



Primera practica

***Búsqueda Informada en Proyectos de Pac-Man***



Jordi Blanco Lopez

NIF:20998

Curso: 2016/2017

En dicha práctica trabajaremos sobre los algoritmos de búsqueda trabajados en clase implementándolos al proyecto de Pac-Man del curso introductorio de inteligencia artificial (CS 188) de UC Berkeley. Para ser más concretos trabajaremos sobre tres algoritmos de búsqueda informada (UCS, A\*, Best-H).

# Algoritmos de búsqueda informada

Lo primero que se nos pide es implementar los algoritmos de búsqueda informada estudiados en clase, con una serie de observaciones a tener en cuenta:

* Considerar el que el coste de transición de un estado a otro es 1.
* Utilizar como heurísticas distancia de Manhattan y distancia Euclidean.
* Siempre que sea posible dar la versión optima del algoritmo.

De forma correcta hemos implementado los siguientes algoritmos:

- Búsqueda con coste uniforme (UCS)

- Búsqueda A\* (A\*)

- Búsqueda “primero el mejor” (Best-H)

Como opcional debemos implementar la búsqueda bidireccional (BDS).

Estos tres algoritmos para implementar-los, decidimos juntarlos en una misma función de búsqueda en la que irán variando los parámetros a usar para asignar costes o heurísticas.

Para la búsqueda con coste uniforme (UCS) se desactiva el uso de heurística manteniendo únicamente el coste del problema, se realiza esta acción utilizando una heurística nula ya definida la cual nos devolverá siempre 0.

En el caso de la búsqueda A\* (A\*) utilizamos la función con todos sus parámetros activos (la heurística que decidamos usar y el coste por defecto).

Finalmente, para la búsqueda “primero el mejor” (Best-H) se determinan los costes a 0 dejando únicamente como guía los valores de la heurística proporcionada.

En los algoritmos con uso de heurísticas se implementa el mecanismo de Path-max para trabajar fortalecer la heurística y que fuera monótona.

El mecanismo de almacenamiento en la frontera que he decidido ha sido utilizar una cola de prioridad ya proporcionada por el ejercicio.

Para la búsqueda bidireccional (BDS) se desarrollan dos algoritmos BFS, uno comenzando desde el inicio y otro desde el objetivo final.

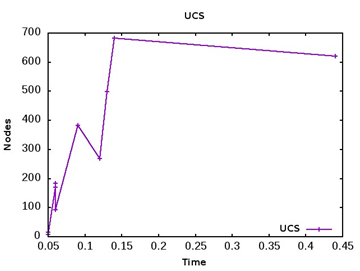
La estrategia de búsqueda es ir expandiendo nodos hasta que ambos caminos se junten, para ello se comprueban si los nodos que actualmente están evaluando se encuentran ya en visitados por el otro algoritmo que está ejecutando a su vez.

Para que el resultado sea satisfactorio hemos de tener en cuenta que el resultado del algoritmo que comienza desde el final está en estado “espejo”, por lo que deberemos cambiar el sentido de todas las acciones tomadas.

El algoritmo ha demostrado un funcionamiento correcto en los mapas proporcionados y en los creados por nosotros.

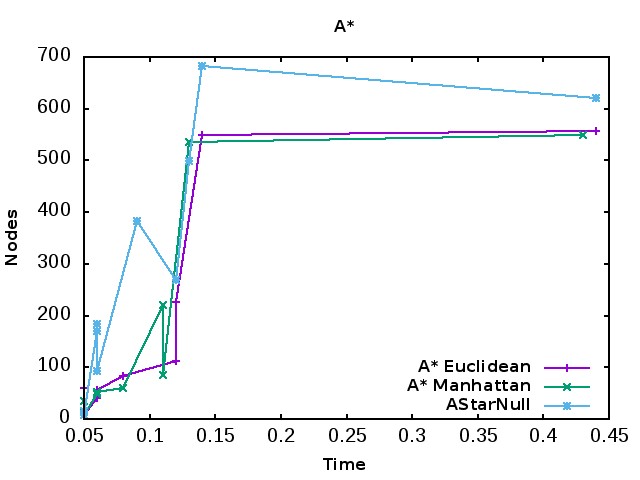
|  |  |  |
| --- | --- | --- |
| UCS | | |
| Mapa | Time | Nodos |
| Amaze | **0.06** | 183 |
| bigmaze | **0.44** | 620 |
| bmaze | **0.09** | 383 |
| Cmaze | **0.13** | 498 |
| contourMaze | **0.06** | 170 |
| mediumMaze | **0.12** | 269 |
| openMaze | **0.14** | 682 |
| smallMaze | **0.06** | 92 |
| testMaze | **0.05** | 7 |
| tinyMaze | **0.05** | 15 |

# Evaluación Experimental



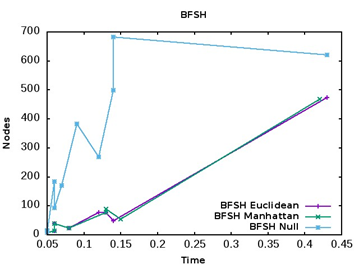
En los mapas de tamaño reducido, la variación es mínima comparado a los demás algoritmos, pero cuando aumenta el número de estados a analizar, la cantidad de nodos expandidos se dispara con el UCS. Podemos observar una gran diferencia en mapas como el Cmaze (Mapa propio) y openMaze respecto al uso del BFSH.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A\*** | **Euclidean** | | **Manhattan** | |
| **Mapa** | **Time** | **Nodos** | **Time** | **Nodos** |
| Amaze | **0.06** | 41 | **0.05** | 34 |
| bigmaze | **0.44** | 557 | **0.43** | 549 |
| bmaze | **0.08** | 83 | **0.08** | 60 |
| Cmaze | **0.12** | 113 | **0.11** | 85 |
| contourMaze | **0.05** | 60 | **0.06** | 49 |
| mediumMaze | **0.12** | 226 | **0.11** | 221 |
| openMaze | **0.14** | 550 | **0.13** | 535 |
| smallMaze | **0.06** | 56 | **0.06** | 53 |
| testMaze | **0.05** | 7 | **0.05** | 7 |
| tinyMaze | **0.05** | 13 | **0.05** | 14 |



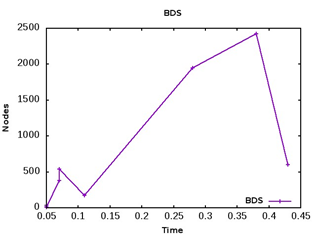
Podemos observar que el tiempo de ejecución del algoritmo con distintas heurísticas tiene una variación mínima, igual que con su cantidad de nodos expandidos, donde únicamente en algún mapa concreto podemos observar la ventaja de Manhattan sobre la Euclidean.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BFSH | Manhattan | | Euclidean | |
| Mapa | Time | Nodos | Time | Nodos |
| Amaze | **0.06** | 14 | **0.06** | 14 |
| bigmaze | **0.42** | 468 | **0.43** | 473 |
| bmaze | **0.08** | 24 | **0.08** | 24 |
| Cmaze | **0.15** | 55 | **0.14** | 49 |
| contourMaze | **0.05** | 13 | **0.05** | 13 |
| mediumMaze | **0.13** | 78 | **0.12** | 78 |
| openMaze | **0.13** | 89 | **0.13** | 77 |
| smallMaze | **0.06** | 39 | **0.06** | 39 |
| testMaze | **0.05** | 7 | **0.05** | 7 |
| tinyMaze | **0.05** | 8 | **0.05** | 8 |



Podemos observar que la variación es mínima entre el uso de una heurística u otra, esto se observa claramente en la grafica, ya que ambas líneas están muy juntas en todo momento. Tambien podemos observar la gran mejora con el uso de heurística , ya que es muy evidente a simple vista, en el caso el cual el algoritmo trabaja sin heurística, dicho algoritmo se comporta como un UCS, se observa claramente observando dicha grafica con la del UCS.

|  |  |  |
| --- | --- | --- |
| Bidireccional | | |
| Mapa | Time | Nodos |
| Amaze | **0.07** | 380 |
| Bigmaze | **0.43** | 600 |
| Bmaze | **0.38** | 2424 |
| Cmaze | **0.28** | 1946 |
| contourMaze | **0.07** | 540 |
| mediumMaze | **0.11** | 176 |
| openMaze | **-** | - |
| smallMaze | **0.05** | 34 |
| testMaze | **0.05** | 8 |
| tinyMaze | **0.05** | 14 |



Datos relativos al funcionamiento correcto del algoritmo: La cantidad de nodos expandidos aumenta de manera gigantesca para algunos tipos de mapa como es el Bmaze y el Cmaze, ambos mapas generados aleatoriamente por el generador de mapas. Para la resta de mapas el comportamiento es parecido a los demás algoritmos analizados anteriormente.

# Problema bien definido

Sobre el problema *Blocks World Problem,* podemos observar que como parámetros principales disponemos de: Estado Inicial y Estado Final que vienen ya definidos en el formato que se solicita. Definimos estos parámetros debido a que no se especifica ningún estado fijo como inicial o final. También se añaden los limites de cada elemento, es decir, su longitud (M), altura (C) y bloques(K).

Ejemplo de un estado:

**K** Bloques = 5 (A,B,C,D, E) // **M** Posiciones = 6 //  **C** Altura Maxima = 2

[ [ A ] , [ B, C ] , [D ] , [ ] , [ ] , [E ] ]

El objetivo de dicho problema no es otro que obtener un estado idéntico al introducido como objetivo final en los parámetros del inicio del problema. Los costes de cada paso serán 1 por cada posición que se desee mover.

Si tomamos el ejemplo de arriba, mover el bloque “A” a la última posición donde se encuentra el bloque “E”, tendría un coste de 5.

La acción definida es ***Move*** en la cual indicamos desde que posición movemos hasta que posición queremos dejar el bloque. Como podemos observar, no especificamos el bloque ya que de esta forma es mucho más sencillo, este control si se habrá de tener a la hora de aplicar sobre este problema un algoritmo.

move(2,1,state2)

Estado Final

**[ [B] , [A] , [ ] ]**

**[[B] , [] , [ A ] ]**

**[[ ] , [B] , [ A ] ]**

move(1,0,state1)

move(0,2, state0)

Estado Inicial

**[ [A] , [B] , [ ] ]**