Sistemas Inteligentes

PRÁCTICA 1 – BÚSQUEDA Y SISTEMAS PROBABILÍSTICOS

JAVIER RIVILLA ARREDONDO
SISTEMAS INTELIGENTES | JRA48@ALU.UA.ES

INDICE

Práctica 1 - Sistemas Inteligentes

Búsqueda y sistemas probabilísticos

Introducción

En esta práctica se desarrollará un sistema con el principal objetivo de guiar al robot a la meta. Este camino que tendremos que escoger deberá ser el camino más corto desde el punto de partida hasta la meta. Calculando así los caminos óptimos con el famoso algoritmo A*, y posteriormente se utilizará un sistema experto difuso para guiar al robot hasta la meta.

Algoritmo A*

Su funcionamiento es, por definición bastante sencillo: **es un algoritmo de búsqueda en grafos,** con lo que nosotros emplearemos el algoritmo A* para calcular el camino de menor coste entre un nodo origen y un nodo objetivo.

Para este algoritmo necesitaremos:

- Un nodo o punto inicial
- Un nodo final que representa el final del algoritmo
- Un método para identificar que nodos son traspasables y cuales son sólidos
- Un método para determinar el costo directo (g) de moverse entre los nodos
- Un método para determinar el costo indirecto (h)

- Una lista de nodos abiertos, en esta lista se guardarán todos los nodos que se han identificado como posibles movimientos, pero aún no han sido evaluados
- Una lista de nodos cerrados, donde se guardarán todos los nodos evaluados y descartados, aunque no es necesario una lista, basta con un estado que indique que el nodo se encuentra cerrado
- Una forma de identificar que nodo procede a otro, para poder retornar la cadena de los nodos

En pseudo-código, el algoritmo es el siguiente:

```
Alg A*
      listaInterior = vacio
      listaFrontera = inicio

      mientras listaFrontera no esté vacía

            n = obtener nodo de listaFrontera con menor f(n) = g(n) + h(n)

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

            listaFrontera.del(n)
            listaInterior.add(n)


            para cada hijo m de n que no esté en lista interior
                  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
      Error, no se encuentra solución
falg
```

Uno de los principales problemas que tienen algunos algoritmos es que se guía exclusivamente por la función heurística, la cual podría no indicar el camino más bajo.

La implementación anterior realizada en java:

```java
//Calcula el A*
public int AEstrella(){
    int resultado = 0;
    TNodo nodo;
    boolean end = false;
    ArrayList<TNodo> vacio = new ArrayList<TNodo>();
    Comparator<TNodo> comparator = new TNodoComparator();
    PriorityQueue<TNodo> auxinicio = new PriorityQueue<TNodo>(10, comparator);
    ArrayList<TNodo> intermedios = new ArrayList<TNodo>();
    ArrayList<TNodo> listaInterior = vacio;
    PriorityQueue<TNodo> listaFrontera = auxinicio; //<--------Cola de prioridad

    //El algoritmo empieza
    listaFrontera.add(inicio);
    while(listaFrontera.isEmpty() == false && end == false){

        nodo = listaFrontera.poll();
        listaInterior.add(nodo);

        if(nodo.equals(fin)){ //Si el nodo llega al fin terminamos
            end = true;
            resultado = 0;
        }
        else{
            //expando los nodos.
            intermedios.addAll(nodo.expandir(mundo));
            //una vez visitados, filtro los nodos.
            filtrarVisitados(intermedios, listaInterior);
            filtrarFrontera(listaFrontera, intermedios);
            listaFrontera.addAll(intermedios);
            intermedios.clear();
        }
    }
```

El funcionamiento del código es el siguiente:

1. Primero, creamos la **lista interior** y la **lista frontera:**
    - Creamos la lista interior a **vacío** y la lista frontera la colocamos en el **inicio**.
    - Si la **lista frontera** no está vacía y el nodo no llega al final desencolamos la lista frontera, si esta no existe nos devolverá **null.**
    - Añadiremos el nodo actual a la lista interior y compararemos el nodo desencolado de la **lista frontera** para saber si este llega a su fin.
    - En el caso de que llega a su fin el resultado será 0 obviamente, pero si no pues expandiremos los nodos y una vez visitados filtraremos los nodos y añadiremos todos en la lista frontera.
2. Después, construiremos el nodo que habrá llegado bien y escribiré tanto el camino como el orden de expansión de los nodos.

```java
    //Aquí construyo el nodo que habrá llegado correctamente.
    fin = listaInterior.get(listaInterior.size()-1);
    totalVisitados = listaInterior.size();


    escribirCamino();
    escribirOrden(listaInterior);
    imprimirCamino();
    imprimirExpandidos();

    return resultado;
}
```

Función de evaluación

Para que el algoritmo A* funcione correctamente, **es necesario disponer de una función que nos indique,** de alguna forma el estado actual del recorrido. A esa función nosotros la vamos a llamar heurística (utilizando manhattan).

Para ello debemos tener en cuenta que el recorrido que hagamos será a veces mejor y otras veces peor.

Para comprobar el funcionamiento de la propuesta:

```
20
10
10
* * * * * * * * * * * * * * * * * * * *
* . * . . . . . . . . . . . . . . . . *
* . * . . . . . . . . . . . . . . . . *
* . * . . . . . . . . . . - . . . . . *
* . * . . . . . . . . . . . . . . . . *
* . * . . . . . . . . . . . . . . . . *
* - * . . . . . . . . . . . . . . . . *
* . * . . . . . . . . . . . . . . . . *
* . * . . . . . . . . . . . . . . . . *
* . * * * * * . . . . . . . . . . . . *
* . . . . . * . . . . . . . . . . . . *
* . . . . . * . . . . . . . . . . . . *
* . . . . . * . . . . . . . . . . . . *
* . . . . . * . . . . . . . . . . . . *
* . . . . . * * * * * * . . . . . . . *
* . . . . . . . . . . . . . . . . . . *
* . . - . . . . . . . . . . . . . . . *
* . . . . . . . . . . . . . . . . . . *
* . . . . . . . . . . . . . . . . . . *
* * * * * * * * * * * * * * * * * * * *
```

Le metemos un fichero .**txt** llamado “mundo” que será el mundo creado y al ejecutarlo nos dará el número de nodos expandidos, y el recorrido que ha hecho nuestro robot, como observaremos en las siguientes imágenes:

```
Camino: (Cuya longitud es: 27)
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. X X X X X . . . . . X X X X X X X .
. . . . . X . . . . . X . . . . . .
. . . . . X . . . . . X . . . . . .
. . . . . X . . . . . X . . . . . .
. . . . . X . . . . . X . . . . . .
. . . . . X X X X X X X . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
```

Ese es el recorrido de nuestro robot, llegando al destino y nos explora los siguientes nodos:

```
Nodos explorados: (en total 50)
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 18 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 16 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1  8 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1  0  1  2  3  4 -1 -1 -1 -1 -1 -1 44 45 46 47 48 49 50 -1
-1 10  9  7  5  6 -1 -1 -1 -1 -1 -1 43 -1 -1 -1 -1 -1 -1 -1
-1 12 11 13 14 15 -1 -1 -1 -1 -1 -1 42 -1 -1 -1 -1 -1 -1 -1
-1 20 21 19 22 17 -1 -1 -1 -1 -1 -1 41 -1 -1 -1 -1 -1 -1 -1
-1 27 23 25 24 26 -1 -1 -1 -1 -1 -1 40 -1 -1 -1 -1 -1 -1 -1
-1 29 31 33 30 28 32 34 35 36 37 38 39 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Comparativa de las heurísticas

Como bien hemos visto anteriormente, el algoritmo estrella se encarga de escoger el camino más óptimo, pero bien tenemos que recurrir a una heurística para poder hacer funcionar nuestro algoritmo A* y la pedida es la de **Manhattan**. Como bien sabemos, tenemos diferentes tipos de heurísticas:

- **Diagonal**: Podemos tomar las diagonales para optar al camino más corto.
- **Distancia Euclídea:** Es la distancia entre dos puntos de un espacio.
- **Exhaustiva:** A diferencia de las demás, esta necesitaría un tiempo más grande.
- **Manhattan:** Se asocia a cada casilla un número que es la suma de las distancias horizontal y vertical.

Como podemos comprobar anteriormente tenemos un total de **50 nodos explorados** por la heurística **manhattan**, ahora bien, comprobaremos alguna más en mi caso he trabajo la diagonal porque es una de las que más cambia respecto a las demás por que digamos que hace un atajo y así las diferencias iba a ser mucho mayores.

Y así comprobando podemos ver la siguiente comparativa:

```
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. X X X X X . . . . . . . . . . . X X .
. . . . . X . . . . . . . . . . X X . .
. . . . . X . . . . . . . . . X X . . .
. . . . . X . . . . . . . . X X . . . .
. . . . . X . . . . . . . X X . . . . .
. . . . . X X X X X X X X X . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .

Nodos explorados: (en total 30)
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1  0  1  2  3  4 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 29 30 -1
-1 -1 -1 11  7  5 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 27 28 -1 -1
-1 -1 -1 -1 10  6 -1 -1 -1 -1 -1 -1 -1 -1 -1 25 26 -1 -1 -1
-1 -1 -1 -1 -1  8 -1 -1 -1 -1 -1 -1 -1 -1 23 24 -1 -1 -1 -1
-1 -1 -1 -1 -1  9 -1 -1 -1 -1 -1 -1 -1 21 22 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 12 13 14 15 16 17 18 19 20 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Como podemos observar los nodos explorados de la diagonal son 20 menos:

Heurística tratada	Nodos explorados
Manhattan	50
Diagonal	30

En mi opinión, la heurística más idónea es la Heurística Diagonal, puesto recorre el mismo recorrido que la Manhattan explorando menos nodos y a su vez tarda menos tiempo.

Si bien es cierto que las demás no las he comprobado como para decir que esta es mejor, pero según teoría la Heurística Exhaustiva necesita mucho más tiempo para desarrollarse (para un problema como este no necesitamos tanto tiempo).

En cuanto a h y h*

Debemos saber que la función del A*:

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

Pueden ocurrir otros casos, entre estos:

- Si h'(x)=0, la función g(x) controla la búsqueda.
- Si h'(x)=0 y g(x)=0 la búsqueda obviamente será aleatoria.
- Si h'(x)=0 y g(x)=1 la búsqueda será primero en anchura.
- Si h'(x) nunca sobrestima a h(x), se garantiza encontrar el camino óptimo, pero se desperdicia esfuerzo explorando rutas innecesarias.
- Si h'(x) sobrestima a h(x), no puede garantizarse obtener el camino del menor coste.

Con esto quiero concluir que, aunque un camino parezca más optimo que otro no debe de serlo, y que optimizar algo completamente es muy difícil casi por no decir que nunca se dan esos casos por eso suelen ser aproximaciones.

Explicación del Fuzzy Controller

El Fuzzy Controller es el debe realizar la función de evitar los obstáculos utilizando los sensores del robot siguiendo obviamente nuestro algoritmo implementado en nuestro caso A*.

Los sensores que tenemos se declaran de la siguiente manera:

```
//EJEMPLO de definición NO VINCULANTE. Hacer lo mismo para los 7 sensores restantes
FUZZIFY s0
    TERM near   := (0,1) (0.5,0.7) (1,0.25) (1.5,0);
    TERM med    := (0.25,0) (0.75,1) (1.25,0.5) (1.5,0);
    TERM far    := (0,0) (1.5,1);
END_FUZZIFY

FUZZIFY s1
    TERM near   := (0,1) (0.5,0.5) (1,0.25) (1.5,0);
    TERM med    := (0.25,0) (0.75,1) (1.25,0) (1.5,0);
    TERM far    := (0,0) (1.5,1);
END_FUZZIFY

FUZZIFY s2
    TERM near   := (0,1) (0.5,0.5) (1,0.25) (1.5,0);
    TERM med    := (0.25,0) (0.75,1) (1.25,0) (1.5,0);
    TERM far    := (0,0) (1.5,1);
END_FUZZIFY

FUZZIFY s3
    TERM near   := (0,1) (0.5,0.5) (1,0.25) (1.5,0);
    TERM med    := (0.25,0) (0.75,1) (1.25,0) (1.5,0);
    TERM far    := (0,0) (1.5,1);
END_FUZZIFY

FUZZIFY s4
    TERM near   := (0,1) (0.5,0.5) (1,0.25) (1.5,0);
    TERM med    := (0.25,0) (0.75,1) (1.25,0) (1.5,0);
    TERM far    := (0,0) (1.5,1);
END FUZZIFY
```

Así hasta el s8, que representan las lecturas de los sonares.

Y luego tenemos lo que nos indicar girar el robot (los grados) que se llama **sig.**

```
FUZZIFY sig
    TERM rightHe   :=    (-90,1) (-75,0.5) (-35,0);
    TERM rightM    :=    (-75,0)(-35,1) (-10,0);
    TERM recto  :=    (-10,0) (0,1) (10,0);
    TERM leftM    :=    (10,0) (35,1) (75,0);
    TERM leftHe   :=    (35,0) (75,0.5) (90,1);
END_FUZZIFY
```

Pues mediante esto, nosotros les vamos a dar una serie de reglas para que nuestro robot pueda conseguir llegar a la meta. Las reglas son las siguientes:

```
//Definir las reglas del SE
RULE 1: IF s0 IS far THEN vel IS fast;
RULE 2: IF s8 IS near THEN vel is slow;
RULE 3: IF s0 IS near THEN vel IS slow;
RULE 4: IF s7 IS near THEN rot IS izqm;
RULE 5: IF s0 IS med THEN vel IS med;
RULE 6: IF s7 is near THEN vel is slow;
RULE 7: IF s1 IS med THEN vel IS slow;
RULE 8: IF s2 IS near THEN rot IS derm;
RULE 9: IF s8 IS near THEN rot IS izqm;
RULE 10: IF s1 IS med THEN rot IS derm;
RULE 11: IF s8 IS med THEN rot IS izqm;
RULE 12: IF s1 IS near THEN vel IS slow;
RULE 13: IF s1 IS near THEN rot IS derm;
RULE 14: IF sig IS rightM THEN rot is der;
RULE 15: IF sig IS rightHe THEN rot is derm;
RULE 16: IF sig IS leftHe THEN rot IS izqm;
RULE 17: IF sig IS leftM THEN rot is izq;
RULE 18: IF sig IS recto THEN rot is cen;
```

Por ejemplo, la **regla 1:**

Si el s0 del robot esta lejos entonces incremente la velocidad (m/s).

Regla 2: Si el s8 está cerca entonces disminuya la velocidad.

Y así sucesivamente las demás. Con estas reglas por así decirlo guiaremos al robot a su meta, evitando los posibles obstáculos este en el mundo que esté.