que se han generado y a los que se les ha aplicado la función heurística, pero que aún no han sido examinados (es decir, no se han generado sus sucesores)
    - Es decir, es una cola con prioridad en la que los elementos con mayor prioridad son aquellos que tienen un valor más prometedor de la función heurística.
  - **Cerrada**: nodos que ya se han examinado. Es necesaria para ver si cuando se genera un nuevo nodo ya ha sido generado con anterioridad.

Algoritmo A*

- **Definiremos una función heurística f como la suma de dos funciones g y h:**
  - ***Función g***: es una medida del coste para ir desde el estado inicial hasta el nodo actual (suma de los costes o valores heurísticos de todos los nodos).
  - ***Función h:*** es una estimación del coste adicional necesario para alcanzar un nodo objetivo a partir del nodo actual, es decir, es una estimación de lo que me queda por recorrer hasta la meta.
  - La ***función combinada f*** una estimación del coste necesario para alcanzar un estado objetivo por el camino que se ha seguido para generar el nodo actual (si se puede generar por varios caminos el algoritmo se queda con el mejor).
  - **NOTA:** los nodos buenos deben poseer valores bajos.

Algoritmo A*

- En cada paso se selecciona el nodo más prometedor que se haya generado hasta ese momento (función heurística).
- A continuación se expande el nodo elegido generando todos sus sucesores.
- Si alguna de ellos es meta el proceso acaba. Si no continúo con el algoritmo.
- Es parecido a la búsqueda en profundidad, pero si en una rama por la que estoy explorando no aparece la solución la rama puede parecer menos prometedora que otras por encima de ella y que se habían ignorado. Podemos pues abandonar la rama anterior y explorar la nueva.
- Sin embargo al vieja rama no se olvida. Su último nodo se almacena en el conjunto de nodos generados pero aún sin expandir.

Algoritmo A*

1. Empezar con ABIERTA conteniendo sólo el nodo inicial. Poner el valor g de ese nodo a 0, su valor h al que corresponda, y su valor f a h+0, es decir, a h.

2. Inicializar CERRADA como una lista vacía.

3. Hasta que se encuentre una meta o de devuelva fallo realizar las siguientes acciones:

3.1 Si ABIERTA está vacía terminar con fallo; en caso contrario continuar.

3.2 Eliminar el nodo de ABIERTA que tenga un valor mínimo de *f*; llamar a este nodo *m* e introducirlo en la lista cerrada.

3.3 Si *m* es meta, abandonar el proceso iterativo iniciado en 2 devolviendo el camino recorrido (punteros a sus antepasados)

3.4 En caso contrario expandir *m* generando todos sus sucesores.

Algoritmo A*

3.5 Para cada sucesor *n'* de *m*:

1) Crear un puntero de *n'* a *m*.

2) Calcular g(n')=g(m)+c(m,n'), tal que c(a,b) es el coste de pasar de *a* a *b*.

3) Si *n'* está en ABIERTA llamar *n* al nodo encontrado en dicha lista, añadirlo a los sucesores de *m* y realizar el siguiente paso:

3.1) Si g(n')<g(n), entonces redirigir el puntero de *n* a *m* y cambiar el camino de menor coste encontrado a n desde la raíz; *g(n)=g(n') y f(n)=g(n')+h(n).*

Algoritmo A*

4) Si *n'* no cumple 3), comprobar si está en cerrada; llamar *n* al nodo encontrado en dicha lista y realizar las siguientes acciones:

Si 3.1) no se cumple, abandonar 4); en caso contrario propagar el nuevo menor coste *g(n')* (por lo que también actualizarán los valores de *f* correspondientes(que llamaremos n_i tal que i=1,2,…, siendo sus costes anteriores g(n_i)), realizando un *recorrido en profundidad* de éstos, empezando en *n'* y teniendo en cuenta las siguientes consideraciones:

4.1) Para los nodos descendientes n_i cuyo puntero (que debe apuntar siempre al mejor predecesor hasta ese momento) conduzca hacia el nodo n_i , actualizar $g(n_i)$=g(n_i') y f(n_i)=g(n_i')+h(n_i) y seguir el recorrido hasta que se encuentre un n_i que no tenga más sucesores calculados o se llegue a un nodo en que ya ocurra que $g(n_i)$=$g(n_i$'), en cuyo caso se habría producido un ciclo y también habría que terminar la propagación.

Algoritmo A*

4.2) Para los nodos descendientes n_i cuyo puntero no conduzca hacia el nodo *n',* comprobar si $g(n_i\text{ ' }) < g(n_i\ )$, en cuyo caso se debe actualizar el puntero para que conduzca hacia el nodo n' (mejor camino desde la raíz encontrado hasta ese momento) y se continúa el proceso de propagación.

5) Si n´no está en ABIERTA o en CERRADA, calcular *h(n')* y *f(n')=g(n')+h(n')*, introducirlo en ABIERTA y añadirlo a la lista de sucesores de *m.*

Algoritmo A*

- **Relaciones con otros métodos**
  - Si para todo *n* y para todo *m h(n)=0* y *c(m,n)*=1, es decir, el coste para pasar de m a n es 1 entonces A* se convierte en un proceso de búsqueda en amplitud.
- **Características**
  - Es un método completo de búsqueda, es decir, termina encontrando la solución cuando ésta exista para cualquier tipo de grafos.
  - Es **admisible**. Es decir, no sólo encuentra la solución sino que la que encuentra es la optima si se cumple la siguiente condición:
    - **Para todo *n* h(n) es menor o igual que h*(n), es decir, si la función heurística que estima la distancia a la meta nunca puede superar la distancia real existente entonces A* garantiza encontrar la solución óptima.**